# Agile Business Process Management

Concepts and Tools for Long-running Autonomous Business Processes

Dissertation zur Erlangung des Doktorgrades
an der Fakultät für Mathematik, Informatik und Naturwissenschaften
Fachbereich Informatik
der Universität Hamburg
vorgelegt von Kai Jander
Hamburg, 2016

Tag der Disputation: 29.04.2016

Die folgenden Gutachter haben diese Dissertation zur Annahme empfohlen:
Prof. Dr. Winfried Lamersdorf, Universität Hamburg
Prof. Dr. Tilo Böhmann, Universität Hamburg
Prof. Dr. Ingo J. Timm, Universität Trier

## Abstract

Business Process Management is a widespread approach for structuring actions and labor within organizations and covers a multitude of social, organizational and technological aspects with the goal of improving efficiency and manageability of organizations. Concepts for production and administrative business process is highly developed and such business processes tend to be well-supported with tools and systems provided by the market. Such processes are often highly structured with a fairly limited number of variations in their execution and are executed many times, often in parallel, in order to manufacture a product or provide a service. The lifetime of such process instances is often very limited.

This type of highly structured mode of operation with high rigidity is very different from another set of business processes that are increasingly common within many organizations: These processes are collaborative business processes, where a group of participants engage in a creative process such as product development. Collaborative processes are characterized by very long execution times in the scale of years, a high degree of expected variation due to changes becoming necessary during execution and workflow participants requiring a high degree of autonomy to perform their creative tasks and applying their expert knowledge. This kind of process is difficult to model using traditional business process management toolsets aimed at production processes and is therefore often organized in an informal fashion.

For these types of processes, flexibility, the ability to react to changes, and agility, the preparedness for future changes, becomes important issues. This work introduces an agile approach for business process management based on goal-oriented business process models and distributed workflow management which aims to provide better support for this type of business processes by focusing on their self-organizing and collaborative properties. The approach addresses both conceptual and technical aspects by directly using business goals as model elements in business process models. This goal-oriented modeling allows for a high degree of execution flexibility, agility and, for the workflow participants, autonomy while maintaining a strong link to the strategic business goals of processes. The distributed workflow management not only enables technical execution of such processes but also allows for high flexibility and agility in terms of organizational structures and structural changes before and during execution. The distributed workflow management also enables the purposeful addition of system redundancy for a high level of robustness over the long execution time of the processes.

## Kurzzusammenfassung

Geschäftsprozessmanagement ist ein verbreiteter Ansatz um Tätigkeiten und Arbeit innerhalb von Organisationen zu strukturieren und umfasst eine Vielzahl an sozialen, organisatorischen und technologischen Aspekten mit dem Ziel, die Effizienz und Handhabbarkeit von Organisationen zu verbessern. Konzepte bezüglich Produktionsprozessen und administrativen Prozessen ist sehr ausgereift und daher werden diese Art von Geschäftsprozessen im Allgemeinen durch am Markt erhältliche Werkzeuge und Systeme gut unterstützt. Dieser Art der Prozesse sind meistens stark strukturiert mit einer begrenzten Anzahl von Ausführungsvarianten und werden viele Male, oftmals auch parallel, ausgeführt um ein Produkt herzustellen oder eine Dienstleistung bereitzustellen. Die Lebensdauer solcher Prozessinstanzen ist in vielen Fällen sehr kurz.

Dieser Typ von stark strukturiertem und sehr rigiden Geschäftsprozess unterscheidet sich sehr stark von einer anderen Menge an Geschäftsprozessen, welche in zunehmenden Maße in vielen Organisationen zu finden ist: Diese Prozesse sind kollaborative Geschäftsprozesse, in welchen die Teilnehmer sich in einem kreativen Prozess wie etwa der Produktentwicklung betätigen. Kollaborative Geschäftsprozesse sind geprägt durch sehr lange Ausführungszeiten welche Jahre umfassen kann, ein hohes Maß an erwarteten Varianten durch Änderungen welche während der Ausführungszeit notwendig werden und Prozessteilnehmer, welche einen hohen Grad an Autonomie zur Durchführung ihrer kreativen Tätigkeiten und Anwendung ihres Expertenwissens benötigen. Diese Art von Prozess ist mit herkömmlichen Werkzeugen des Geschäftsprozessmanagements, welche vor allem die Produktionsprozesse im Fokus haben, nur schwer zu modellieren und werden daher häufig informell organisiert.

Für diese Art von Prozessen werden Flexibilität, die Fähigkeit zur Anpassung an Änderungen, sowie Agilität, die Vorbereitung auf zukünftige Änderungen, zu wichtigen Belangen. Diese Arbeit führt einen agilen Ansatz für das Geschäftsprozessmanagement basierend auf zielorienterter Geschäftsprozessmodellierung und verteiltem Workflow-Management ein, welcher zum Ziel hat, die Unterstützung für diese Art von Geschäftsprozessen durch Fokussierung auf ihre selbstorganisierenden und kollaborativen Eigenschaften zu verbessern. Der Ansatz umfasst sowohl konzeptuelle als auch technische Aspekte durch die direkte Verwendung von Geschäftszielen als Modellelemente in Geschäftsprozessmodellen. Dieses zieloriente Modellieren erlaubt einen hohen Grad an Flexibilität, Agilität und, für die Geschäftsprozessteilnehmer, Autonomie unter Beibehaltung einer starken Bindung an die strategischen Geschäftsziele von Prozessen. Das verteilte Workflow-Management ermöglicht nicht nur die technische Ausführung solcher Prozesse, sondern erlaubt ein hohes Maß an Flexibilität und Agilität in Bezug auf organisatorische Strukturen und strukturelle Änderungen sowohl vor als auch während der Ausführung. Das verteilte Workflow-Management ermöglicht ebenfalls das gezielte Hinzufügen von Systemredundanz für eine hohes Niveau an Robustheit während der langen Ausführungszeit der Prozesse.

## Acknowledgements

This work is the results of cross-disciplinary research that was only possible in the conducive research environment created, against all external and internal odds, by a large number of people involved in it, for many of whom I will not be able to personally express my gratitude here.

Nevertheless, I would like to express my explicit and deep gratitude for a number of key individuals that made this research and journey possible.

First, I would like to honor and acknowledge my colleagues Dr. Lars Braubach and Dr. Alexander Pokahr for the fantastic research collaboration and highly productive work and exchange, which was enabled and furthered by the freedom and support granted by my advisor Prof. Winfried Lamersdorf.

In addition, I would also express my gratitude towards my colleagues at Daimler AG for letting me have a peek into the structure and organization of a large corporation as well as the collaborative work we were able to conduct, in particular, my gratitude goes towards Birgit Burmeister for her initial work and her unrelenting support for this project as well as Birgit Leißler for the hands-on experience of a real-world collaborative process.

Finally, on a personal level, my sincere gratitude belongs to Meg Phillips for supporting me personally, enduring frustrations and joyfully sharing the progress of this project over the years.

# Contents

# Chapter 1

# Introduction

Due to its practical benefits, *business process management (BPM)* is a topic of intense research interest, consisting of a combination of methods, best practices and technology which aim to improve organizational performance. BPM touches a wide area of research, applications and multiple diverse communities, especially business administration and computer science (see [161]). It includes topics as diverse as supply chain management, organizational structures and production automation. Nevertheless, a core element focuses on the assessment of processes used by an organization and using information technology to partially automate processes, monitor their parameters during execution, then use monitoring data to correct and improve the processes.

Businesses increasingly rely on the use of such fully-automated and semi-automated processes (*workflows*), which increasingly assist with control and management of various aspects of the business itself. As a result, workflows and business process management are a key factor for the success of an organization and have resulted in a large landscape of various systems for modelling, simulating, executing and monitoring workflows.

However, in many cases, business process management has focused on a particular set of business processes in organizations which are both highly repeatable and where each business process instance has a short lifespan before it is finished. Despite this, many business processes in organizations such as research and development do not fall into this mold. These type of processes are typically very long running and require a certain flexibility due to unexpected and unforeseeable issues occurring. The next section will give an overview of this problem, followed by an introduction of the types of business processes which this work aims to address along with the concepts needed to approach them.

## 1.1 Traditional Focus of Business Process Management and Agile Business Process Management

Business process management, while aiming to address all types of business processes in an organization, often focuses on a specific set of business processes for which an abundance of literature, methodologies and software is available to support them. In order to understand this set of business processes, Weske offers a few degrees for business process to help catogerize them (see [161]):

- *Degree of Repetition*: The degree of repetition describes the number of times a process is repeated within an organization. The process to produce typical mass products such as office material, toys and vehicles tends to have a very high degree of repetition since the explicit goal of mass production is to cut cost by establishing a production process to be used repeatedly to reduce cost. Business processes with a high degree of repetition are generally allowed to have a very large budget for designing the business process since expenses incurred during the development of the business process are shared by each repetition of the process. On the other end of the scale are, for example, development processes, such as a product design process or even the process for designing a production process itself. These types of processes can typically be used only once in an unchanged form. While development cycles are often repeated, the lessons learned from a previous cycle are usually integrated into the next cycle, requiring changes in the process. In addition, while the focus of production processes is the timely application of resources, the focus of processes with a low degree of repetition is often to facilitate collaboration between the participants. As a result, these types of processes are often also called collaborative business processes. Since these kinds of business processes can generally only used once before they need to be adapted, changed or redeveloped, the development cost of such processes must be fairly low because such development costs must be borne by the budget of a single cycle of the business process.

- *Degree of Structuring*: A business process model has a high degree of structuring if it describes all activities and execution constraints with a high degree of detail. Few decisions have to be done by the participants of the business process, instead, the business process model contains sufficient detail and constraints to provide a framework that points to an unambiguous business decision based on the information available. For example, in a business process dealing with granting credit, the business process alone includes sufficient information and rules to make a decision about the credit grant. In this case, while humans can process the rules of the business process, the result is unambiguous and requires no independent judgement by the participant. Structuring is a key element of business process management since it reduces the chances of introducing human error and arbitrary decision making. If the business process is firmly rooted in the business goals, the resulting decisions will further those goals and not impede them. Nevertheless, not every process can be structured to such a degree, in many cases an organization is required to rely on the good judgement or training of its employees to reach its goals. This is especially true for creative processes, where a rigid structure would impede the goal of the process.

In addition, Leymann and Roller provide another degree in which helps to categorize business processes (see [100]):

- *Business Value*: Business value represents the importance of a business process to an organization. A business process with a high business value is critical for the purpose of the organization and in case of business, may even be critical for its continuing existence. Business processes with a high business value are part of the core of an organization and directly contribute to the value the organization adds to its customers. For example, the business process for granting loans has a high business value for a bank because it is core to the business purpose for a bank. Business processes with a low business value do not contribute directly to the organization but enable the organization to perform business processes that do. For example, administrative business processes such as purchase approvals are of low business value to the aforementioned bank. This concept of business value is closely related but not quite identical to the distinction between primary and secondary business processes, where business processes are categorized as core competency and ancillary business processes (see also section 2).

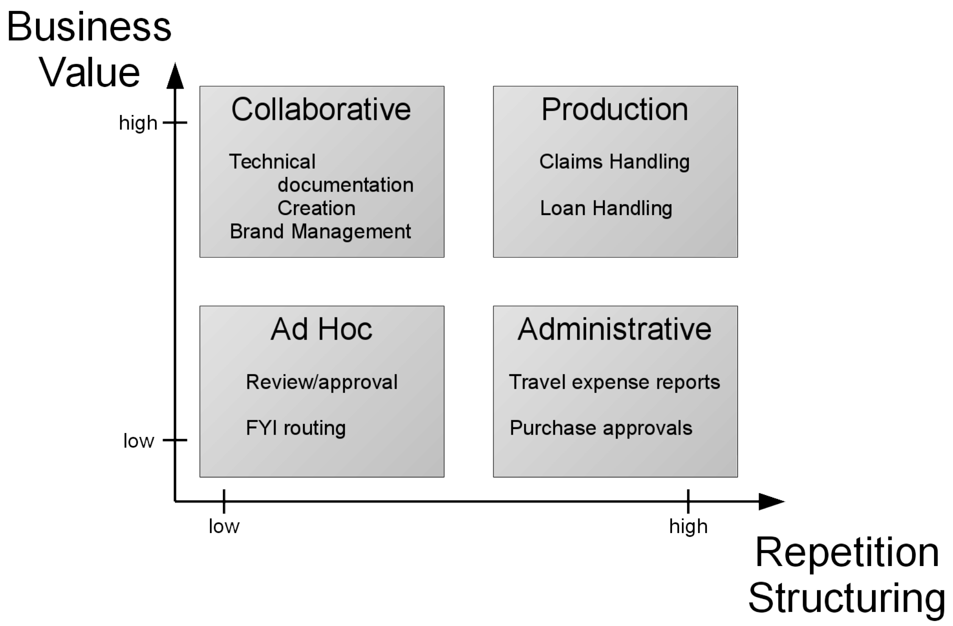


Figure 1.1: Categories of business processes based on their business value and degree of repetition (from [100])

These concepts led Leymann and Roller to categorize business processes along the dimensions of business value and repetition and structuring (see Fig. 1.1). They describe four groups of business processes (Leymann and Roller primarily focus on workflows, but the categories also apply to business processes, see section 2.6 for discussion about the distinction between the two concepts.) on different parts of two scales:

- *Ad-hoc Business Processes* are considered to have both low repetition and low business value. They are often organized only as needed (hence their name) and typically involve bureaucratic tasks such as reviews and approvals. They generally lack a fixed structure and every performance of such business processes can differ from the last. They are usually set up when required, at which time resources and time schedules are requested and provided when a business process instance is about to be performed. A typical example of such an informal process is the so-called for-your-information (FYI) routing. In this process, information is passed to participants of the process who then decide autonomously whether and to whom the information should be passed along, if at all. The process is not planned in advance and executed multiple times, the process itself provides no decisions or guidelines to the participants whether to pass the information along and the business value is low since they do not directly add to the core of the organization.

- *Administrative Business Processes* usually involve typical bureaucratic standard tasks which organizations are obliged to fulfill. One of the examples given by Leymann and Roller is purchase approvals. While the purchase itself may be related to the core of the business, for example, by being part of the supply chain process of the production side of the business, the approval of the purchase order does not directly contribute to the core of the business. As a result, administrative business process have a correspondingly low business value. This does not mean they are unnecessary, merely that they do not directly advance the core business but are ancillary to it. However, administrative business processes such as purchase approvals are highly repetitive: In general, purchases occur continuously in a business and the approval process does not change (in the case of purchase approvals, this is also functionally desirable to ensure consistent decision-making). As a result, their degree of repetition is very high.

- *Production Business Processes* have both high degrees of business value and repetition. As the name implies, they are often the business processes governing the external output of a business. In a traditional manufacturing business, this would be the manufacturing of products, but in a more service-oriented business it can involve tasks such as claims and loan handling. Production business processes are near the core competency of organizations, resulting in high business value for the organization and directly contribute to the purpose of the organization. Production of electronic products for an electronic manufacturer and the printing of books for a printing business are typical production business processes.

- *Collaborative Business Processes* often revolve around long-term planning and strategies regarding the future of an organization. For example, a mobile phone manufacturer must design and evaluate upcoming mobile phone models in order to present them to the market in a timely manner. Failing to offer a upgraded products would eventually mean that a business is eclipsed by competitors and even the most efficient production business processes could not make up for the loss of competitiveness. This means they have a very high business value and are key for an organization to maintain its core competencies. As the name suggests, these types of processes are often long-term, with execution time ranging in terms of months to years instead of hours to days in case of production business processes. Collaborative business processes also often involve human participants who are required to interact and contribute towards a common goal. While production business processes can and often are automated through the use of machines, collaborative business processes rely on the creative potential of participants involved in the process. Since rigid frameworks tend to stifle creative progress, collaborative business processes are often very loosely organized, similar to ad-hoc business processes, giving each participant a considerable amount of autonomy. In addition, the process is often only performed once in the same fashion experiences from the last iteration being used in the next one to restructure and improve the process. As a result, collaborative business processes are considered to be low in terms of repetitiveness and structuring.

Leymann and Roller in [100] point out that businesses generally focus on the production business processes and workflows since the potential gains appear to be the greatest. The reasons for this can be ascertained when looking at the details of this particular category of business processes: A high business value means that the processes directly contributes to core competencies and therefore also further the competitiveness of the business while the high degree of repetition means that any gain as a result of engineering investment in design and optimization of such business processes are multiplied by the high number of repetitions. As a result, existing approaches to business process management — including modeling, execution, monitoring and analysis — focus on these types of processes.

In addition, the category of administrative business processes can be addressed by the same sort of models: While their business value is lower than production business processes, the repetitive structure means that similar approaches to modeling and execution can be effective and thus the tools made for production business processes can be repurposed to support administrative business processes as well.

However, as a result of this, the categories of ad-hoc and collaborative business processes received comparably little attention (see Fig. 1.2) from traditional business process management tools and methodologies. In case of ad-hoc processes, the potential gain of business process management is indeed comparably low as they are often very simple.

For example, the ad-hoc business process of FYI-routing consists merely of sending a message containing information that needs to be conveyed to interested people, while leaving it up to the receiver to pass the message on to any additional parties they may deem interested in the information. This sort of process is both informal and of low business value and would therefore gain very little from the introduction

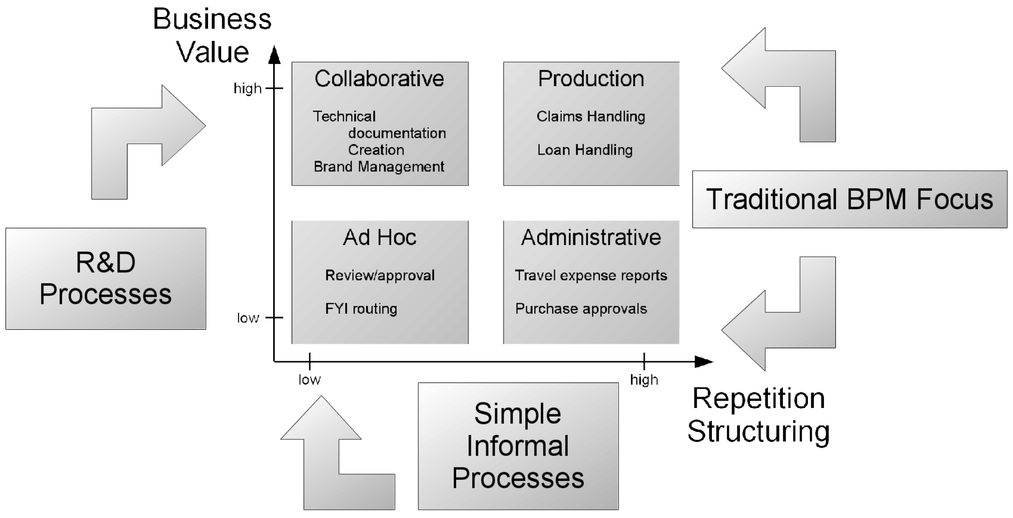


Figure 1.2: Traditional Business Process Management tends to focus on business processes with a high degree of repetition and structuring (center image from [100])

of business process management.

On the other hand, collaborative business processes are a more interesting case: Collaborative business processes cover issues such as research and development which are key for the future success of organizations. They therefore represent an area of potentially large gain from the introduction of business process management but have traditionally been neglected due to their unusual structure and properties which make models used for production business processes a poor match for their requirements. The next section will focus on this category of business processes and attempt to distill some properties of common examples in this category in order to evaluate the requirements necessary when attempting to introduce business process management in this category of business processes. The resulting properties will then spell out the particular challenges and research questions that need to be addressed for a different approach to business process management supporting these types of business processes.

## 1.2 Research Questions and Goals regarding Long-running Autonomous Processes

Collaborative business processes are often different from the business processes that are targeted by traditional business process management such as production business processes. Production business processes and administrative processes are often highly structured, with one or more clear paths of actions taken by its participants to accomplish the process. A production process such as the printing of a book is over quickly, ranging from seconds in a fully automated business process for the manufacture of a simple product to a few hours for more complex products which require human involvement as in the case of the production of a vehicle.

In contrast to this, collaborative business processes often have a different set of properties. This category of business processes includes processes which involve the research and development of new products, long-term planning and strategic processes like brand management. As a result, many collaborative business processes feature a number of properties that make them hard to tackle with traditional business process management (cf. Fig. 1.2):

- The execution lifecycle of collaborative business processes are often very long. The development of new products can take months even for the most simple products and can stretch into years for more complex ones.
- They often involve a diverse set of participants coming from multiple departments and potentially external suppliers who need to interact and contribute to the process to be successful. The development process for a complex product can involve experts not only from production engineering but also accounting, supply chain management, human resources and marketing in order to ensure not only a good product that the customer desires but also a marketable and producible product.
- The participants of such processes are often highly independent. They are experts in their respective fields and a large part of the process relies on the participants using their expert knowledge to reach their decisions. The business process itself can only facilitate the exchange of information between participants but cannot supply any decision.
- In addition, despite attempts to standardize equipment and *Information Technology* (IT) systems in large organizations, the experts involved in collaborative business processes often use unique and specialized software tools and their IT systems may have a certain degree of independence not only to install additional software but also to acquire and deploy hardware systems as needed. This is often done without the involvement of the IT department that normally organizes the IT system in the operational side of businesses in order to reduce administrative overhead and enable the participants to quickly set up experimental systems.
- Due to their longevity in terms of execution, collaborative business processes instances can outlast the organizational structure a business had when the process instance was enacted.
- Furthermore, the business circumstances may change during the execution of a collaborative business process, potentially invalidating assumptions about what actions must be performed to successfully finish a process instance.
- Collaborative business processes are deeply connected to the strategic planning of companies. They are key to the future success of the business and are the means by which businesses hope to achieve their long-term business goals such as gaining market share or entering a new market niche. It is therefore very important for businesses that such processes have a high degree of alignment

with the business goals set by the strategic planning level (see also [100] and [64]).

These properties point towards two distinct aspects of these type of properties which make them hard to address using traditional business process management methodologies and tools:

- The processes are *long-running*, meaning they have execution life-cycles in terms of months and years instead of hours.
- The participants and the processes themselves have a high degree of independence or *autonomy* compared to the more controlled environments of production and administrative business processes.

*Process participants* are actors that participate in the process, such as human actors but also machine actors, that perform tasks contributing to the process goals. While the notion of a long-running business process is clear, being defined as a process with an execution time of multiple months or years, the notion of autonomy is more obscure. For example, the Merriam-Webster Online Dictionary defines autonomy, among other definitions, as "a self-governing state" and as "self-directing freedom" (see [40]). This definition implies a certain independence from external influences for the autonomous actor.

The agent technology community offers more technical definitions of autonomy as a feature for software agents which also indicates this sort of independence by emphasizing the self-control aspects. For example, Castelfranchi as well as Jennings and Wooldridge advance the following definition of autonomy (see [32] and [164]):

> autonomy: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state [32]

In terms of the processes, it makes sense to consider an executing process to react to situations without human intervention. It is also natural that they can perform actions like sending e-mails or performing database operations. It is also easy to imagine that a business process has an internal state, in fact, this tends to be the norm since the process, at the very least, needs to keep track of its own progress.

However, with regard human participants of a business process, some aspects of this definition are not very helpful: An autonomous human participant is already a human and as a result the "direct intervention of humans" can only be interpreted as other humans directly interfering (e.g. through direct orders) in the behavior of the participant.

Aside from this notion, the rest of the definition clarifies how autonomous business process participants should be considered: They are expected to have at least some control over their actions and use their knowledge about the current process instance as well as their expert knowledge ("internal state") to contribute to the progress of the business process.

As a result, this work seeks to provide business process management tools for such long-running, autonomous business processes. Based on the earlier discussion, these processes are defined as follows:

> *Long-running autonomous business processes* are business processes exhibiting *autonomous behavior*, involve an average execution time greater than a month but typically lasting multiple years and require the cooperation of *autonomous process participants.*

As a result of this definition, long-running autonomous business processes exhibit three main aspects that set them apart from other business processes, especially the traditional production business processes: The long execution times, autonomous behavior of the processes and autonomous process participants. Autonomous process behavior can be defined analogous to the definition of autonomous agents by Jennings and Wooldridge as follows:

> *Autonomous process behavior* describes the behavior of business processes operating without the intervention of humans or others, and have some kind of control over their actions and an internal state helping them adapt in a changing business environment.

This definition means that while there is the notion of a structured process, implying a business process model, the actual behavior of an enacted business process instance exhibits a dynamic behavior which can adapt to a changed business situation without remedial external intervention.

The third component of long-running autonomous processes, the autonomous process participants, can be similarly derived from the same notion of autonomy as follows:

> *Autonomous process participants* are human or non-human actors performing actions within the context of a business process without direct intervention by other actors and have some kind of control over their actions outside the business process and apply their expert knowledge in those actions.

Unlike production business processes, where the actions are as closely defined by the business process as possible, the participants of long-running, autonomous processes retain a deliberate amount of independent control over their actions to allow them to apply their expert knowledge as part of the process and give them the flexibility for creative freedom commonly required as part of such processes.

These two properties of long-running execution time and autonomous behavior lead to the central research questions for this work:

- What concepts, languages, systems and tools can be developed to improve the support for long-running autonomous business processes?

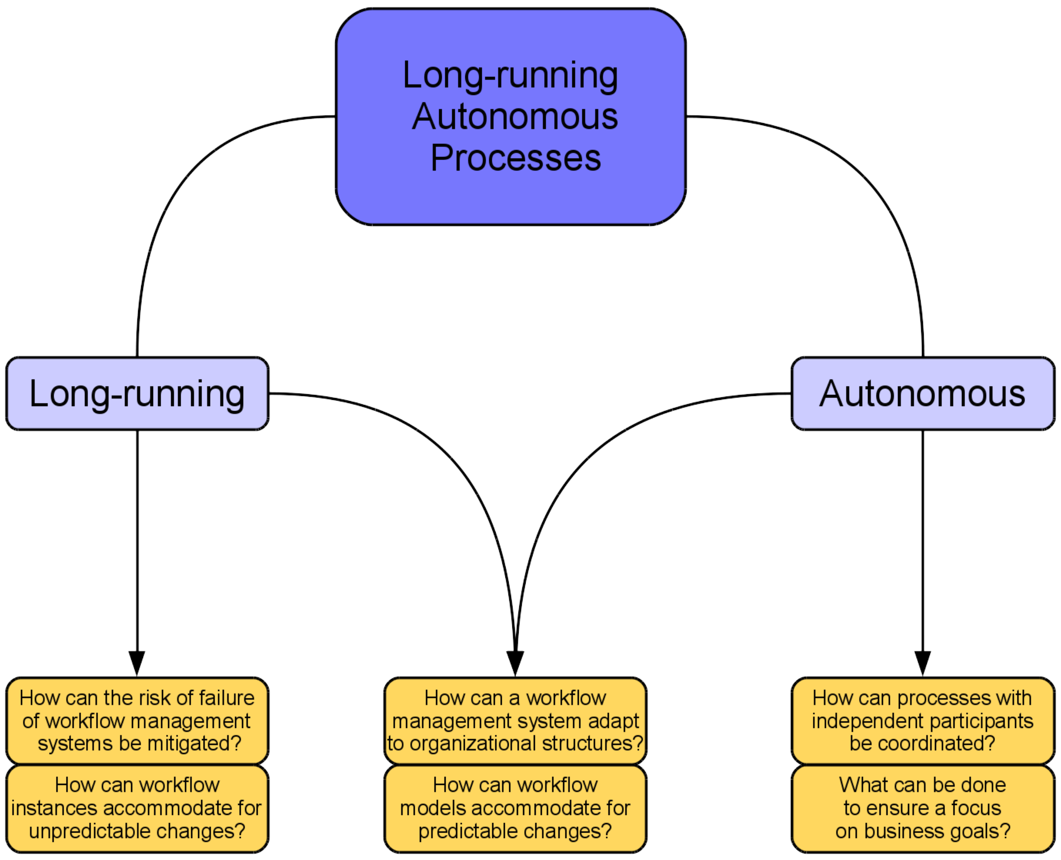

Long-running Autonomous Processes
Long-running
Autonomous
How can the risk of failure of workflow management systems be mitigated?
How can workflow instances accommodate for unpredictable changes?
How can a workflow management system adapt to organizational structures?
How can workflow models accommodate for predictable changes?
How can processes with independent participants be coordinated?
What can be done to ensure a focus on business goals?

Figure 1.3: Long-running autonomous processes are a particular challenge to business process management due to their unique requirements which differ from production and administrative business processes

In order to address this question, one can look at each of the two properties, autonomy and long execution times, separately and derive the particular challenges resulting from them. In addition, one can investigate what challenges can be derived from their combination. Figure 1.3 provides questions for each of the two properties as well as questions that affect the combination of the two.

Due to the long-running property of the business processes, the supporting workflows are expected to be executed on a workflow management system for a lengthy period of time. This both increases the chance of the system failing during the time frame the workflow is executing as well as increasing the impact of a failure by potentially increasing the amount of time and labor lost. While autonomous processes are more free-wheeling and can potentially be continued manually after a system failure, the impact can still be severe due to the accumulated state of the process such as documents and other business information that can be lost if the system fails. Since the chance of failure multiplied by the impact of failure is defined as the risk, this means the risk of failure for a workflow management system used for long-running autonomous business processes is increased, leading to the first research question that needs to be addressed:

- How can the risk of failure of workflow management systems be mitigated?

In addition, over the given period of time, changes in the business situation may occur which can be predicted in advance requiring the workflow instance to adapt to the new situation. Furthermore, these changes can happen at any point in the processes, for example due to a participant identifying an issue or requiring a change to earlier work. This requires the workflow designer to include a large number of potential contingencies, leading to the following question:

- How can workflow models be prepared for predictable changes?

Not all of the changes that can occur over a long period of time can be predicted. A development process may run into unforeseeable problems during the development that require a different approach. While this question raises issues of flexibility, it also includes aspects of agility since the preparation of the workflow models for possible future changes is an agile aspect of workflow modeling.

Alternatively the business environment may change in an unpredictable fashion, for example through announcement by competitors. Therefore, the following question needs to be addressed:

- How can workflow instances allow for unpredictable changes?

Business processes and workflows are used to coordinate resources such as labor within organizations based on a business strategy. Since the participants of long-running autonomous business processes require a great degree of independence and are responsible for many decisions, the question of such coordination becomes more difficult:

- How can processes with independent participants be coordinated?

Since the participants are independent and processes must adapt to changes occuring after process enactment, the question becomes how these adaptions are still in line with the business strategy. Therefore, a connection should be made to the strategic level of the business process management:

- What can be done to ensure a focus on business goals?

Finally, the participants of long-running autonomous business processes are also autonomous regarding their organizational structure, which may not entirely conform to an ideal process-oriented organization (see section 2.1). As a result, an answer should be found for the following question:

- How can a workflow management system adapt to organizational structures?

In order to address these questions, a number of research goals affecting various parts of a business process or workflow management system ought to be considered. These can include both topics during design time, such as concepts or language features to allow the inclusion of predictable changes as well as challenges that become important during runtime like the question of system stability.

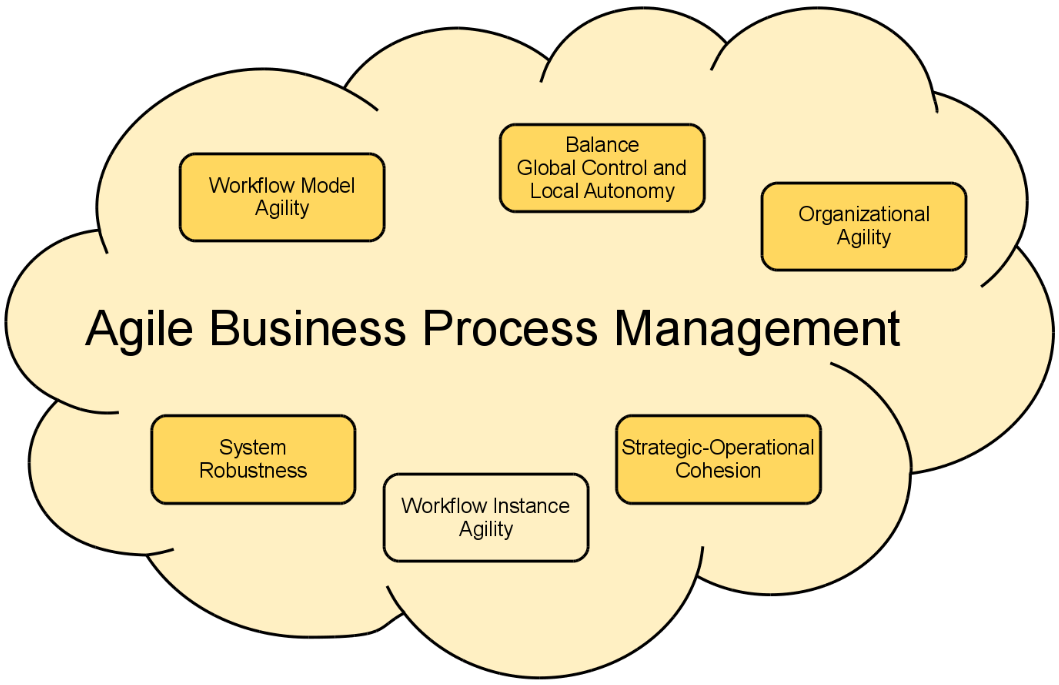

Workflow Model Agility
Balance Global Control and Local Autonomy
Organizational Agility
Agile Business Process Management
System Robustness
Workflow Instance Agility
Strategic-Operational Cohesion

Figure 1.4: Long-running autonomous processes are a particular challenge to business process management due to their unique requirements that differ from production and administrative business processes

Software development being an example of a collaborative business process, the software development community has long recognized the need for increased flexibility and reaction to changing requirements during such processes. Such processes also fall into the category of long-running autonomous business processes where the development time of a software package can be very long (or even indefinite, if maintenance is included) and the participants need a considerable amount of freedom in order to work effectively. The Agile Manifesto (see [7]) outlines a number of best-practice principles that have proven themselves to be beneficial when performing software development as part of an organized process. While the focus of the manifesto is centered primarily around software development, some of the principles point towards general issues with creative, collaborative, long-running and autonomous business processes. For example, the manifesto provides the following four principles (from [7]):

1. Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

2. Business people and developers must work together daily throughout the project.

3. Build projects around motivated individuals. Give them the environment and support they need, and trust them to get the job done.

4. The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

The first of the principles listed here indicates the need to allow a certain autonomy towards the execution of processes, so they will be able to react to changes. The second principle notes the necessity for deep integration with the business side. Finally, the third and fourth principles indicate the need for autonomy regarding the participants in the process in order to allow them the necessary leeway to apply their expert knowledge and exchange information with each other.

Alpar et al. also emphasize the ability to react to changes and provide a more concrete definitions of the terms agility and flexibility (see [125]):

- *Flexibility* refers to the ability of information systems to easily adapt to *current* changes.

- *Agility* refers to preparedness of information systems for *future* changes.

Since one cannot be prepared for future changes without being able to react to current changes, as shown by Alpar et al., a high degree of flexibility is necessary in order to attempt the introduction of agility.

Long-running autonomous business processes from a general business process management perspective introduce a number of factors that necessitate both flexibility and agility such as the high degree of freedom afforded to the process participants increase the chance of introducing changes. The long execution time further increases this chance by extending the time period during which changes can occur.

As a result, supporting long-running autonomous business processes requires workflow models, workflow engines and workflow management systems to offer options to support flexible and agile approaches as opposed to the more rigid models and frameworks emphasized by traditional business process management with a focus on often-repeated, highly-structured production and administrative business process. As a result, the tools and methods that help to support long-running autonomous processes are called agile business process management in this work, using the following definition which is based on the general definition of business process management used by Weske (see chapter 2 and [161]) as well as the ideas of flexibility and agility provided by Alpar et al. (see [125]):

> *Agile business process management* includes concepts, methods, and techniques which increase both flexibility and agility of the processes and their environment in order to support the design, administration, configuration, enactment, and analysis of long-running autonomous business processes.

Based on the research questions, a number of research goals can be derived (see Figure 1.4). The first research question addresses the issue of long-term stability of the system. A workflow management system accomodating these questions needs to remain stable despite running over multiple months. As a result, the first research goal is to increase the robustness of the business process or workflow management system:

- *System Robustness*: The long execution time of the processes means that there are higher risks of failure, not only with regard to the chance of a system failing but also the impact such failure may have on the overall system, so that a fault or deliberate shutdown of parts of the system does not immediately result in complete failure of the whole system. The system should therefore be resilient to change and degrade gracefully.

Another research question asks how business process models can be developed that can include options to react to changes in the business environment or based on the autonomous workflow participants. While such alternative courses of action can be included in some form in existing workflow modeling languages, this is not always sufficient for long-running autonomous processes (see chapter 3 for a discussion on this topic). As a result, modeling languages need to be developed or improved to tackle this question:

- *Workflow Model Agility*: Since the business process will be executing for a long time, the business situation may change while the process is running or revisions of a later stage may even invalidate the results of an earlier state due to issues identified only at a later point in the process which were not apparent at an earlier time. Another issue arises from the fact that research and development processes are conducted in a particular way due to their creative and autonomic nature (see 1.2): Often, a long development processes is divided into stages, phases or milestones, which will be reached by meeting certain criteria after a certain time. However, constraints which become apparent at a later stage of the process may necessitate changes that require verifying and adjusting development done to reach the previous milestone. This means that steps which were already considered finished have to be done again at a later stage in the process. As a result, long-running autonomous processes must be able to adapt to such new situations without manual outside interference by allowing the inclusion of contingencies in the business process model without excessively increasing the complexity of that model or including modeling elements simply for technical reasons in the model. It allows the resulting workflows to flexibly react to changes and allows workflow designers to increase agility by preparing the workflows for potential future changes.

The participants in long-running autonomous business processes are independent or autonomous actors. This means that in order to perform adequately, they require an increased freedom to chose when to perform certain actions based on their own expertise while a traditional approach to a workflow emphasizes control over the processes by deciding which actions must be performed at a given point in time. One therefore must seek a balance between the control aspects of workflows and the independence of the workflow participants:

- *Balance Global Control and Local Autonomy*: One goal of business process models and workflow models is to impose a structure or guideline on how a particular business process is to be performed. On the other hand, the participants in a long-running autonomous business process require a certain autonomy within their area of contribution to decide on which actions are necessary. A workflow model language therefore has to address how to balance between giving the participants free reign over their labor in their local area of expertise while still maintaining as much global structure as possible given the nature of the processes.

Due to their autonomic nature, both the participants and the workflows themselves are relatively flexible in selecting work and performing it. The requirement to balance global control and local autonomy means that the control over the process by the strategic planning of the organization is weakened. As a result, one has to address how to maintain a focus on the business goals of the organization despite this loss of control:

- *Strategic-operational Cohesion*: Autonomous process participants and autonomous processes have a greater degree of flexibility and reduced control. A business process and workflow modeling language has to somehow maintain a connection to the strategic goals of an organization despite this reduced direct control.

While it is always desirable to structure an organization in alignment with its business processes (see chapter 2), it is not always possible to impose such structure on an organizational layout in all cases. In particular, research and development processes often involve the inclusion of a diverse set of people across the organization, often in contrast to the usual structure used (for example, the production manager may be required to give his input as to how a new prototype could actually efficiently be produced). As a result, different forms of organizational structures have to be supported:

- *Organizational Agility*: A business process management system should offer support for different forms of organizational structures. This includes traditional functional organizations as well as process-oriented ones with a focus on a different set of processes. This can include bridging technical issues such as specialized IT systems and approaches to information management.

The final research goal centers around the question regarding business changes or changing requirements based on the autonomous behaviors of the participants and processes which cannot be predicted during the design of the model, meaning that the changes must be performed after the workflow instance has been enacted:

- *Workflow Instance Agility*: Business changes can require that a workflow instance which has already been enacted be modified since the change could not or was not predicted during workflow or business process modeling. This means that the running workflow instance needs to be adapted, preferably in a fashion capable of maintaining a structured approach. In addition, the change should ideally be implemented by process experts.

The last research goal is only partially addressed by this work for the following two reasons:

- An approach for workflow instance agility is already available in the form of ADAPT2 (see 3.10), which demonstrates the concept in principle.
- The current runtime approach for executing the long-running autonomous business processes with the approach proposed here would require substantial expansion to support this goal. Nevertheless, a discussion on a possible approach can be found in section 10.2.

These research goals represent an attempt to tackle processes that are long-running, exhibit autonomous process behavior and allow for autonomous business participants.

The aim is to improve the support for such business processes by addressing some of their more pressing requirements identified by investigating primarily research and development business processes and thus may not yet represent the full spectrum of business processes of this type.

As a result, agile business process management may include other issues which have not yet been identified based on similar business processes with slightly different requirements. In addition, the modeling languages, techniques, systems and tools developed for this approach may be useful for other types of business processes.

This work therefore represents an attempt to improve support beyond traditional means for such business processes, nevertheless further work may be yield additional improvements for such processes as well as the opportunity to expand this work to a broader spectrum of business processes. The contributions of this work are centered around the business process management lifecycle introduced in the next chapter (see Fig. 2.6) with the aim of supporting long-running autonomous business process in every stage of the cycle. The major contributions of this work are as follows:

- The Goal-oriented Process Modeling Notation (GPMN) modeling language as presented in chapters 4 and 5, providing support for the design and implementation phase of the lifecycle.
- Execution of GPMN workflows using a model conversion-based workflow engine shown in chapter 6, which is part of the solution for the execution phase of the lifecycle.
- The distributed workflow management system approach and implementation in chapters 7 and 8, which provides the remaining requirements for the execution phase as well as support for the monitoring phase of the lifecycle.
- An approach for analysing and validating GPMN workflows using a simulation-based approach presented in chapter 9. This provides an approach for tackling the analysis phase of the lifecycle.

The structure of this work is as follows: The next chapter will provide an overview of business process management in general; however, since this involves a relatively broad area of research, will focus on particular aspects that are relevant to the business processes tackled by this work (see chapter 2). Following this, a spectrum of typical business process modeling languages is introduced and their limitations regarding long-running autonomous business processes is evaluated (see chapter 3). Afterwards, chapters and introduce goal-oriented business process modeling and the Goal-oriented Business Modeling Language (GPMN). The GPMN workflow engine for executing GPMN workflows is introduced in chapters 4, 5 and 6. The concept of distributed workflow management is introduces in chapter and an implementation is presented in chapters 7 and 8. Chapter 9 provides an approach for simulating and validating goal-oriented workflows in combination with a simulated workflow management environment. The final chapter provides a major deployment scenario and evaluation of the overall system components and the system is evaluated based on the originally introduced research goals for agile business process management.

# Chapter 2

# Business Process Management

Business process management owes its legacy to a variety of disciplines, most prominently in business administration, due to being directly applicable in that field. It also touches a large number of stakeholders such as business managers, IT professionals, software developers, business consultants and business employees in general. Each group has its own unique perspective on processes and different priorities regarding various aspects of business process management.

For example, a business manager may be interested in increased efficiency or better customer service, a software developer may be concerned with (semi-)automated execution of business process and integration with legacy software. An IT professional is likely to be interested in administrative overhead in terms of necessary IT infrastructure like servers and with security aspects of such an effort that spans the entire organization and may even include the organization of suppliers. A business consultant interest often includes opportunities to optimize already existing process and general employees of an organization want to know how it will affect their work and are concerned about job security.

Furthermore, like any essential business topic, it is affected by short-lived terminology. As a result, an very large vocabulary has been developed to describe various aspects of business process management, which is not always used in a well-defined or consistent manner (see [156]). In order to clarify this situation, this work will use the following definitions for general business process terms:

A business process organizes business activities within an organization in order to reach an organization goal. In [161], Weske defines a business process as follows:

> A *business process* consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but may interact with business processes performed by other organizations.

Note that this definition of business processes does not include any sort of automation or even organized effort to describe them. In fact, even today business processes exist that were created implicitly by the individual effort of its participants. The discipline that concerns itself with explicit administration of business processes within an organization is called business process management, which Weske defines as follows:

> *Business process management* includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes.

The central aspect of business process management is the orientation around customers. Despite the obvious importance of customers, this is not always a given in most organizations. Schmelzer and Sesselmann explain this as follows (translated from [64]):

> It depends on the satisfaction of the customer whether they buy the offered products and therefore ensure the existence and future of a company. Despite their outstanding importance, in practice customer relations and customer satisfaction are often neglected.

The authors additional points out some additional facts that emphasize not only the general difficulty of companies to address customer satisfaction but also point out a particular deficit of German businesses in this area (translated from [64], based on [70], [29] and [66]), according to this:

- German companies possess an inferior knowledge of their customers and customer profiles than companies in other European countries,
- there exists a need to catch up investigating the conceptions and desires of customers (customer perspective)
- 72% of the companies recognize the great importance of customer-orientation but only 19% of the companies act according to this principle
- 20% of the executives and 66% of the employees do not know their own customers
- managers are more intensely concerned with their competition than their customers
- 40% of the employees are unaware of the importance of satisfied customers
- the acquisition of a new customer is five to eight times as expensive as the enduring loyalty of existing customers
- ISO-certification failed to improve customer focus

One approach to addressing these concerns is the introduction of business processes. Schmelzer and Sesselmann emphasize this point as follows (translated from [64]):

> A reliable approach to remedy the presented deficits is the introduction of business processes. Business processes put customers and customer relations at the center. Using business processes the thoughts and actions

of the whole company will be focused on customers. The more efficiently business processes fulfill customer demands and expectations, the more satisfied are the customers and the more successful is the company.

This customer-centric approach is often reflected by the fact that business processes often start and end with the customers, usually by receiving a customer request to start the business process and ending with the customer receiving the desired goods and services. Note, however, that the customers of business processes are not always the same as the customers of a business. Again, Schmelzer and Sesselmann distinguish two types of customers (see [64]):

- *External customers* are customers of the business or organization that approach the organization for the offered goods and services. As such, they largely represent the customer base of the organization and are the potential receivers and users of the output of the organization. In many cases, these customers are end-users who intent to use or apply the products and services themselves.
- *Internal customers* on the other hand participate themselves in the business processes of the organization. In fact, any participant in a business process who performs a part or even a single task of a business process is both the customer of the business process step preceding their part and the supplier of the business process step following their part. They often belong to the organization itself in the form of competency centers or other employees tasked with parts of a business processes.

Internal customers are as critical as external ones. If a supplier provide a flawed output to an internal customer it either results in the customer being unable to adequately perform their own part in the business process and thus ultimately providing an external customer with a poor experience down the line. Alternatively, the internal customer has to supply additional effort in terms of labor and capital in order to make up for the flaws in the supplied product or service.

As a result, the goal of business process management is to put the business processes and therefore the customers at the center of operations and efforts of the organization by providing a number of techniques, methods and approaches for tackling the business processes within organizations. However, due to their orthogonal nature with regard to traditional business organization, one of the core hurdles for the introduction of business processes and business process management is the organizational layout and structure. The next section will provide an overview of the changes necessary for organizations introducing business process management and the challenges that result from this introduction.

## 2.1 Organizational Aspects of Business Process Management

While the previous definition of business process management encompasses the methodologies and best practices, in terms of organizational strcutures business process management is an approach for managing a organization or business that centers its activities around the activities or work performed by the organization. This stands in contrast to the traditional approach of structuring an organization along specialized departments providing functions (see [123]). Schmelzer and Sesselmann provide the following properties to differentiate the two approaches in [64]:

| Functional Organization | Process Organization |
|---|---|
| vertical alignment | horizontal alignment |
| high division of labor | integration of labor |
| focus on execution | focus on object processing |
| deep hierarchies | flat structures |
| status-centric thinking | business success thinking |
| power-oriented | customer- and team-oriented |
| department goals | process goals |
| goal: cost efficiency | goal: customer satisfaction, productivity |
| centralized external controlling | decentralized self-controlling |
| controlled information | free and open information |
| efficiency projects | continuous improvement |
| substitute processes, redundancy | focus on adding value |
| complexity | transparency |

Figure 2.1: Managing organization using business processes differs from the more traditional functional approach (translated and adapted from [64])

The listed properties highlight the differences between the two organizational approaches, the more traditional functional structure which has been the focus of large organizations focused on mass production and the more modern, service-oriented form of organization based on business processes. The following will attempt to illustrate the properties of both organizational forms.

### 2.1.1 Functional Organizations

Both functional and process-oriented organizations attempt to follow goals with the aim of producing planned results, but the source and target of both differ (see 2.1). Functionally-oriented organizations are structured along the lines of highly specialized departments. A functionally-oriented organization is vertically aligned, which means that each department is focused solely on best fulfilling its own function, i.e. the goals of the department is performing its role instead of contributing to the business process.

This organization results in a strong division of labor: Each department specializes on its function and can optimize it to produce good ourcomes with regard to their results, which leads to an alignment vertical to the business process (see 2.2). Each department is focused on executing its function and due to deep hierarchies, the department management is given substantial autonomy to organize the structure of the department which other departments can only influence by appealing to higher levels of management.

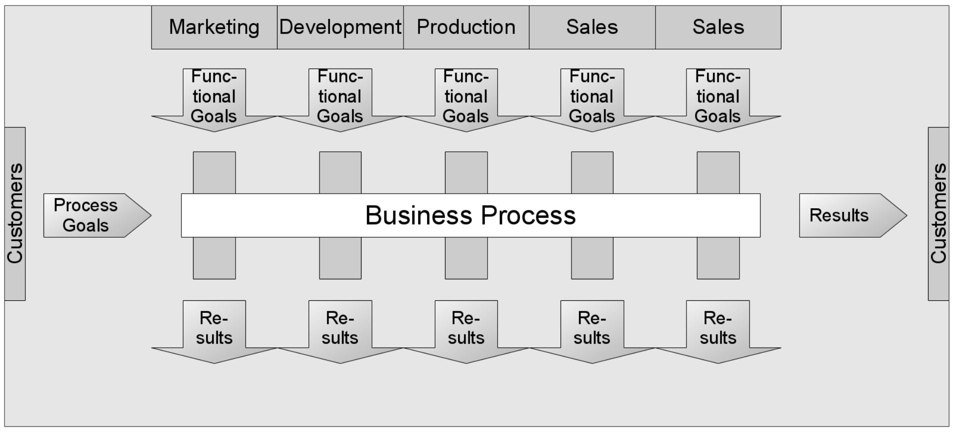


Figure 2.2: Functional organization are aligned vertically, with each department focusing on best performing its function (translated and adapted from [64])

This results in status-centric thinking, where department heads compete for prestige, such as the department's size, its importance to the business and the attention from higher management which it may or may not receive. Along with deep hierarchies, this also orients the department and its members to seek power and influence within the organization, either for themselves or their departments.

Each department adheres its own departmental goals and objectives. For example, the accounting department has a strict focus on performing excellent and efficient accounting, while the production department focuses on low-cost production. As a result of this, this isolation of departments along with the status- and power-centric thinking within them result in the potential of department goals diverging from contributing to the overarching process goals: For example, the accounting department may attempt to impose onerous documentation requirements on the production department since it reduces costs, increasing the cost during production but reducing the cost for the accounting department.

If the difference between the cost reduction and the cost increase is negative, the overall cost for the business process increases to the detriment to the organization. Nevertheless, this goal may still be pursued by the accounting department since it benefits their interests (to the detriment of the production department).

The information the departments possess is also tightly controlled internally and not generally conveyed to other departments. In part, this is due to the aforementioned goal and power dynamics. However, another factor is the impact departmental isolation, deep hierarchies and vertical orientation: Interdepartmental information exchange can generally only be accomplished by conveying the information up and down the hierarchy. Specifically, this means that information has to travel up the hierarchy until it reaches a point where the receiver is responsible for both departments before it can be passed down towards the targeted department. In the worst case scenario, this means the top management of the organization. It it then processes and passed down the hierarchy again until it reaches its final destination.

This not only results in fairly long delays for information exchange, it also involves more people and resources than necessary. Even relatively minor details are passed to a very high level of management, which then has to deal with this information despite the fact that it has already been escalated beyond its requirement. This means that the higher layers of management become impeded by a large amount of very detailed information and become a bottleneck of information exchange.

Since functional organizations hardly acknowledge the existence of business processes, departments tend generate and adopt implicit and ad-hoc substitute processes which emulate the process behavior. This is usually done through institutional knowledge. A typical example is setting up interfaces between departments where business objects are passed from one department to the other since they are not regarded as a particular focus as they are in process-oriented organizations. However, these interfaces have a number of disadvantages (first three from [123] and [64]):

- Interfaces are *storage locations*, where business objects are stored for a certain amount of time. This is the result of temporal synchronization problems between departments, where one department has finished its tasks on the object but the next department is not ready to continue working on it. This introduces inefficiencies and delays.
- Interfaces are a *source for errors* since they invariably lose information regarding interrelation between tasks.
- Since the responsibility for a department starts and ends with interfaces, they are a source for *organizational irresponsibility.* In case of mistakes and issues, it is difficult to trace the responsibility for the issue to one department or the other, which is compounded by the previous point of interfaces being an error source.
- Finally, interfaces are a barrier for the transmission of knowledge, since implicit experiences, knowledge and proficiencies must be made available to allow unambiguous communication without loss of context. In design, this challenge is known as the ”throwing it over the wall” problem, where one department finishes a design step and simply passes the result to the next department while accidentally omitting information which is vital for the following steps, resulting in otherwise avoidable errors and delays in the next step (see e.g. [170])

The resulting landscape consists of a number of process islands for each department, increasing the necessary coordination effort required to be performed by management. Since, as as mentioned before, the management is already weighed down by acting as an interdepartmental communication channel, the load on this particular bottleneck increases further.

Nevertheless, functional organization has its merits, which is why it was adapted in the first place: Once coordination has been accomplished and responsibilities are clearly assigned, departments can drastically increase productivity through internal specialization and optimization.

In a static business environment, this means the organization can eventually reach an optimum of efficiency and productivity. In the past, this situation was pretty close to the norm (cf. [143]): Markets were simple and stable, technological progress was steady but relatively slow, product life cycles were long and the focus was on mass production. In such an environment, the goal of functional organization of cost efficiency fit the situation.

However, nowadays the business environment is more dynamic and volatile: Markets change rapidly, as do customer demands. Technology often progresses at a more rapid rate, quickly making products obsolete and shortening product life cycles.

This highlights the main issue of functional orientation of organizations when used in a dynamic business environment: Once an organization needs to change and adapt its processes, the coordination effort, the challenges of communication and the difficulty of defining interfaces between departments arise anew. This is not a major issue when it happens rarely, but such adaptations are now required of organizations at an increasing pace.

### 2.1.2 Process-oriented Organizations

Process-oriented organization attempts to mitigate the issues of functional organizations and improve businesses by allowing organizations to increase their flexibility and react more rapidly to a changing environment. This approach calls for realigning the focus of an organization with its processes instead of the functional departments. This results in a different set of priorities (see 2.1):

First, the alignment of the organization is no longer vertical along the lines of the departments, but horizontal along the lines of the processes (cf. 2.2). This means there is less of an emphasis on division of labor but rather the integration of perfomed labor between departments. Instead of the mere execution of functionality, process-oriented organizations focus on the actions required to process business objects as they pass through the organization.

By aligning an organization with its business processes, previously-needed deep hierarchies used to coordinate fragmented departments is no longer necessary. The business process and participants are instead coordinated by a dedicated business process manager whose sole responsibility is the smooth and efficient execution of the business processes they have been assigned.

Specialized labor is still required in such a system but, compared to a functional organization, the importance of the departments is reduced: Instead of forming isolated and self-contained silos, they become service or competency centers providing services to the business processes. This means that the job of the management of those centers is reduced to quickly and efficiently introduce the services the business process managers have requested due to necessity as part of the business processes.

The coordination effort is provided by a business process manager with an overarching view on the business process and how it interacts with the organization, instead of multiple coordinators in each department with insufficient information attempting to provide the necessary labor. The resulting management structure can be relatively flat, often with only two levels; consisting of the business process management implementing the strategies of top management, and the competency center management providing services to the business processes.

Since the focus of the organization is the business processes, the participants become stakeholders in its success. As a result, instead of focusing on the personal and departmental status, participants' thoughts and efforts are now centered around the business processes. Considering that the business processes are meant to represent organizational strategy, the thinking of the participants is now more closely aligned with the overall success rather than status.

In addition, the flat structures and competency centers provide fewer incentives for power-orientation. The competency centers are no longer power bases like the functional departments and the flat structures provide fewer opportunities for power struggles. Instead, since the business process becomes the center of attention, the focus shifts to the business process team and the customers it aims to serve.

Instead of employing a monolithic and centralized management control, process-oriented organization use decentralized and process-based management control. The management control is exercised as part of the business processes and emphasizes self-coordination (cf. [64]). One of the benefits of this approach is that it offers opportunities for performance increases and learning process that not only benefit a single participant but also generate a knowledge base for the benefit of the whole organization.

These benefits are furthered by the free and open exchange of information. Instead of departments hoarding potentially useful knowledge and experiences, they are exchanged due to the emphasis on common business processes. Participants are encouraged to propel this exchange by encouraging them to submit suggestions for improving the business processes, leading to a cycle of continuous improvement instead of separated and uncoordinated projects.

Synergistically, this leads to an overall culture of transparency, where power struggles are reduced as the focus centers around the goals of the business process whose sole aim is to provided added value to its customers. The free exchange of information and the horizontal view of the business process managers reduce redundancies compared to functional departments and their rigid and error-prone inferfaces between them.

However, the main benefit is derived from the fact that it allows an organization to adapt with high flexibility and reorganize available resources according to the needs and demands of its customers. Schmelzer and Sesselmann emphasize this as follows (translated from [64]):

> Process-oriented organizations facilitate a flexible reaction based on the needs and demands of the market and the customers. The common view on the customer dictate the goals and actions of the employees. This common orientation enforces a strong coordination effect and supports horizontal cooperation and collaboration.

However, introducing business process orientation is a demanding task for an organization. As a result a myriad of methodologies, techniques and organizational structuring is required to align organizations with their business processes. This is even harder for pre-existing organizations which are already structured in a functional manner; nevertheless, the benefits derived from it and its necessity in a modern business environment are especially important for organizations that need to sustain themselves in competitive markets (see [135] and [150]).

## 2.2 Types of Business Processes

Business processes are used by a large variety of organizations, most importantly but not exclusively by commercial businesses. These organizations have different purposes, are structured in a different ways and, as a result, have different approaches to fulfilling their purpose. As a result, business processes cover a wide area of applications and can differ tremendously between organizations, despite the existence of certain standard processes (see [149]) which are applicable for typical standard business situations or needs.

As a results, multiple attempts have been made to propose general categories for business processes. A very common approach is dividing business processes into primary business processes and secondary processes (for example, see [157], [41] and[146]). In addition, the concept of tertiary business processes, introduced by van der Aalst and van Hee in [1] is sometimes used, as follows:

- *Primary Business Processes* are processes which are directly involved with the main purpose of the organization. Organizations generally exist in order to fulfill a specific purpose, for example, the production of vehicles, the sale of goods or the collection of taxes. Primary business are directly involved with that purpose. For example, for a car manufacturer, the production process for new cars would be a primary process.

- *Secondary Business Processes* manage issues an organization has to deal with but do not directly contribute to the core purpose of the organization. Despite not directly furthering the main purpose, they are nevertheless necessary for the organization, either by maintaining the existence of the organization or contributing indirectly towards the core purpose. A typical example would be an accounting purpose, since it doesn't generate any product for the business but it is still required for the organization.

- *Tertiary Business Processes* are a category of meta-processes that deal with managerial issues arising from the coordination of primary and secondary processes. This includes organization of meetings and reports and adjustment of business strategies. If the term tertiary business processes is not used, they are counted as secondary business processes.

Schmelzer et al. also give a number of examples for primary and secondary business processes in [64]. According to them, primary business processes not only deal with the production of goods or provisioning of services like order execution processes and service processes, but also include innovation processes, product planning processes, product development processes and marketing processes.

Secondary business processes include controlling processes, quality management processes, IT management processes, resource management processes, financial management processes, human resource management processes and strategic planning processes.

The primary difference here is that primary business processes provide something for an *external customer*, meaning entities outside the business employing the primary business processes. Secondary business processes on the other hand deal with *internal customers*, i.e. customers who are actually a part of the company that uses those processes.

In [134] Porter notes that primary business processes are the business processes that generally convey a competitive advantage to a business, since they benefit external customers and are often integral to the core competencies of a business, to the point that business processes that are particularly tied to the core competencies and competitive advantage form the special class of *core business processes.* In contrast, secondary process may contribute to the competitive advantage but only do so indirectly. In addition, secondary processes are often required by many types of businesses and thus are usually standardized, allowing an opportunity for *business process outsourcing* (BPO), where a business process is performed by an outside entity (see [168]).

This approach mostly revolves around the idea of a core mission of an organization. However, sometimes it is more helpful to distinguish business processes based on their abstraction level. Here, Weske [161] offers five levels of abstraction, where more abstract levels describe the organization's approach in a broad but less detailed manner, while more concrete levels include more implementation details but have a narrower focus.

## 2.3 Organizational Challenges

Introduction of business process management is a challenging task for any entity, whether it is an organization just starting to structure itself or an organization with an existing structure attempting to introduce and formalize its business processes. In fact, business process management is only able to maximize its full benefits in a completely process-oriented organization (see Sections 2.1 and 2.1.2) as described by Schmelzer and Sesselmann as follows (translated from [64]):

> In the long run, business processes can only develop the desired performance in process-oriented structural organizations. As a guideline, structures should follow the processes and not the processes the structure. As a result, the introduction of business processes should be taken as a starting signal for a process of change with the goal of a process-oriented structural organization.

This statement emphasizes the need for restructuring organizations in order to ultimately achieve a process-oriented structure. Nevertheless, the authors also note the difficulty in introducing such an organizational form especially for organizations with pre-existing structures (translated from [64]):

> The replacement of a functional organization with a process-oriented organization is a far-reaching organizational intervention which in particular affects the people in position of the middle management layer. This strenuous effort has only been attempted by few companies and managers in Germany. This is demonstrated by organizational charts of German companies, which, as a rule, exhibit functions and only rarely business processes.
>
> [...]
>
> For many managers, thinking in terms of business processes is more difficult than thinking in terms of functions and hierarchies. Often managers and their employees are not yet prepared for their duties and responsibilities in process organizations. Exceptions are production and logistics, for which process-orientation is a reality in most cases.

In essence, business process management and process orientation introduces considerable change in any organization, as shown in [64]:

- Rather than functions and departments, business processes are now at the center of the organization – the organization changes.
- Instead of department heads, process owners are responsible for the business – positions and rules change.
- Only things that benefit the customer have value; everything else is a waste – activities and work content change.
- The employees control and improve the business processes by themselves – the roles of management and employees change.
- Instead of cost center budgets, the timing, quality and cost of the business processes are the fundamental operative control quantity – controlling and reference inputs change.

In order to introduce business processes in organizations, management has to have a different set of skills and responsibilities while their employees have to take on additional responsibilities which were not previously part of their skill sets. On top of that, a reorganization processes involves whole companies and is an extremely strenuous and risky process which, if it fails, can result in dire consequences for the organization both in terms of time, cost and morale. This risk can be high enough for the whole company to be at stake, resulting in managers seeking alternative solutions to mitigate risk.

This is often compounded by resistance from within departments to adapt the new organizational structures due to the persistence of power-oriented thinking in functionally-oriented organizations (see section 2.1). Departmental heads wield considerable power in functional organizations which they would necessarily have to partially surrender to the introduced business process manager. As shown in section 2.1, functional organizations actually implicitly encourage a power-centric culture, making it harder for the participants to cede power for the good of the organization and dismantle traditional departments in favor of business process-oriented competency center, in which their role is reduced to providing services for the business processes of the organization.

Resistance to the introduction of business process management and process-oriented structures can be grouped into two categories, personal resistances and organizational resistance, as show in [64]:

- Personal Resistances
  - Necessity of the change is not recognized
  - Goals, approaches and results of the change are misunderstood
  - Fear of the unknown
  - Fear of status loss
  - Threat to existing relations
  - Threat to existing work activities and habits
- Organizational Resistances
  - Incentives reinforce the status-quo
  - Threat to the existing power equilibrium
  - Conflicts between groups inhibit collaboration
  - Incompatibly of the change process and organizational culture
  - Resource commitments to past decisions and actions

Sometimes, especially in smaller organizations, at least some of these resistances can be overcome. For example, non-recognition of the necessity of the new approach and misunderstood goals, results and approaches can be mitigated using an information strategy that adequately conveys the intents of the process to the members of the organization. The fear and threat factors on the other hand are often elevated by ill-conceived attempts to introduce business processes as a mere cost-saving measure, which instantly sparks resistance to the perceived cost-cutting attempts, especially among rank-and-file employees rather than management. The following mitigating tools are provided in [64]:

- Information, communication and training
- Participation
- Relief and support

- Visible success
- Negotiation and proposing compromises
- Coercion and pressure

Aside from the already explained communication aspects, participation in the process is a key factor. A properly introduced business process management has a strong participatory component which involves helping and encouraging employees to point out flaws and help improve the business processes the organization uses (see [72]). This can be very motivating to employees since it empowers them to help shape and form the way their organization operates.

Relief and support is necessary if responsibilities start to exceed the capacity of participants to meet them. It must be recognized that business process orientation often comes with additional responsibilities beyond mere execution of labor which must be fairly shared in order to grow acceptance within the organization.

As soon as success of the approach becomes evident, resistance generally deminishes and participants become more eager to support the process of change.

In case resistance persists, two more options are available: The first is to negotiate with critical participants resistant to the proposed change and attempt agree on a compromise that is acceptable to all parties.

However, this may not always be possible, which leads to the option of last resort: coercion and force. Obviously, coercion and pressure should be the least favored options since it reduces morale and may even increase the support for further resistance. As such, it is only an actual option if the resistance is weak and not very serious or widespread.

Nevertheless, sometimes the costs, required effort and resistances overcome the will of an organization to introduce business process management. That said, business process management offers a number of approaches to mitigate this problem. For example, one popular technique is to retain the vertical departmental structure, thereby avoiding the associated resistance towards and the substantial effort to implement vast organizational change. As a substitute, organizations can introduce an additional horizontal layer on top of the existing business structure to represent the business processes and business process management. This approach is known as a *matrix process organization* (see [55]) since the process management layer is put in place on top of an orthogonal functional organization forming a two-dimensional matrix.

By retaining the familiar functional structure, the reorganization effort and the resistance by individual participants or the whole organization are avoided while nevertheless allowing for the introduction of business processes. All the organizational positions like the business process manager are available.

However, this approach is a compromise which, while avoiding the expense of transitioning to a full process-orientation, comes with a number of disadvantages:

- Responsibilities remain unclear. In a true process-oriented organization, the business process manager is responsible for the process and can enforce necessary changes in the service and competency centers to allow for a smooth execution of business processes. In a matrix organization, the departmental heads retain most, if not all of their power as well as the traditional incentives to protect their departments. The business process manager may be unable to enact changes which are vital to ensure efficient business processes.

- Deep hierarchies and monolithic departments are retained. The business process oriented management is only added to the management layers of the company. This means that a substantial amount of redundancy is built into the organization. The lean and flat management structures that are one of the goals of process-oriented organization cannot be achieved as envisioned.

As a result, a process-oriented organization is always more desirable over the compromise of a matrix organization. However, this goal may not actually be achievable, especially in very large organizations with considerable inertia. This means that while process orientation is the superior approach, real world business process management and business process management systems will have to deal with matrix organization types at least in the medium term.

## 2.4 Introducing Business Processes Management in Organizations

The right approach to introducing business process management into organizations depends on their individual situations. For example, a newly-formed organization has different challenges than a organizations with pre-existing structures (see section 2.1). The first step for introducing business process management is to identify the business processes which the organization needs. In general, there are two basic approaches used to identify business processes, a *bottom-up approach* which attempts to analyze the internal behavior of the organization in order to identify existing substitute business processes and the *top-down approach* which attempts to define new business processes.

The *business strategy* of the organization gives an outline for the overall organization and describes both the purpose of the organization and how the organization aims to fulfill that purpose. The business strategy therefore represents the most abstract level of business process management. A business strategy contains a number of useful information which can be used to identify business processes. Schmelzer and Sesselmann give the following examples in [64]:

- Target markets and customer groups
- Customer requirements, needs and expectations
- Competitive strategy
- Core competencies

- Critical success factors of the business
- Strengths and weaknesses of the business
- Business goals

The business strategy can not only be the starting point for a top-down approach but also offers a reference point for a bottom-up approach to the introduction of business process management. A business strategy describes the general approach of an organization; this then needs to be refined into something more tangible which the organization can perform. Since business goals are ideally derived from the business strategy, taking other factors such as target markets and customers into account, they represent the goals of the organization itself.

Ideally, business goals describe clear, achievable and sometimes long-term milestones for the organization to achieve. They are determined using the business strategy; moreover, if all business goals are found and defined, they collectively represent a successful implementation of the business strategy once accomplished.

The bottom-up approach usually involves a process called *Business Process Redesign* or *Reengineering* (BPR, see [35]). The first step of this process is to identify existing business processes within the organization. This can be accomplished by creating a *process map*. These maps highlight the current business processes in the organization, the resources they use as well as the participants and departments that are assigned to them. This process map represents an organization's status-quo and the starting point for the bottom-up process.

In the next step the processes are analysed in detail and the existing processes are compared with the business strategy to identify divergences, redundancies and waste; for example, by identifying redundant activities or even entire business processes that are unnecessary and eliminating them. This is part of another research and practice field called *Business Process Optimization* (BPO, see [158]), which is, however, not the focus of this work. In addition, IT capabilities are considered and how they could benefit the business processes.

After the business processes are analyzed in detail, optimized, then either adapted or eliminated to match the business strategy, a new process map is created to determine the goal of the BPR for the organization. This new process map is then balanced to determine the necessary resources for the organization. A useful tool to be employed in this process is the balanced scorecard (see [94]), which highlights key data of the organization with regard to its business strategy (see [93]).

The top-down approach on the other hand does not take the existing business processes into consideration. Instead, the business strategy is used directly to determine the business processes. Since the business strategy is often very abstract, so are the first business goals derived from it. These *top-level business goals* often only describe very large-scale targets in very broad terms for an organization, such as becoming the innovation or cost leader in a certain market or providing certain goods and services to customers.

As a result, this first level of goals does not yet provide concrete instructions for the participants of an organization and, in fact, often need to be further refined to formulate concrete actions. This can be done by deriving subgoals from the top level business goals, which, just like the top goals represent the successfully implemented business strategy, represent the successfully achieved top level business goal. If the derived subgoals are still somewhat unclear, they can yet again be further refined into another layer of subgoals until the exact targets and outputs can be determined.

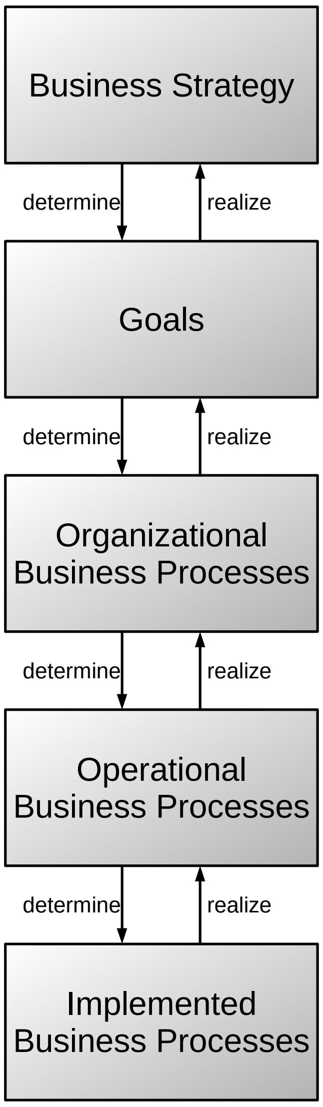

Business Strategy
determine
realize
Goals
determine
realize
Organizational Business Processes
determine
realize
Operational Business Processes
determine
realize
Implemented Business Processes

Figure 2.3: Business process management starts at the strategic level of business planning and extends down to the operational level, where processes can be (partially) automated (based on [161])

After the business goals are determined, the next, more concrete layer is called *organizational business processes.* This typically represent the first layer where business processes are used and process models become important (cf. 2.3). Here, the steps necessary to achieve a business goal are determined along with the resources, particpants and the necessary order or sequence. Typically, this is done in an informal manner

Nevertheless, the models used at this level are usually still informal models, describing the inputs, outputs and the result of the processes. These processes help an organization to provide the resources and structure necessary to achieve the business goals. For example, one can determine the employees necessary, how they are supposed to be organized, what suppliers are needed and how to ensure necessary resources are available.

The *operational business processes* on the other hand describe in detail the process steps that are required to match the desired output and result of the business processes. However, since they lack implementation details, such as detailed task instructions, guidelines or automation code, they cannot themselves be implemented in the organization.

This step is accomplished by the *implemented business processes*. This type of business processes include all necessary infomation for the organization to execute the process. By itself, this does not imply automation (executions by machines rather than human actors) but can include any degree of automation.

The processes of introducing business process management into organizations can be assisted by software systems. The next section will provide a definition of such systems and describe how they can assist in the successful introduction and maintenance of business process management within organizations.

## 2.5 Business Process Management Systems

Introducing business processes and applying business process management to an organization can be an involved process due to the complexity of typical organizations. However, it can be employed by organizations in a purely manual fashion, in which the people within the organization who are involved in the processes use their institutional knowledge, training and manual approaches based on paper documents to implement business process management. While identifying, formalizing, analyzing and improving business processes in organizations can be done on paper or through meetings and discussions, for most organizations the complexity of such a task quickly exceeds any reasonable effort. In order to reduce the effort necessary, the introduction and maintenance of business process management can be assisted with software systems. This can start with the use of common office automation tools such as spreadsheets, word processing or e-mail. However, if a generic and integrated software system is used to assist business process management, the system is referred to as a business process management system, as defined by Weske in [161]:

> A *business process management system* is a generic software system that is driven by explicit process representations to coordinate the enactment of business process.

Note that this definition does not require the business processes themselves to be automated or even formalized. It merely requires that the systems supports business process management through explicit process representation. At this point it is useful to make a distinction between *models* and *instances.* Models represent the static nature of an entity type, i.e. the structure and information any entity of the model's type contain, while the instances are a concrete occurence of the entity type. For example, a blueprint for a house represents a model for a house, while a built-up house is an instance of a house. This pattern is also common in object-oriented programming, with classes representing static models and objects representing instances of the model (see e.g. [109]).

A similar notion applies to business processes. A business process model describes the entities involved in the process and the general plan to be followed when the business process is performed. A business process instance on the other hand is a current performance of a business process in progress working its way to completion within the organization. In terms of information content, unless the business process itself changes, the business process model stays consistent and does not need to be modified. Contrary to this, the business process instance has to be provided with information relevant to the current performance of the process such as details about the order like color or information about the customer. Weske defines business process models and business process instances in [161] as follows:

> A *business process model* consists of a set of activity models and execution constraints between them. A *business process instance* represents a concrete case in the operational business of a company, consisting of activity instances. Each business process model acts as a blueprint for a set of business process instances, and each activity model acts as a blueprint for a set of activity instances.

The creation of new business process instances in a business process management system is referred to as either an *enactment* or an *execution* of the business process or business process model. After enactment, the business process controls the workflow instance and ensures the correct performance of the workflow as specified in the model.

In many cases, after identifying and modelling business processes, part of the improvement of the processes is the full or partial automation of the business processes so that IT systems can assist the enactment and execution of business processes and thus reduce errors and cost while improving quality. The next section will provide an introduction to this process and describe how IT systems can assist business process management and organizations that use it.

## 2.6 Business Processes and Workflows

Business process, as defined in section 2, describe a set of coordinated activities in an organization that are aimed at reaching a particular (business) goal or goals. Once they have been identified, they can be modelled in business process models as described in section 2.5. However, while business process models describe a sequence of actions and tasks performed, they do not specify how information is passed or exactly how the sequence of actions are coordinated.

Technically, this can be done in an entirely manual fashion by printing, passing and mailing business documents and coordinating over communications media like phone or e-mail. Unfortunately, this puts an additional burden on the employees who already have increased responsibilities in business process-oriented organizations. Furthermore, these coordination tasks are often relatively mundane and require no particular skill, thereby wasting the time and effort of employees who have been trained with considerable cost and effort in order to implement business process management.

Many actions and steps in a business process can and often already are automated, in many cases even before the introduction of business process management. For example, a step in a business process to weld a car frame can be executed by an automated assembly system instead of manual labor.

However, it may also be beneficial to support the automation of the execution of the business process itself as well as the action of passing outputs from one process step to the next. While it may not be always possible to fully automate an entire business process, at least parts of it generally lend themselves to automation.

When a business process is partially or fully automated, the result is referred to as a workflow. A large standards body in the field of workflows is the Workflow Management Coalition (WfMC), which, along with Weske (see [161]), define workflows in relation to business processes as follows:

> Workflow is the automation of a business process, in whole or in part, during which documents, information, or tasks are passed from one participant to another for action, according to a set of procedural rules.

Workflows are therefore a refinement of the concept of implemented business processes (see section 2.4). Instead of simply implementing the business processes, workflows automate, at the very least partially, the coordination of activities within a business processes. This means that unlike business processes, workflows necessarily imply the use of IT systems in some fashion and may even encompass the full automation of a business process.

However, as noted, full automation is not always possible. As a result, conversion of a real world business process to the IT-based workflow may be incomplete and the result may not actually represent the full business process which may consists of additional activity outside the workflow.

The terms used to describe aspects of business processes have equivalents in terms of workflows with the narrower scope of the technically implemented and automated parts. Since workflows necessitate IT support, it follows that an equivalent to business process management systems must exist for workflows. These integrated systems for creating, managing and enacting workflows are refered to as workflow management systems (WfMS), which are defined by Weske in [161] as follows:

> A *workflow management system* is a software system that defines, creates, and manages the execution of workflows through the use of software, running on one or more *workflow engines*, which is able to interpret the

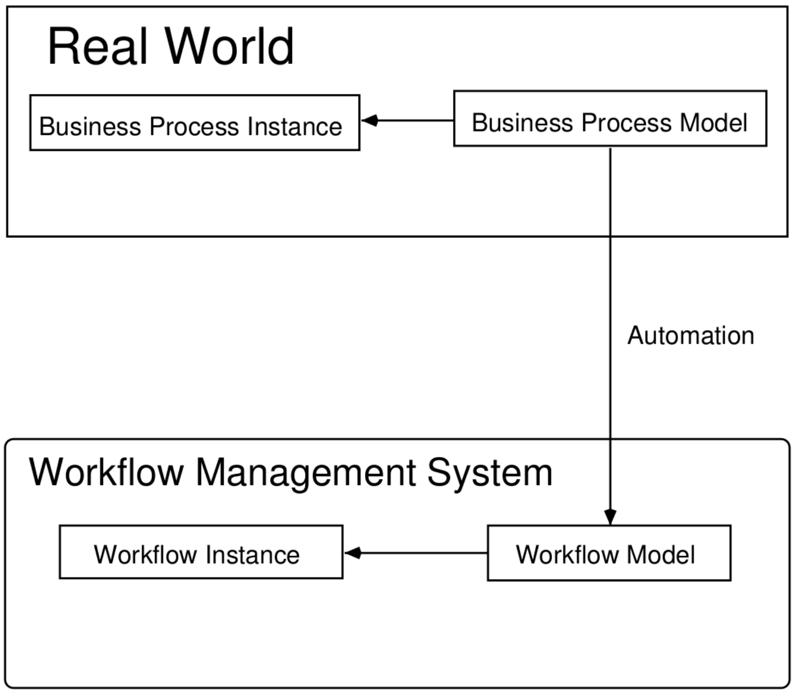


Figure 2.4: Business processes are based on the real world organization. If a business process model is automated, it becomes a workflow model (from [80] based on [100])

process definition, interact with workflow participants, and, where required, invoked the use of IT tools and applications.

This means that once a business process model exists, it can be converted into a *workflow model* by adding the necessary information for an automated workflow engine to interpret it once it has become a *workflow instance* (see 2.4) or *case*, according to van der Aalst and van Hee in [1]. According to them, a case or workflow instance differentiates from the workflow model in that they carry a particular *state* during their lifetime, consisting of the following three elements:

- Case attributes are the part of the workflow state which allows the workflow engine to determine whether a certain task should be skipped or executed. During execution of the case, the state of case attributes may change.
- Conditions specify which tasks have finished their execution and which are still in line to be performed as part of the workflow. They defined the state that is necessary for a task to be carried out.
- Content is business information relevant to the case such as files, documents, databases and archives. In many cases, the content is not contained in the workflow management system but provided by an external source.

While the definition by Weske does not specify how exactly a workflow instance is interpreted by workflow engines, the most common approaches for workflow models are based on tasks, which represent steps in both the workflow and the underlying business process which need to be performed for the workflow to be successfully executed.

Tasks represent a unit of labor which cannot be further deconstructed into smaller pieces and must be fully performed to be considered complete. This means that tasks within a workflow are *atomic*, meaning they are either performed in full, or, in case of failures, the changes done by the task up to that point have to be reverted to the original state (*rollback*).

Tasks themselves do not always require automation. The definition only requires that the coordination or execution order of the steps is performed by a workflow management system. In fact, van der Aalst and van Hee offer three distinct categories of tasks in [1], only one of which is fully automated:

- *Manual tasks* are performed by a person such as an employee or a group of people without the use of software support. Typically, these are tasks that are not easily automated because they require some sort of physical interaction, for example physically fastening a screw or signing a paper document with a pen. In this case, the workflow management system is reduced to simply providing a prompt to the participant or participants to perform the action and confirm the execution.

- *Semi-automatic tasks* are performed by a person or a group of people who are supported by a software application or a similar piece of software to perform the task. This can involve creating or modifying a document with a word processor or filling in a sales form with a customized application. Here, the workflow management system can not only prompt participants to perform the action but also execute supporting applications as necessary, provide the data for the application and pass the results to the next part of the workflow after the task has been performed.

- *Automatic tasks* are units of labor that can be performed without any human interaction at all. Depending on the context, this can mean different things: For example, the task may invoke an automated service implemented in software such as a task which accesses a database. Alternatively, a task may also involve physical labor if the labor is actually performed by a robot or some other piece of automated production machinery that can commonly be found in assembly lines. This also means that a task can be performed simply by using information made available by the workflow management system or external, IT-based systems. A human is not required to provide or interpret additional information.

Once a task is in execution by the workflow engine, the task, which is a workflow model element, needs to be instantiated with the information available in the case or workflow instance that is relevant to the job that the actual labor of the task requires. The resulting instantiated task is called a *work item*, and represents a concrete action or actions that need to be performed in order to advance the workflow instance.

However, the work item is still a passive object, representing a work description. In order for it to be performed, some entity, for example a human in the case of a manual and semi-automatic task, alternatively a software system or industrial robot in case an automatic task, has to receive the work item and begin performing the necessary actions. Once the work item has been assigned to such a *workflow participant*, the work item is regarded as an *activity* while the work is in progress.

## 2.7 Business Process Management and Workflow Management

While business processes represent the real world processes as they appear in organizations, workflows represent a subset of such business processes that have been automated using IT systems. As a result, workflows often only represent a subset of the whole business processes. Consequently, the approach for business process management and workflow management overlap but differ in portions where the wider view of business process management comes into play.

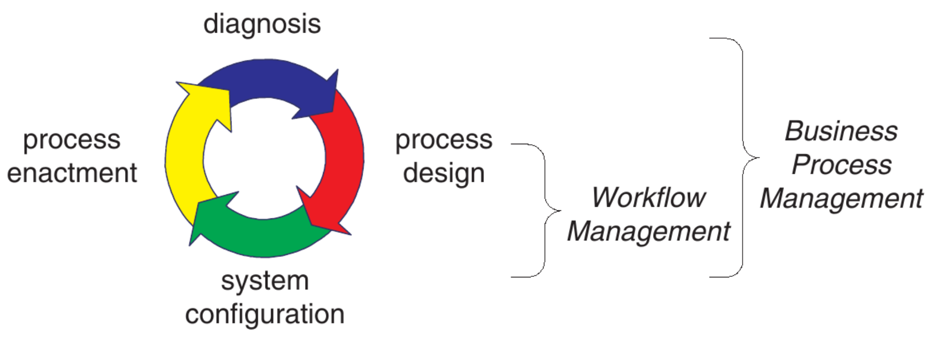


Figure 2.5: Since workflow management focuses on the execution part, business process management differs by including a feedback from diagnosing existing business processes and comparing them to strategic planning (from [155])

Practical business process management is often represented in the form of a lifecycle that can be followed to initially design new or model existing business processes. Since the actual requirements for such a lifecycle differ in detail, many variants exist. For example, a more workflow-centric cycle may emphasize the technical aspects; a simple example of such a lifecycle is shown in Figure 2.5, which was used by van der Aalst in [155] to demonstrate the difference between workflow management and business process management. It divides the lifecycle into four phases. First, the process design where the business process or workflow is modeled in an appropriate business process or workflow modeling language.

Second is the system configuration, where the business environment is prepared for enactment of the business process or workflow. For example, in the case of workflow management, this includes the set up of the preparation and configuration of relevant IT systems, configuration of the workflow management system and including the workflow model in the system. In business process management, this can also include other aspects such as organization of resources and labor.

The third phase encompasses the process enactment. Here, one or more business process instances or workflow instances are created and then performed as designed. For workflows, this includes the execution of the workflow instance on the workflow management system.

The final phase of the lifecycle presented by van der Aalst is diagnosis. Here information gathered during enactment of the business process or workflow is collated, then analyzed. The goal of this phase is to use the experiences gained from the enactment and performance of a model under real circumstances in order to improve the design of a business process or workflow. In particular, the diagnosis is used to evaluate the business process with regard to the organization's business strategy. If the business process results deviate from the goals of the strategy, the issue can be identified with the information processes used during the diagnosis phase which allows the business process model to be adapted to be more closely aligned with the goals of organizations.

Since the diagnosis phase cannot be automated because it requires evaluation by human actors, it is the main differentiating factor between business process management and workflow management. While a workflow management system can certainly gather information which is later used in a diagnosis phase, it cannot provide an evaluation of such information beyond fairly simple threshold and assertion comparisons.

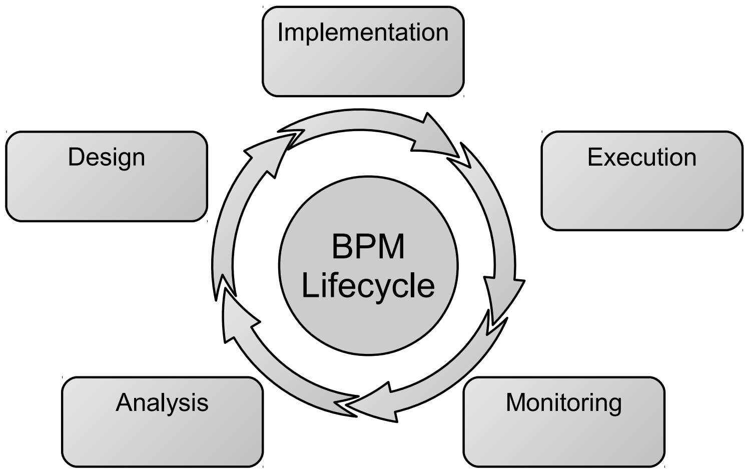


Figure 2.6: Since workflow management focuses on the execution part, business process management differs by including a feedback from diagnosing existing business processes and comparing them to strategic planning (from [155])

Long-running autonomous processes have both long execution times and are often used only once before being evaluated. Therefore, the importance of the diagnosis phase of the lifecycle is generally reduced and used much more rarely with regard to such process types when compared to production workflows. Nevertheless, some diagnosis is still helpful and necessary, both to diagnose potential problems in advance by enacting and simulating test instances as well as a source for gaining organizational knowledge, even if the model is not reused exactly.

Based on this, the approach presented in this work emphasizes the workflow aspects of business process management while still providing some aspects of the diagnosis phase. Therefore, a more technically-centered and workflow-oriented lifecycle model is used as shown in Figure 2.6 and further explained in [87].

Here, the lifecycle is partitioned in five phases. The first phase is the design of a business process model. This phase emphasizes the strategic planning aspects of the process design in order to ensure a strong cohesion with the business goals. The resulting model is not executable but expresses the business goals of the organization as close as possible. This phase requires an business process or workflow modeling language which is able to describe the process behavior while maintaining a strong cohesion with strategic planning.

The second phase is the implementation of the workflow model. This phase starts with the business process model gained in the design phase and enhances the model with additional runtime information with the goal of producing an executable workflow model. This enhancement should be performed in a machine-readable format in order to allow the resulting workflow to be executable. In order to support a workflow designer in creating such workflows, editing tools are required that are capable of reading and writing in that format.

The execution phase of the lifecycle shown in Figure 2.6 is the most technically challenging. It involves using the workflow model generated by the editing tools and enacting workflow instances based on that model. This is usually accomplished through utilization of a workflow engine which interprets the model and runtime data and performs the execution. However, a workflow engine is insufficient, since interactions with users and automated services is required. These tasks are usually accomplished by various parts of a workflow management system which includes the workflow engine as a crucial component.

Another key aspect of workflow management systems is the monitoring, which is represented as the fourth phase in the lifecycle. During enactment of a workflow instance various information can be gathered about the workflow behavior such as start and duration of tasks, enactment of sub-workflows and interactions of workflow participants with the system. The monitoring gathers such information and stores them for later use.

Finally, the analysis phase uses the information gathered from workflow instances to help workflow designers to evaluate workflow models. This can include both gathering information during a real world enactment of a workflow as well as enacting workflow instances in simulated environments to gauge their behavior. The experiences gained during this phase can then be used in the design process to improve the workflow models as well as evaluating how well the result match an organization's strategic goals.

The research goals regarding improved support for long-running autonomous process are not all affected by every phase of the BPM lifecycle used here. As would

| | Design | Implementation | Execution | Monitoring | Analysis |
|---|---|---|---|---|---|
| **Strategic-operational Cohesion** | X | | | | X |
| **Workflow Model Agility** | X | X | X | | X |
| **Balance Global Control/Local Autonomy** | X | X | X | | |
| **Organizational Agility** | | | X | X | |
| **System Robustness** | | | X | X | |
| **Workflow Instance Agility** | | X | X | | |

Table 2.1: Now all phases of the BPM lifecycle affect the research goals for additional support of long-running autonomous processes

be expected and shown in Table 2.1, the BPM lifecycle as a whole affects all of the research goals, but each phase affect only some of them.

The design phase emphasizes strategic planning and does not yet concern itself with execution. As a result, the most critical research goals for this phase are strategic-operational cohesion to form a strong connection with the business strategy and a workflow language which will allow inclusion of predictable changes at runtime to be easily included in the model.

The output from this phase is only a business process model tied to the strategic goals of the organization. The execution of a later workflow model is important in so far as the generated business process model should not impede a later effort to generate a workflow model. Consequently, the workflow model agility aspect is already relevant in this phase. Balance of global control and local autonomy is partially affect by this phase since the resulting model must retain the necessary amount of autonomy for the workflow participants. Since the execution aspects are not part of this phase, organizational agility, system robustness and workflow instance agility are not relevant in this phase.

During the implementation phase, process models resulting from the design phase are enriched to mold them into workflow models by preparing them with additional information for enactment. Strategic-operational cohesion has been accomplished by the previous phase and the current phase must simply maintain that cohesion so that the resulting workflow model does not lose its connection to strategic planning.

Additionally, workflow model agility remains important as the process model is prepared for execution and detailed runtime instructions are added which include contingency options for predictable changes in the business environment. The implementation phase also offers the opportunity to include possibilities for later workflow instance agility.

The execution phase of the BPM lifecycle is affects most of the research goals. While the strategic-operational cohesion is supported by the generated workflow model at that point and does not require explicit support by the runtime system, all other research goals need support from the workflow management system during the execution phase in order to be achievable. For example, the workflow engine must be able to execute the workflow model resulting from the previous two phases. In particular, the contingencies included in the workflow model for handling predictable changes in the business environment must be invoked by the workflow engine whenever necessary. Adapting workflow instances can only be adapted for unpredicable changes in the business environment during runtime, requiring this task to be performed during the execution phase.

Balancing global control and local autonomy becomes a practical issue at this point when the autonomous workflow participants begin to interact with the workflow management system during the execution phase. In addition, organizational agility becomes important since the system must allow parts of the organization to manage and administer parts of the system independently from one another. Finally, due to the long execution times of the processes, the robustness of the system is key during this phase.

For the monitoring aspect of the BPM lifecycle, two research goals are primarily affected. First, the monitoring must be able to adapt to different organizational models by allowing each part of the organization to perform independent monitoring. In addition, since real world enactments of workflow should also be monitored during their long execution times, system robustness is once again key during this phase.

Finally, the aspects of workflow model agility and strategic-operational cohesion are again elevated in importance during the analysis phase. The analysis must take the flexible nature of the workflows into account and present the monitored data in a way that allows workflow designers to reconcile the flexible execution path with the designed workflow model. In order to allow a comparison with strategic planning, the strategic-operational cohesion once again becomes critical at this point. This means that either the cohesion must be maintained during the whole lifecycle or the system must be able to restore this connection during this phase.

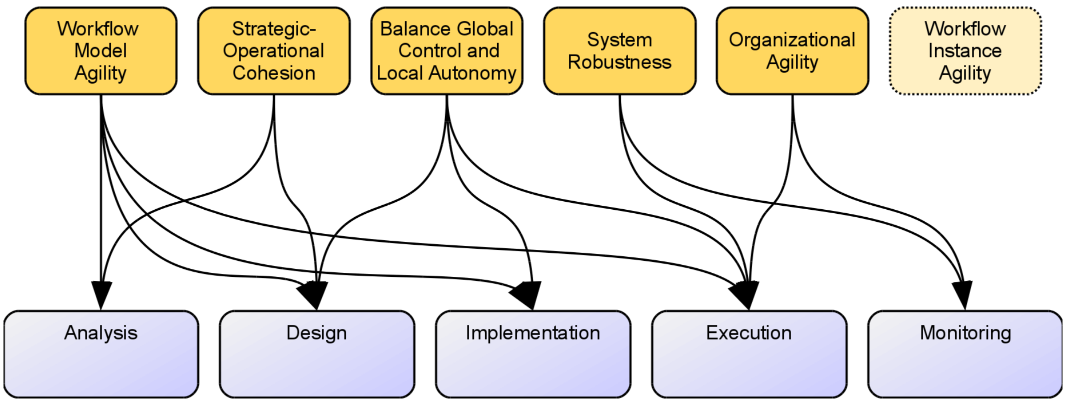


Figure 2.7: The different phases of the BPM lifecycle are affected by the research goals to varying degrees, the primary influence being shown here. Solutions have to be available to address the research goals and represent the whole lifecycle.

The primary influence of the research goals on the lifecycle phases are shown in Figure 2.7. Since business process management encompasses the whole lifecycle, a solution for every phase has to be available that take the research goals into account to adequately support long-running autonomous business processes.

In order for both business process models and workflow models to be designed and implemented, a language has to be defined to express all parts of the workflow. A large number of such languages are available. The next chapter will present a number of options in the area of business process and workflow languages, evaluating them with regard to the research goals of this work. Following that, a new workflow language specifically aimed at long-running autonomous business processes will be presented which also incorporates one of the more common workflow languages available.

## 2.8 Service-oriented Architecture (SOA)

The service-oriented architecture approach (SOA, see e.g. [44], [142], [43], [42] and [97]) is a design pattern for developing applications, especially business applications, on a large and distributed scale in complex and heterogeneous environments. It represents a relatively recent approach for both programming-in-the-large (see [38]) and the development of distributed software systems.

Since many modern business process management systems require the integration of many divergent systems from potentially differing organizations or parts of organizations, SOA is often used in the implementation of business applications related to business processes and is therefore often considered to include concepts and approaches from business process management.

SOA represents a design pattern for developing large-scale applications and is supported by a large number of software development companies, vendors and software engineering researchers. This means that it encompasses a large set of overlapping and sometimes competing concepts and technologies which can be used to develop SOA applications. This section will therefore give a brief overview of the basic concepts and ideas as well as some of the more common techniques to develop SOA applications.

The core concept of SOA is the idea of a service, which represents an encapsulated piece of code capable of performing activities by implementing service operations and offering them in a defined manner to the rest of the system. This allows the implementation of applications and further services by interacting with already existing services to implement the desired functionality.

In order to employ a service-oriented architecture, certain functionality must be provided by technologies to enable this approach. The functionalities are often summarized using the SOA triangle shown in Figure 2.8. Services are used by some sort of service used, denoted as ”service requester” in the figure. This service requester can be anything that requires a particular service such as an application or workflow but can also be another service which uses the requested service as part of its functionality.

Before a particular service can be used, it has to be found. In a SOA approach, this is accomplished by a service broker which has the means of locating services,

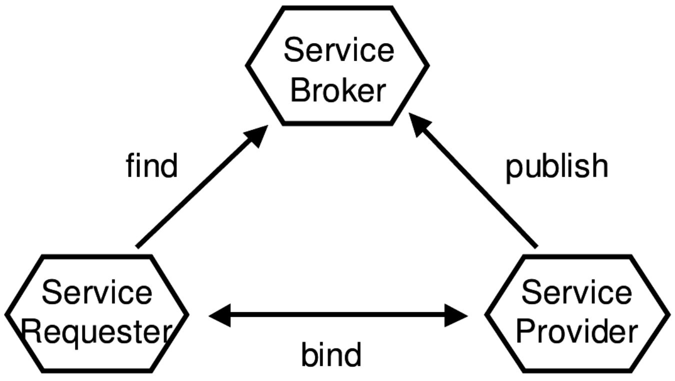


Figure 2.8: The SOA triangle illustrates the functionality required for SOA approaches (from [110])

usually by maintaining a searchable set of available services. This set of services is compiled with the help of the service providers offering services by publishing and therefore offering their services through the service broker.

Finding services that fit the requirements of the service requester often is the most difficult part in service-oriented architectures. This is due to multiple factors: First, the requested service must match syntactically with the service requester and cannot require the provisioning of more information than the service requester expects to provide or offer a different type of response from that which the service requester requires. This aspect can often be reasonably be assured through an adequate typing system.

Second, a service must fulfill the semantic expectations of the service requester. For example, both an addition and a substraction service may require two integer values and return a single one, however, if the service requester requires one and binds the other, the results will not be as expected. Defining such semantic descriptions tends to be notoriously difficult due to lack of standardization and importance of details, raising question whether the addition service supports negative numbers as well or how overflows are wrapped or alternatively if they are clamped.

Finally, once an adequate service is found, it is bound to the service requester which can now issue service requests to the bound service. Service bindings can be both bound permanently, rebound only if the current service is lost or bound dynamically with each use.

Services in an SOA environment are ideally designed to maximize reuse and minimize the necessity of changing the service implementations in favor of combining and changing the way the services are used. Services generally implement larger-scale operations while still being universal enough to be applied in a variety of contexts. The services are loosely-coupled and they abstract from the underlying operation system and other technical details of the system used to implement them.

In contrast to traditional software libraries, services do not offer a language-specific interface. Instead, it defines the protocol and functionality of the service. Depending on the details of the technologies used, this allows the transcendence of language and system barriers.

The specific technologies that can be used to implement an SOA application are not specified. Often established technologies from distributed system development are used to specify, implement, discover and combine services when an SOA approach is used to develop applications. A prominent approach used to specify and communicate with SOA services is the specification as web services, but alternatives are available including remote function call/method invocation approaches along the lines of Java RMI (see [76]) and CORBA (see [119]).

However, approaches based on web services are often the more popular ones due to the widespread support and universal support across systems, middleware and programming languages. Many of the earlier approaches used the Web Service Description Language (WSDL, see [167]) as the service description language, the Simple Object Access Protocol (SOAP, see [63]) as the communication protocol and Universal Description Discovery and Integration (UDDI, see [10]) registries as service brokers.

More recent approaches often favor the use of the Representational State Transfer (REST, see e.g. [44]) approach to web services, which employ the Hypertext Transfer Protocol (HTTP, see [45]) as the communication protocol, avoiding the considerable overhead of the earlier WSDL and SOAP-based approaches and allows for easier integration with standardized web server environments.

In the context of business process management, SOA approaches are often used to encapsulate business services provided by the competency centers of process-oriented organization (see section 2.1.2), making them available to the organization at large as well as potentially interested outside parties. It also allows the automated coordination of such services using workflows.

The next chapter will introduce available business process modeling languages. The chapter will evaluate their viability for modeling long-running autonomous processes based on the research goals defined in section 2.7 and general criteria that facilitate a business process-oriented approach.

# Chapter 3

# Business Process Modeling Languages

Since a major aspect of business process management involves the identification of business processes in organizations as well as defining their properties, languages are required to build models of business processes once they have been identified. A modeling language can also offer extension points for partial automation of business processes in the form of workflows.

Any business process modeling language which also aims to support workflows struggles with the somewhat divergent demands of two categories of users (see [24]): The business user, who tends to provide expertise from a business perspective and the context of the business process within the organization and the technical user who is versed in the knowledge of IT systems and software. This means that a major aspect of the art of translating business processes into workflow centers around bridging this gap of understanding between these two group of users.

As a result business process modeling languages can be divided into three categories (see [102]) which address this dichotomy in different ways: First, *informal* modeling languages allow the representation of the business process in informal terms which explain the process to another human being but the semantics of which are inaccessible for a machine. In addition, details necessary for an automated execution is often omitted in order to reduce complexity.

The simplest representation of this category of modeling language is natural language, which can be used to simply describe a business process in written form, however, since natural language often fails to quickly provide an overview of a business process, languages with higher expressiveness and precision are available in this category. Since this category of modeling languages lacks a formal definition of the semantics, it cannot be automated and thus used to model workflows and are only used to get an overview of a business process as well as a stepping stone on the path to a workflow model in a different modeling language. The advantage of informal languages is that they are easily accessible to a technical layperson and thus allowing a wider participation of stakeholders while designing processes in this category of modeling language.

On the other side of the scale are *formal* business process and workflow languages. In this category of modeling languages, all modeling elements are assigned a strict formal semantic definition. As a result, once a business process is fully modeled using a formal business process modeling language, it becomes executable on a suitable IT-based system and therefore can be instantiated as a workflow instance. Formal modeling languages require model designers to include considerable detail in the model in order to support execution of the model. Furthermore, a formal language can be restrictive in how its language elements can be used to ensure that a valid execution semantic exists for the model. As a result, formal modeling languages are generally less accessible than informal ones to a layperson who is not well-versed in the technical details and semantics of that language and lacks requisite expertise to add necessary execution details.

The final category of business process modeling language are *semi-formal* modeling languages. This category of languages attempts to strike a middle ground between formal and informal approaches with the aim of being accessible to technical laypersons as well as allowing some form of automation at a later stage. As a result, while many elements of such languages have a formal runtime semantic, others do not and are merely used in an informal context. This allows the creation of both informal overviews of business processes as well as the creation of execution-ready workflows.

The semi-formal approach also offers a more seamless path from the first assessment of a business process to a formally designed workflow: Directly implementing a business process in a formal language may overwhelm a designer by forcing him to formalize small execution details from the start. On the other hand, first assessing a business process in an informal modeling language and then then translating it into a formal workflow model involves a break in the language used to model the business process, which, since the step is manual, risks the loss of information in the transfer and increases the chance of introducing errors.

Semi-formal languages on the other hand allow for an informal rough draft of the business process model to be created which can then be successively enhanced until a fully-formalized model is achieved. The disadvantage of this approach is that it necessitates a compromise between the two extremes. On the one hand, the execution environment does not support all language features and elements, which can lead to a workflow designer finding surprising failures while executing a workflow instance based on the workflow model. On the other hand, even the partial executable semantics of a semi-formal language puts restrictions on how that language can be employed which makes it harder for a layperson to use.

An alternative way of differentiating business process modeling languages is based on the representation of the language. A modeling language can use a *textual* representation to express the details of a given business process. Languages with textual representations are relatively compact and lend themselves to modification using only standard text editors, though specialized editors with specific language support are sometimes available to assist a model designer.

However, the more common approach is a *graphical* language, in which most of the major language elements have a graphical representation. The advantage of graphical languages over textual ones is the facility with which they can offer a quick overview of the business process and are often considered to be more intuitive for laypeople.

These two sets of categories can be used to categorize business process modeling languages. For example, using natural language to describe a particular business process of an organization represents the use of natural language as an *informal* and *textual* business process modeling language.

In chapter 1 and chapter 2 it was noted that many aspects of business process management primarily address production and administrative business processes. As a major aspect of business process management, process modeling languages are no exception in this regard: Mainstream business process and workflow modeling languages usually center around a task-based approach, where the business process is defined by a sequence of tasks and variants are usually implemented using a number of splitting elements to allow for different sequences to be taken depending on the process state. These types of modeling language form the group of *task-based modeling languages* and will be address in the first part of this chapter.

However, while the group of task-based modeling languages are well-suited for the aforementioned group of business processes, they are often not well-suited for a number of business processes including collaborative business processes (and long-running autonomous business processes in particular, see section 1.2), as well as more exotic examples of both production and administrative processes. As a result, a number of languages has been developed to address these types of business processes. They will be introduced in the second part of this chapter, along with the goal-oriented approach developed to support the long-running autonomous processes which this work aims to address.

## 3.1 Task-based Business Process and Workflow Modeling Languages

Task-based business process and workflow modeling languages represent the most common approach to modeling both workflows and business processes. The approach is based on identifying small and uninterruptible (atomic) units of work (*tasks*) of a given business process and arranging them in a sequence describing the order in which the tasks must be performed for the business process to be successfully completed.

Since not every instance of a business process involves the same tasks (process variants), task-based languages offer modeling elements for branching the flow of execution in some form, the most common being a choice between two or more sequences of tasks, one of which will be chosen in any given business process instance. Another frequently included language element selects two or more sequence flows following the element which are then executed concurrently.

For many business processes, this approach is fairly intuitive since it resembles the use of informal methods of process planning: A common approach for planning a complex project is often informally accomplish by compiling a „To-Do List" which simply lists step-by-step instructions explaining how the project can ultimately be finished. As a result, these lists are a primitive form of this approach but lack the more complex elements which allow parallelism and variants (though it can be accomplished with this approach in a limited fashion by making some steps optional).

This section will give an overview of task-based business process and workflow modeling languages and as well as categorizing them in terms of both formality and representation.

## 3.2 Flowcharts

An early example of a graphical modeling language is the flowchart. Introduced by Gilbreth (see [58]), they were later standardized by the American Society of Mechanical Engineers (ASME, see [6]). Since they were one of the first examples of a graphical modeling languages for business processes, they were quickly adopted to model various industrial processes. Out of the flowchart, a number of derivative modeling languages and language extensions have been developed.

Depending on the exact variant of the language used, a large number of language elements are available; however, common elements include a rectangular shape for units of labor that are to be performed which are often referred to as an *activity*. A rhombus shape represents a *decision*, where the user of the chart must choose between one of the outgoing paths available depending on the context.

Other common symbols include a parallelogram for *inputs and outputs*, a hexagram for *preparatory work* and a downward-pointing pentagram as a reference to another page for multi-page flowcharts. Flowchart symbols are connected using arrows or flowlines, which the user of the flowchart follows while performing the process described by the chart.

Figure 3.1 shows a simple process modeled using a flowchart. The process describes how to troubleshoot a non-functioning lamp. The user begins by starting at the top where the issue is described and is then met with the first decision in which the user verifies if the lamp is plugged in. If the user answers the decision with „No", the next activity describes how to resolve the issue (by plugging in the lamp), otherwise the user proceeds to the next question. Eventually the user will reach one of the final activity nodes describing how to proceed after the problem has been identified by the user using the flowchart.

The instructions on the elements are written in natural language and therefore lack the necessary formality for IT-based execution. This is unsurprising considering that flowcharts were designed to assist humans in formulating processes for other humans and IT systems capable of performing workflow execution in the modern sense which were not available at the time they were proposed.

However, flowcharts distinguish themselves from pure natural language descriptions by offering an intuitive graphical representation of business processes and there-

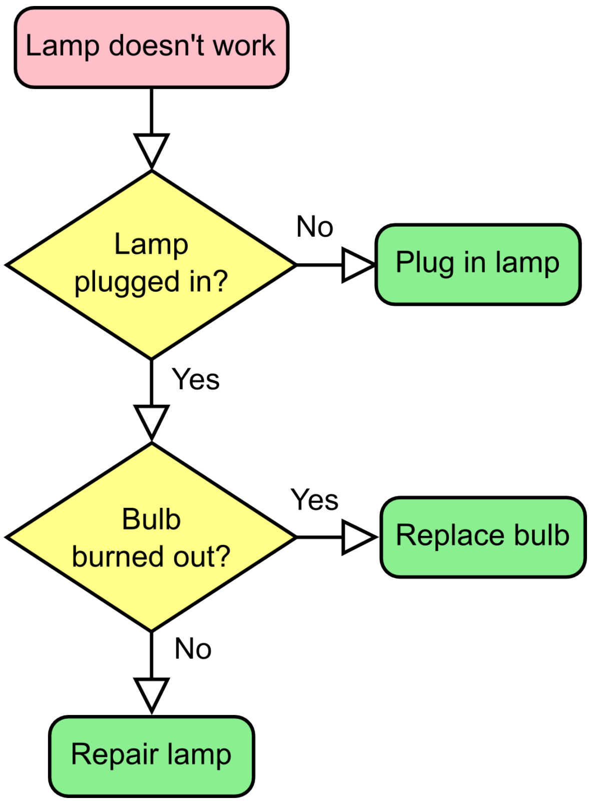


Figure 3.1: Example of a flowchart illustrating the troubleshooting of a lamp (from [126])

fore are an example of an informal and graphical business process modeling language. Furthermore, due to their popularity they often serve as a starting point or inspiration for more formalized task-based business process and workflow modeling languages.

## 3.3 Workflow Languages based on Petri-nets

Petri nets as a modeling language precede the idea of business process modeling in the modern sense. Originally proposed by Petri as a language for modeling chemical processes, it was found to be useful for modeling distributed and otherwise concurrent systems (see [137]). The model consists both of a graphical representation and a rigorous mathematical definition of the semantics.

The basic petri net graph or, in the sense of business processes, model consists of a very limited number of elements, consisting of the following three-tuple:

$$(S, T, W)$$

Element $S$ represents a finite set of *places* and element $T$ being a finite set of *transitions*. The sets $S$ and $T$ are disjunct and thus places are never transitions and transitions are never places. $W$ represents a multiset of arcs which assigns a multiplicity to each arc as follows:

$$W : (S \times T) \cup (T \times S) \rightarrow \mathbb{N}$$

This multiset allows arcs to be defined between places and transitions and assigns them a multiplicity, however, arcs between two places or two transitions are not included in the multiset and are, as a result, not part of the language definition.

Compared to a petri net graph, a petri net or, in terms of business processes, a petri net instance, is extended to a four-tuple:

$$(S, T, W, M_0)$$

$M_0$ designates the initial *marking* of places in the graph. Once the initial markings are defined, the petri net is ready for execution. The execution semantics are as follows:

- Given a marking $M$, a transition t, when firing, consumes the marking from its input places $s$ in the multiplicity defined by $W(s,t)$ and conversely generates markings in the output places s as in the multiplicity defined by $W(t,s)$, producing marking $M'$.
- If enough markings are available on the input places of a transition for it to be able to perform the consumption operation, the transition is considered *enabled*, meaning it may fire. Transitions which are not enabled cannot fire. If multiple transitions are enabled, they may fire in an arbitrary order. Since firing a transition may disable previously enabled transitions, the execution is non-deterministic.

If the arc multiplicity is always one, as some petri net approaches mandate, there has to be at least one marking on each of the input places for the transition to be enabled and ready to fire.

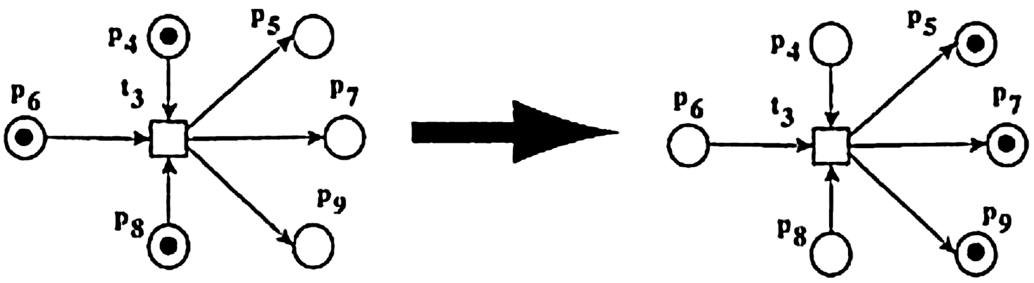


Figure 3.2: An example of a petri net transition firing: Due to places P4, P6 and P8 being marked, transition T3 may fire, after it does, the places P5, P7 and P9 are marked (from [59])

As mentioned, petri nets also have a graphical representation as shown in Figure 3.2. The places of the petri net are represented by circles, while the transitions are drawn as rectangles. The arcs of the petri net are shown as directed arrows. Places are often marked by drawing dots in the places that are marked, but other representations are possible such as a counter.

The figure shows a transition firing as an example. The transition ”t3” is enabled because its input places ”P4”, ”P6” and ”P8” are marked (contain a *token*) and the transition is therefore ready to fire. The right side represents the petri net after the transition has fired. The marking on the input places have been consumed and new markings are generated on the output places ”P5”, ”P7” and ”P9”.

Petri nets are well-researched and are fairly intuitive especially in terms of modeling concurrency, since each of the transitions are considered independently by the execution semantic. They are based on a solid mathematical framework and allow the verification of certain properties. However, basic petri nets are fairly simple and are, in fact, not Turing-complete. In order to address this, numerous variation and extensions of this basic model have been defined which maintain the verifiable properties of a basic petri net to various extents, trading verifiability for computational power and syntactic enhancement in varying degrees. The following examples include common enhancements to the basic petri nets, which are often called high order petri nets:

- The addition of inhibition arcs allows the attachment of conditions to transitions. With the addition of inhibition arcs, petri nets become Turing-complete.
- Colored petri nets allow the distinction between marks by assigning a value to mark or token on places (see [92]), which can tested using *guard expressions.*
- Prioritized petri nets assign a priority to transition. This approach imposes a well-defined execution order (see [12]).
- Nets-within-nets (see [151]) allows the inclusion of petri nets as a token in other petri nets and reference nets (see [98]). This allows the inclusion of objects for the object-oriented programming paradigm.

Reference nets in particular have been successfully used to implement not only workflow models but also a workflow management system (see [153]). As a result, certain petri nets lend themselves to be used for workflow modeling, representing a graphical workflow language. The rigorously defined semantic results in it being considered among the formal workflow languages.

However, not every type of petri net is suitable as a workflow modeling language, in particular, more basic variants force designers towards implementing elements purely for technical reasons and thus fail to concentrate on the operational and business requirements of the workflow. As a result, a workflow language based on petri nets has been developed which addresses the particular demands of workflow modeling, which will be presented in the next part of this chapter.

## 3.4 Yet Another Workflow Language (YAWL)

Since many types of petri nets already appear to support workflows to some degree and also offer both a graphical representation and rigorous semantics, many attempts have been made to use them for workflow modeling (see [152], [153] and [1]). Based on the experience gained by these attempts, a specific workflow language based on the concepts and advantages of petri nets called Yet Another Workflow Language (YAWL, see [154]) was developed.

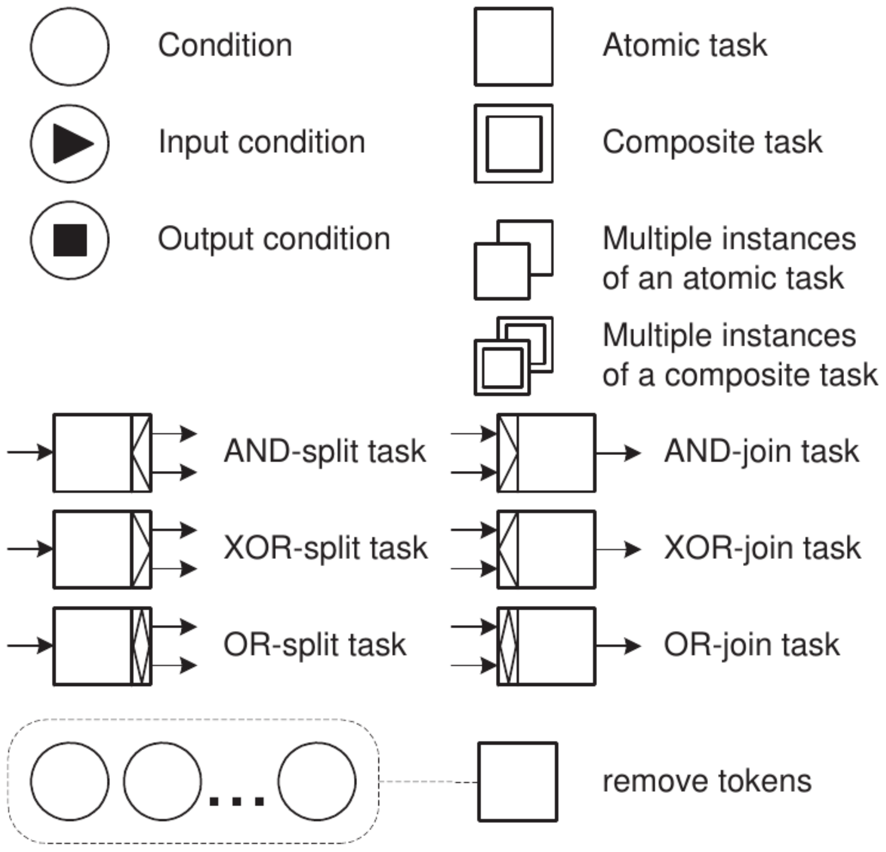


Figure 3.3: Symbols used in YAWL workflows (from [154])

Like the petri net-based approaches, YAWL is a task-based workflow language with a graphical representation. Figure 3.3 shows the symbols available in YAWL. The most basic symbol or element is the atomic task, which carries semantics similar to the tasks in other task-based business process and workflow languages, representing a piece of labor executed as a single unit without interruption.

A special case are the composite tasks, which are used as tasks in a YAWL workflow but are not atomic and are represented by another YAWL model, allowing hierarchical modeling of workflows similar to the nets-in-nets approach (see [153]).

Splits and forks in processes are accomplished using the respective split and join elements. The process flow is split using a split symbol and can later optionally be joined using the join symbol of the same type. Three types of splits/joins are available: The XOR-split is used for differentiating different cases and only one outgoing path is chosen during execution. In contrast, the and-split will cause the execution of all outgoing paths concurrently. Finally, the OR-split will result in some but not necessarily all outgoing paths.

Like petri nets, YAWL uses markers or tokens to track the execution flow. This means that a join after a split is not always necessary but may result in multiple tokens being used in the flow in cases of XOR- and OR-splits. It also allows YAWL to be used based on data flow instead of a control flow pattern, where the tasks being executed are decided based on the data available.

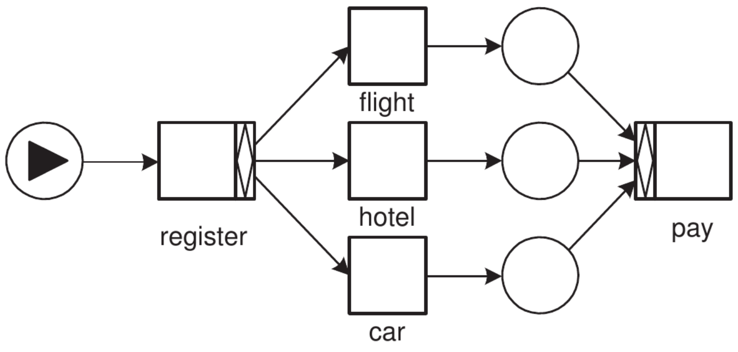


Figure 3.4: A composite task modeled in YAWL (Pattern 7 from [154])

Figure 3.4 shows an example of a YAWL model used to represent the internal structure of a composite task. It starts with an input condition defining the inputs for the composite task. It continues with an OR-split, in which one or more of the following paths are chosen depending on the runtime state.

This means that of the set of tasks following each of the paths, a certain subset which can but does not necessarily include the complelet set are executed, resulting in at least a flight, a hotel or a car being booked but allowing up to and including all three bookings to be performed. The condition following each of the tasks ensures that the pay operation is only executed once when all started tasks due to the OR-split having been executed.

An alternative pattern is based on a single condition used by all three paths, resulting in the pay operation being performed once for each path chosen by the OR-split (Pattern 8, see [154]). This may be useful if the payment is separate for each of the service providers for the flight, hotel and car and therefore need to be processed separately.

Since YAWL was derived from petri nets, it exhibits similar properties as a workflow language but are syntactically more suited for business processes and workflows. Since it is executable, it represents a formal language and it includes a graphical representation for modeling the workflows. YAWL heavily emphasizes sets of modeling patterns which are documented in detail by van der Aalst and ter Hofstede in [154], with which various common tasks can be accomplished.

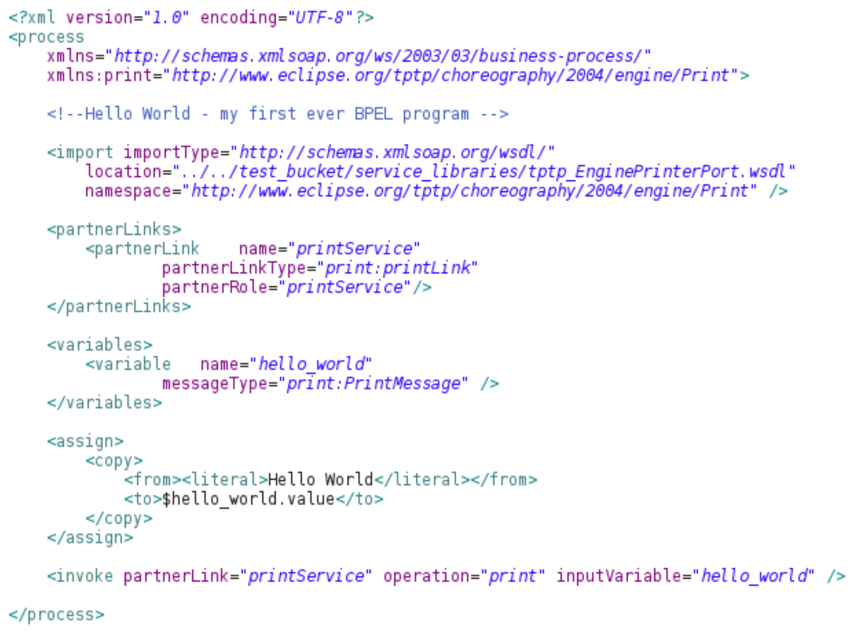
As a language, it is strongly task-centric, though it allows a degree of data flow-based modeling, for which additional patterns are also available.

## 3.5 Business Process Execution Language (BPEL)

The business process execution language (BPEL) or Web Services Business Process Execution Language (WS-BPEL) is standardized by the Organization for the Advancement of Structured Information Standards (OASIS) as part of their larger set of standards centering on web services. It is based on earlier works of IBM by Leymann with the Web Services Flow Language (WSFL, see [99]) and a competing approach called XLANG (see [147]) by Thatte working for Microsoft. Both approaches offered a language for modeling workflow based on web services defined with the Web Service Definition Language (WSDL, see [167]).

The language uses an representation based on the Extensible Markup Language (XML, see [166]) and does not offer a graphical representation. In fact, since one of the goals of the language was a unification of the approach of multiple vendors to allow the interchange of executable workflow models, a specific graphical representation was explicitly excluded as a design goal.

The key objective of BPEL is to provide a platform for workflow and therefore is meant to be executable. The tasks as modeled in other business process and workflow languages are represented by service calls onto previously defined web services. Since web services represent fairly technical interfaces, explicit extensions are available in the form of the BPEL4People (see [78]) and the WS-HumanTask (see [79]) specifications.



```
<?xml version="1.0" encoding="UTF-8"?>
<process
    xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
    xmlns:print="http://www.eclipse.org/tptp/choreography/2004/engine/Print">

    <!--Hello World - my first ever BPEL program -->

    <import importType="http://schemas.xmlsoap.org/wsdl/"
        location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
        namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />

    <partnerLinks>
        <partnerLink    name="printService"
                partnerLinkType="print:printLink"
                partnerRole="printService"/>
    </partnerLinks>

    <variables>
        <variable   name="hello_world"
                messageType="print:PrintMessage" />
    </variables>

    <assign>
        <copy>
            <from><literal>Hello World</literal></from>
            <to>$hello_world.value</to>
        </copy>
    </assign>

    <invoke partnerLink="printService" operation="print" inputVariable="hello_world" />

</process>
```

Figure 3.5: Example of a simple „Hello World" BPEL workflow (from [114])

Figure 3.5 shows an example of a simple „Hello World" example of a BPEL workflow. Web services that can be invoked as part of the workflow are defines as partner links. In this case, a single partner link defining a „printService" is included in the model of the workflow. State can be included in the workflow as variables, as shown in the „hello_world" variable in the example, including an assignment of the variable in the assign section of the model.

Web services are invoked using the invoke tag. In case of the example, the „printService" is invoked using the „hello_world" variable as input that is passed to the service. Branching, which is not shown in this example, can be accomplished using if-else-statements. In addition to these, switch-statements for defining multiple branches are available. Both types of branching uses a special condition tag to define the condition for the branching operation.

As the name implies, the language is meant to be used to defined executable workflows. It also explicitly lacks a graphical representation, which means that it is a formal and textual workflow modeling language. It is centered around its use in the context of WSDL web services, which are used as entry points for workflow tasks. While it is possible to use the language directly to design workflows, it is often used in combination with a graphical language (for example, see section 3.7) which is later translated into an equivalent BPEL workflow for execution (see e.g. [124]). As a result, BPEL often serves as a standardized execution language for workflow engines rather than a modeling language.

## 3.6 Event-driven Process Chains (EPC)

Event-drive Process Chains (EPC) is a business process and workflow language developed as a research project of SAP AG and the University of Saarland, Germany (see [95], [67] and [141]). EPCs are based on concepts derived from stochastic network as well as petri nets (see section 3.3).

While the basic EPC language has been extended into what is known as the extended EPC (eEPC) notation, the core still revolves primarily around the concept of events and functions: Events are considered to be passive, meaning they are only descriptions of state and do not perform an action within the workflow. They provide the preconditions in which a function or process works.

The functions on the other hand are the active parts of a process which details actions which should be taken and represents the "tasks" in terms of a task-based approach. The elements are connected using control flow edges, with information flow edges providing the option of modeling information sources.

Branching is accomplished with branch/merge elements and parallel forks are accomplished using fork/join elements. In addition, an OR element is available for allowing partial forks, similar to the YAWL (see section 3.4) and BPMN (see section 3.7) or-type forks.

Figure 3.6 shows a simple EPC-based business process model. The model starts with a function that checks the incoming order. This is followed by a branch element which offers two options for events in the next step, one for orders below 5000 Euro and one for orders meeting or exceeding that value. This is followed by a message element as well as a function in which the order is processed directly or a signature from a supervisor is acquired.

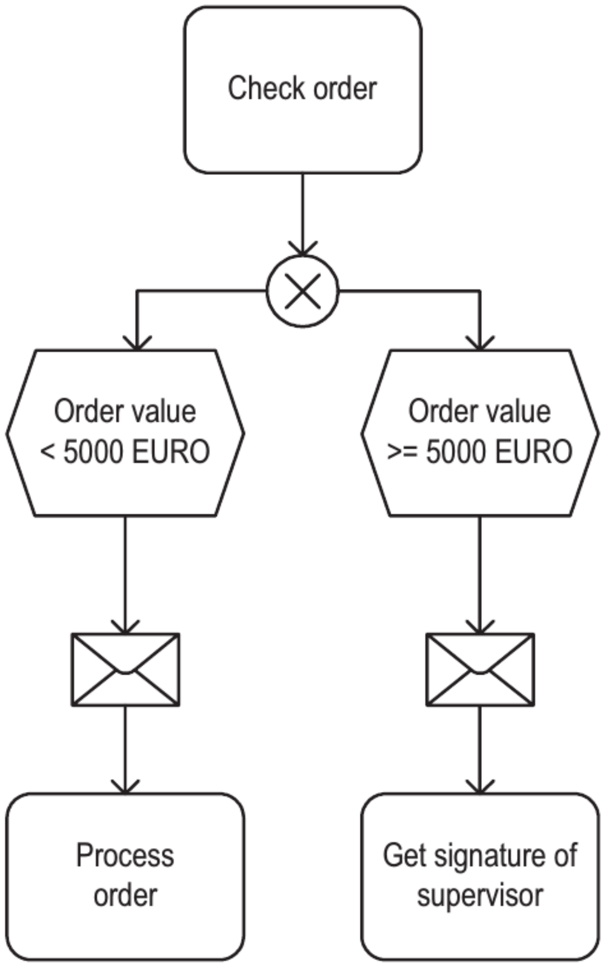


Figure 3.6: Example of a simple EPC process model (from [141])

## 3.7 Business Process Model and Notation (BPMN)

The *Business Process Model and Notation (BPMN)*, originally called Business Process Modeling Notation, is a popular example of a graphical business process and workflow modeling language. It was developed by the Business Process Management Initiative (BPMI), which was later absorbed by the Object Management Group (OMG). Since BPMN will be used as part of the solution for modeling long-running autonomous processes, it will be covered here in detail. The modeling language is part of the category of semi-formal languages, which means it aims to cover both the modeling of informal and therefore non-executable models for documentation purposes, as well as the formal specification of executable workflows.

As a result of the semi-formal specification, BPMN offers a large variety of modeling elements, some of which only have weakly or informally defined semantics. In fact, the early version lacked formal semantics. If used in the context of a workflow management system, the implementation of the system has to restrict the formal model to a subset of the available elements and attempt to interpret the informal descriptions as necessary in order to make them available as executable elements if a reasonable interpretation can be found. The traditional approach to executing BPMN processes involved the transformation of the BPMN process to a Business Process Execution Language (BPEL) [118] model, which could then be executed by a standard BPEL system.

The original version 1.0 was released in 2004, which was superseded under the auspices of OMG with version 1.1 in 2008 [120]. Changes primarily revolved about event modeling with the introduction of the concept of catching and throwing events. Version 1.2 was released the following year, but contained mostly minor editorial changes.

The current version of BPMN is version 2.0, which was released 2011 and represents a major update, with another large overhaul of event modeling but also included updates regarding choreography modeling. In addition, version 2.0 includes partial formal semantics for execution of some of the elements. These semantics are largely the result of BPMN being used in a BPEL context and are therefore mostly derived BPEL execution semantics.

In addition, choreographies can now be modeled using the collaboration part of the specification. For data exchange, a standardized XML-Format with extension points has been defined, which alleviates the need for general purpose formats like XPDL [165]. Since BPMN is also part of the workflow management solution presented later, the following will give an overview and detailed description of the most important BPMN elements.

### 3.7.1 Events

In any business process or workflow, there can be instances when an internal or external situation causes some action or state change to occur or be triggered that has relevance to the business goal of the process modeled in BPMN. These situations can be modeled in BPMN processes as events with varying semantics depending on the exact event modeled; however, following the theme of BPMN as a semi-formal modeling notation, not all event types are associated with strongly defined semantics but are open to interpretation or further specification by the user.

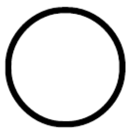
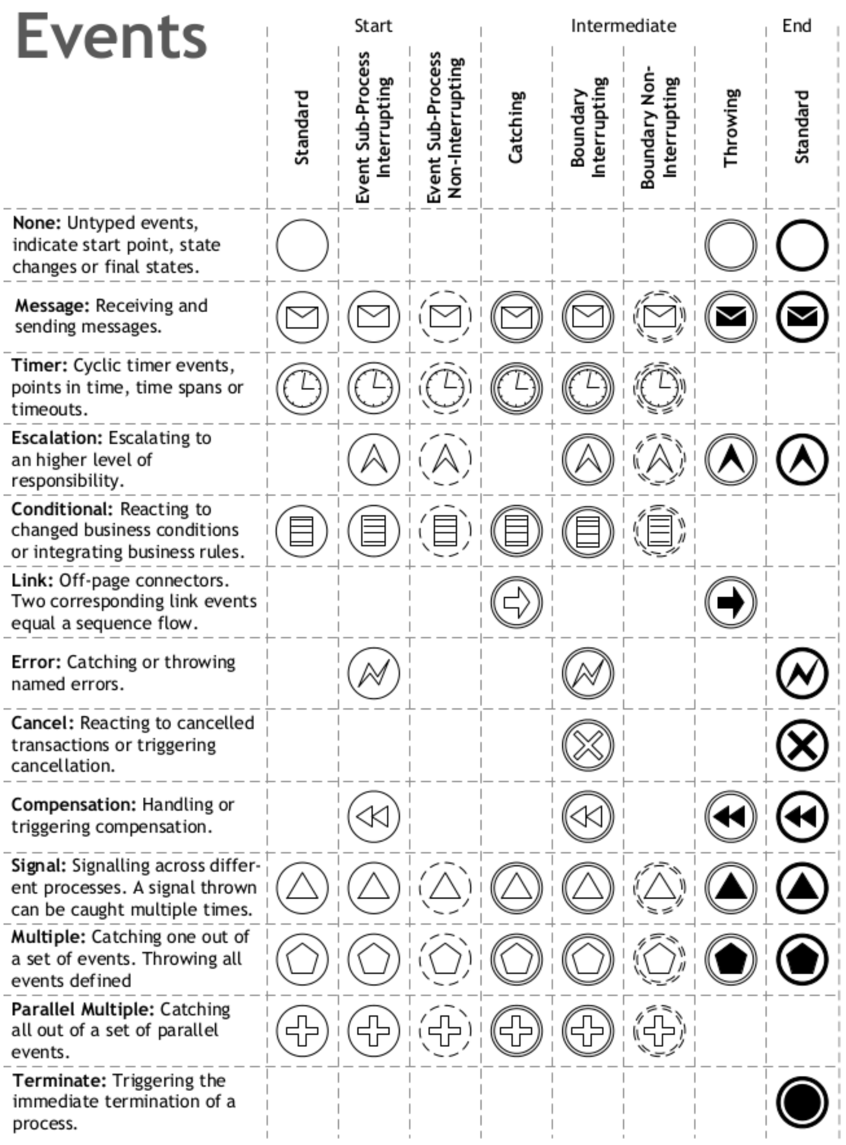


Figure 3.7: Graphical representation of an event shown in [121], the basic form shown here represents a standard (empty) start event

Events are represented graphically in BPMN as circles as shown in 3.7, with varying border options and a number of pre-defined contained icons to represent different types of events. For example, an event with a thin border denotes a *start event*. This means that the event models an occurrence that initiates the BPMN process, such as a purchase order by a customer.



| Events | Start | | | Intermediate | | | | End |
|---|---|---|---|---|---|---|---|---|
| | Standard | Event Sub-Process Interrupting | Event Sub-Process Non-Interrupting | Catching | Boundary Interrupting | Boundary Non-Interrupting | Throwing | Standard |
| **None:** Untyped events, indicate start point, state changes or final states. | ✓ | | | | | | ✓ | ✓ |
| **Message:** Receiving and sending messages. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Timer:** Cyclic timer events, points in time, time spans or timeouts. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| **Escalation:** Escalating to an higher level of responsibility. | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| **Conditional:** Reacting to changed business conditions or integrating business rules. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| **Link:** Off-page connectors. Two corresponding link events equal a sequence flow. | | | | ✓ | | | ✓ | |
| **Error:** Catching or throwing named errors. | | ✓ | | | ✓ | | | ✓ |
| **Cancel:** Reacting to cancelled transactions or triggering cancellation. | | | | | ✓ | | | ✓ |
| **Compensation:** Handling or triggering compensation. | | ✓ | | | ✓ | | ✓ | ✓ |
| **Signal:** Signalling across different processes. A signal thrown can be caught multiple times. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Multiple:** Catching one out of a set of events. Throwing all events defined | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| **Parallel Multiple:** Catching all out of a set of parallel events. | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| **Terminate:** Triggering the immediate termination of a process. | | | | | | | | ✓ |

Figure 3.8: Table of event types in BPMN 2.0 as shown in [11]

Events with a border consisting of two thin lines represent *intermediate events*. These event types represent events which can occur or be triggered at any point while the process is running. Finally, processes that end the control flow of a process are *end events* and are represented as circles with a thick border.

BPMN also distinguishes between *catching* and *throwing* events, which is used to denote event sources and drains for events. If an event occurs outside of, or somewhere else within the same process, and the process is supposed to react to this event, it is modeled as a catching event (the event that occurred is "caught" by the modeled event element, similar to catching exceptions in object-oriented programming languages such as Java [62]). Conversely, throwing events actually trigger the event itself, such as when a timeout occurs or a message is sent.

As a corollary, not all events can be both catching and throwing. For example, since start events start a new control flow, they cannot be throwing since throwing an event by itself does not impose a condition for the start of a control flow and throwing an event immediately after the start of a control flow can be easily modeled by a throwing intermediate event following the start event.

This becomes even more clear in the case of end events. Since end events terminate the control flow, catching an event at that stage prevents any option of actually processing the event. As a result, end events are always throwing events.

Events can also contain icons to further specify the type of event, whereas an event without an icon is considered to be an unspecified empty event. Event type icons include *message events*, which denote the sending or reception of a message by the process. Another common type is the *timer event* for chronologically-related events such as delays, timeouts or scheduled triggers. *Escalation events* are a type of business exception handling when a process needs to escalate a business decision to a high level, such as when a management decision is required after a failure of the common case handling in the process. *Conditional events* denote a change in business conditions and allow the process to react to such a situation. *Link events* tend to be more technical by allowing sequence crosslinks between event elements of that type.

Errors in the process can be handled by *error event* elements, while cancelations of a business action is modeled using a *cancel event*. In case of transactions, the compensation event denotes the trigger and start of a compensation action. If a system-wide signal should be issued or caught by multiple processes, the *signal event* type can be used. *Multiple* and *parallel multiple events* can combine multiple separate events. Finally, the *terminate event* can trigger the immediate termination of a process.

Not every combination of these event types are meaningful with both catching and throwing semantics or are compatible with start, intermediate and end events. Fig. 3.8 gives an overview of the available event type combinations in BPMN 2.0, as well as a brief explanation of their meaning.

### 3.7.2 Activities

Business processes and workflows organize the labor necessary to meet the goals of the process. As such, they need elements in the modeling to represent work being performed by some entity, be it a worker, department or organization. Fragments of work performed in a process are modeled using activities in BPMN.

Activities are graphically modeled as rectangles with rounded corners, as shown in Fig. 3.9. From the perspective of the process, tasks can be both atomic and non-atomic (*compound*). The most common form of activities is the *task*. Tasks represent simple business activities that can be performed manually by a human actor such as entering data into a form or they can represent automated processes such as an action by machinery or call of a remote service. Simple tasks only consist of the rounded rectangle with an additional label describing the task, however, in BPMN

Figure 3.9: Graphical form of a BPMN activity as defined in [121]

2.0 more specialized tasks are available such as user tasks which specifically denote work being carried out by a user of a workflow system. Such specialized tasks show additional icons in the top left corner to designate their specialization.

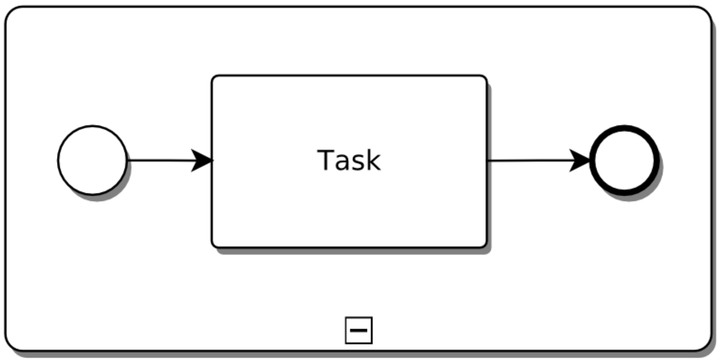


Figure 3.10: An uncollapsed subprocess, showing its internal structure with a start event, a task and an end event

The other type of activity is the *subprocess*. As the name implies, subprocesses are themselves processes containing further BPMN elements as part of the subprocess which are executed as a processes once the activity triggers. A number of special subprocesses are available such as *transactional subprocesses* or *event subprocesses* for even processing. Subprocesses can be either uncollapsed, as shown in Fig. 3.10, or collapsed. In the uncollapsed state, the subelements within the subprocess are displayed, along with a minus icon on the bottom center of the subprocess. This icon lets the user switch a subprocess to its collapsed state, in which the elements contained in a subprocess are hidden and the icon changes to a plus sign.

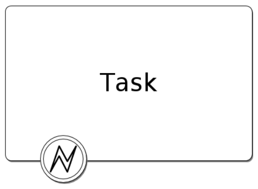


Figure 3.11: A task activity with attached error handler

All activities can be combined with special event elements called *event handlers*, which are attached to the activity as shown in Fig. 3.11. These handlers are triggered if a specified event occurs within the activity and allows handling of such an event. As a result, event handlers are always event elements of the catching type. Typical examples are error event handlers, which allow the interception and handling of errors which may occur while executing a task or subprocess, timer events which trigger a timeout if the task execution exceeds a specified time limit or cancel event handlers in case a particular activity is canceled for business reasons.

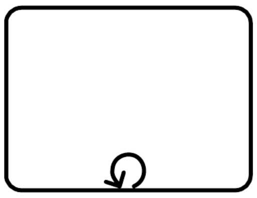


Figure 3.12: A task activity with a looping modifier as shown in [121]

Activities are also able to perform loops, where a task or subprocess are executed multiple times iteratively. Figure 3.12 shows such a looping task, which is marked graphically with a looping arrow icon. Looping activities are defined by two parameters: First, a condition which will end the loop once it evaluates true. Second, a maximum loop count which will also terminate the loop once the loop has executed the same number of iterations.

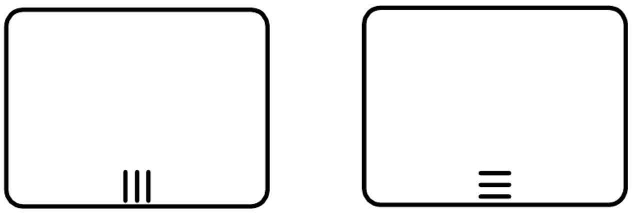


Figure 3.13: Task activities with parallel (left) and sequential (right) modifiers (from [121])

Looping activities are essentially a shorthand for a common pattern consisting of the activity, an exclusive gateway (see 3.7.4) and a sequence flow (see 3.7.3) looping back on the task.

Additional modifiers for activities include *parallel* and *sequential activities.* Parallel activities are graphically represented with an icon consisting of three vertical lines, while sequential activities display an icon of three horizontal lines as shown in Fig. 3.13. Sequential and parallel activities are data-driven, which means they receive a list of data items as input. For each item in the list an *activity instance* is generated to process the item. As the name implies, sequential activities process items sequentially in the order they are contained in the list, while parallel activities process all items in parallel.

As with the looping activities, sequential activities are syntactic sugar which can be replaced by combining gateways and sequence flows. They could either be represented by an appropriately configured looping task or a combination of exclusive gateways and conditional sequence flows.

Parallel activities are not easily replicated with standard BPMN modeling elements. While parallelism is supported through the use of parallel gateways (see 3.7.4), the number of items in the list is a runtime (instance) parameter which is unknown at model time. As a result, the number of outgoing sequence flows cannot be predicted in advance, thus impeding the parallel processing of item lists and necessitating the use of parallel activities.

### 3.7.3 Sequence Flows

Like any activity-based business process modeling language, BPMN requires a modeling element to specify the execution order of an execution thread of the process. This function is fulfilled with the *sequence flow* element.

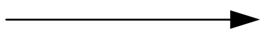


Figure 3.14: Graphical form of a BPMN sequence flow as defined in [121]

The sequence flow element is graphically represented as an arrow (see Fig. 3.14) and denotes possible transition from one activity to another within the BPMN process in execution.

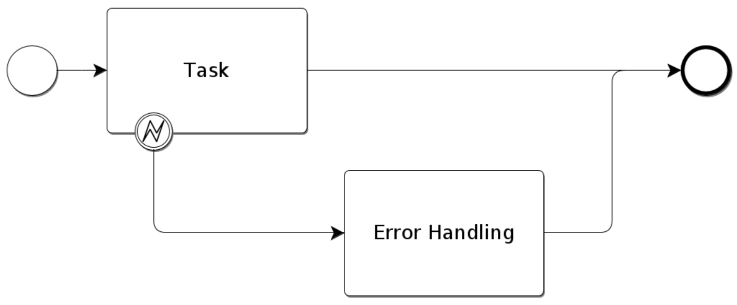


Figure 3.15: Example for the use of sequence flow within a process [121]

Labels with a description of a particular sequence flow can be attached, illustrating the business meaning of the transition within the context of a process. As demonstrated in Fig. 3.15, sequence flows can be used between any activities or events, including event handlers. In this example, the process starts from an empty start event, and attempts to perform a task. If the task is successfully performed, the control flow proceeds to the empty end event and the process terminates. If the task fails, an error handler is triggered and the sequence flow directs control to the error handling task. After that task has been executed, the control passes along the sequence flow to the end event and the process is over.

### 3.7.4 Gateways

Since most business processes are not simple linear work lists, BPMN introduces *gateway* elements to support forking, branching and merging behavior for BPMN processes.

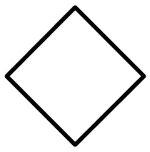
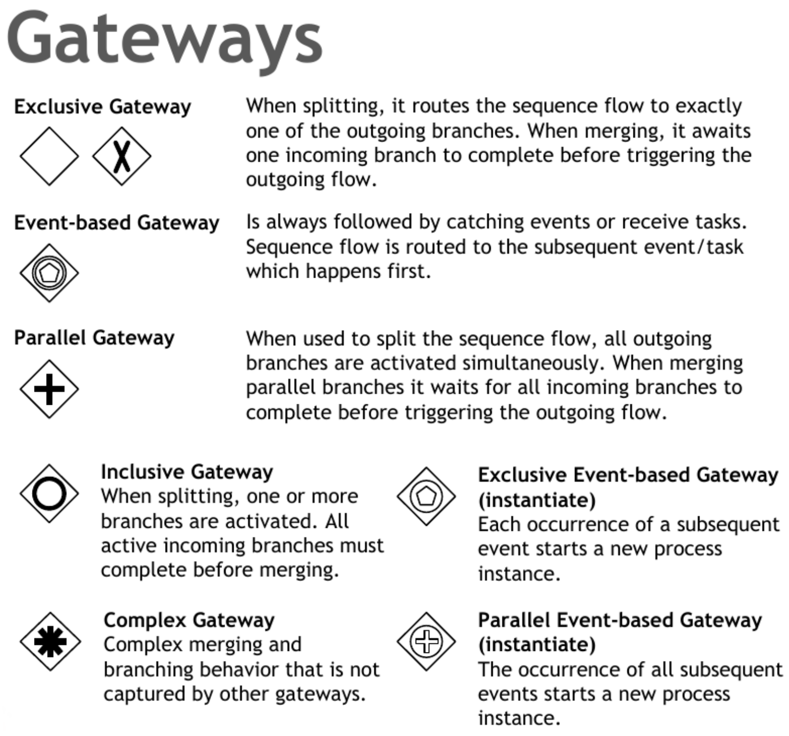


Figure 3.16: Basic BPMN gateway shape (from [121])

Gateways are represented graphically as diamond shapes as shown in Fig. 3.16. Generally, gateways either have a single incoming sequence flow and multiple outgoing sequence flows (branching or forking, depending on the semantics of the specific gateway) or the gateway has multiple incoming sequence flows and a single outgoing sequence flows (merging).



Gateways

**Exclusive Gateway**
When splitting, it routes the sequence flow to exactly one of the outgoing branches. When merging, it awaits one incoming branch to complete before triggering the outgoing flow.

**Event-based Gateway**
Is always followed by catching events or receive tasks. Sequence flow is routed to the subsequent event/task which happens first.

**Parallel Gateway**
When used to split the sequence flow, all outgoing branches are activated simultaneously. When merging parallel branches it waits for all incoming branches to complete before triggering the outgoing flow.

**Inclusive Gateway**
When splitting, one or more branches are activated. All active incoming branches must complete before merging.

**Exclusive Event-based Gateway (instantiate)**
Each occurrence of a subsequent event starts a new process instance.

**Complex Gateway**
Complex merging and branching behavior that is not captured by other gateways.

**Parallel Event-based Gateway (instantiate)**
The occurrence of all subsequent events starts a new process instance.

Figure 3.17: Table of gateway types in BPMN 2.0 from [11]

A variety of gateways with different semantics are available in BPMN as shown in Fig. 3.17. The semantics specify whether a process control reaching a gateway continues as multiple parallel threads along each of the outgoing sequence flows (forking) or if the control takes only a single specific path based on annotated conditions on the outgoing sequence flows (branching) or a combination of both behaviors where some but not all outgoing sequence flows are followed in parallel. While the merging behavior is different for each type, it is merely a mirror-image of the branching and forking behavior: The merging gateway merely acts as a barrier, waiting for all forked threads to arrive at the merging gateway before continuing with the outgoing sequence flow. The difference between gateway types are therefore based on theirF branching and forking behavior.

However, as BPMN is a semi-formal language, not all gateway types have strongly defined semantics associated with them. For example, the *complex gateway* only specifies that a complex branching/forking behavior is supposed to be performed at that gateway but does not clarify how this behavior is specified. The most common gateway types with well-defined semantics are the *exclusive gateway*, the *parallel gateway* and the *inclusive gateway*. The exclusive gateway is a strict branching gateway. An incoming thread will continue on exactly one outgoing sequence flow. The concrete sequence flow the thread will continue on is decided based on conditions annotated on the outgoing sequence edges. In addition, a *default sequence flow* can be defined which is followed in case none of the conditions on the other outgoing sequence edges evaluates as true.

The parallel gateway on the other hand is a strict forking gateway. A thread executing a parallel gateway will fork into multiple parallel threads for each of the outgoing sequence flows. The execution will continue in parallel from that point on until the threads are again merged at a merging parallel gateway. The inclusive gateway is a combination of the behavior of the exclusive and parallel gateway. First, the conditions on the outgoing sequence flows are evaluated and the thread is forked for each of the sequence flows where the condition evaluates as true.

### 3.7.5 Message Flows

BPMN processes can define messages being exchanged between different parts of the process. Since the sending and receiving of messages constitutes an event, the message event element (see 3.7.1) has to be used to specify such occurrences.

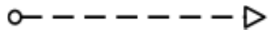


Figure 3.18: Graphical form of a BPMN message flow as defined in [121]

The connection between two such message events can be modeled using *message flows*. Message flows are represented as dashed arrows as shown in Fig. 3.18. Since a message must have both a sender and a receiver, this means that the source of a message flow must be a throwing message event and the target, at which the arrow points, must be a catching message event.

While it is possible to model sending a message by one thread which is later received by the same thread, a more typical case would involve an interchange between two parts of a process being executed in parallel or interchanges between a process choreography being contained and represented by the same BPMN model.

### 3.7.6 Pools and Lanes

In BPMN, pools define the *participant*, particularly in a collaboration context. The entity representing a participant in a collaboration process is process dependent;

examples could include departments of the same organization or even different organizations in cross-organizational collaborative processes.

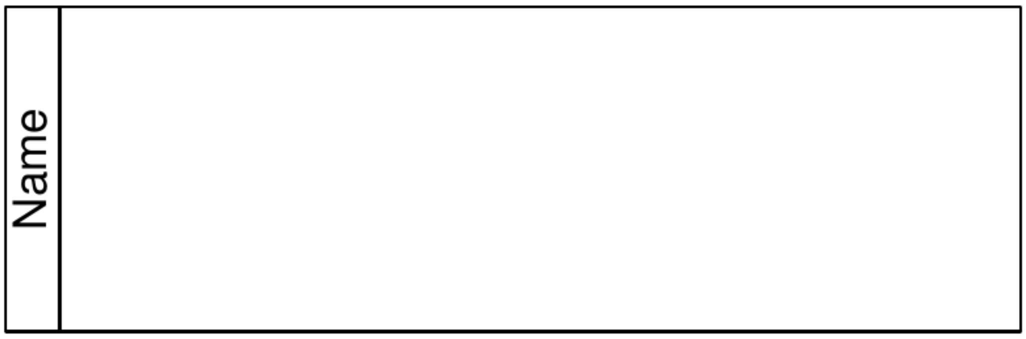


Figure 3.19: Pool element as defined in BPMN (from [121])

Pools, as shown in Fig. 3.19, are optional in terms of processes and are only needed when defining choreographies. However, they are sometimes useful as a visual aid to group the elements of a process. Since pools define major participants in a collaborative process, it is not possible to connect direct sequence flows between elements of different pools. However, message flows can be defined to model information exchange between the participants.

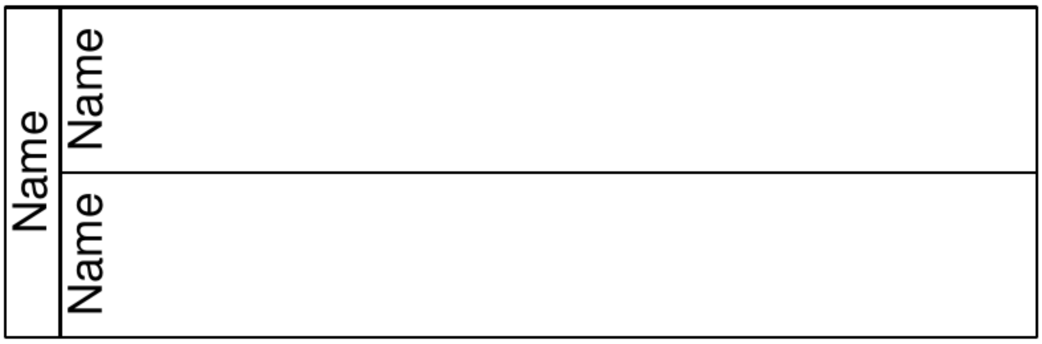


Figure 3.20: Lane elements as defined in BPMN (from [121])

*Lanes* or *swimlanes*, as shown in Fig. 3.20, are used to further subdivide the process. Like the pool containing them, they extend the full length of the process. They are used to organize and categorize activities within the process. For example, they can specify service centers or groups of users in charge of a part of the process. Unlike pools, sequence flows between different lanes of the same pool are allowed. In addition, lanes can be further partitioned with additional nested lanes.

### 3.7.7 Text Annotations

Sometimes, for example in large and complex processes or processes with unusual patterns or element usage, it may become necessary to explain aspects of the process to a human reader of the model. This feature is especially important for informal (non-executable) process models which are only read by a human reader who may need additional explanation to understand the process. In BPMN processes, attaching such additional information can be accomplished using text annotations.

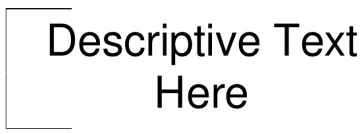


Figure 3.21: BPMN processes can be annotated using text annotation as shown here (from [121])

Text annotations are graphically represented by an open rectangle. The annotation text is attached to the right next to the open part of the rectangle as shown in Fig. 3.21. Since the text has an informal role, there are no restrictions on the text itself.

Text annotation is part of a number of BPMN elements called *artifacts*. Artifacts are not active BPMN elements with specific semantics, rather they enhance such elements or the overall process with additional information.

### 3.7.8 Associations

Information and artifacts such as text annotations (see 3.7.7) can be associated with BPMN elements using association elements.

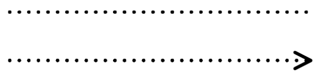


Figure 3.22: As described in [121], information and artifacts like text annotations can be associated with BPMN elements using associations

Association elements are graphically represented in BPMN using dotted lines (see Fig. 3.22). An optional arrow can give the association directionality if the attached artifact requires it. While text annotations do not require directional associations, some artifacts such as data objects (see 3.7.9.1) benefit from specifying whether the object is consumed or produced by the associated BPMN element.

### 3.7.9 Additional BPMN Elements

In addition to the BPMN elements and artifacts mentioned in the previous sections, the BPMN standard also defines additional elements and artifacts that can be used both for informal processes and processes which are converted to workflows in the Business Process Execution Language (BPEL). Since the system presented in this work follows a different approach for processing BPMN, some of the elements were not strictly required. Furthermore, some elements were not included since there was no direct necessity for the in for the specific types of processes that are intended to use the proposed system. Nevertheless, in order to present a complete picture, these additional elements are presented with a brief description in the following parts.

#### 3.7.9.1 Data Objects

Data objects represent business data being exchanged in the context of a BPMN process. In standard BPMN 2.0, they are used to specify data that activities such as tasks or subprocess can either consume or produce while they are executed. Usually, a data object represents some sort of business data; for example, a form that a customer is supposed to submit which is later passed to another task performed by an employee who processes the form.

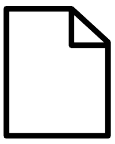


Figure 3.23: Graphical representation of a data object in a BPMN process as defined in [121]

Data objects can represent both singular objects or they can represent lists or collections of data. Graphically, a data object is represented in the model as an icon showing a folded piece of paper (see Fig. 3.23). They are attached to their producing and consuming activities using directional BPMN associations (see 3.7.8). In addition, data objects can be attach using non-directional associations to convey the notion that the data object is used by the associated element in some loose fashion, for example by representing a file on the file system which is accessed by an activity.

#### 3.7.9.2 Messages

Standard BPMN also includes an artifact type for representing a message being exchanged within the process. This artifact type can be used to either model messages that are externally associated with the process or messages that are part of a more formal exchanged modeled with message events (see 3.7.1) and message flows (see 3.7.5).

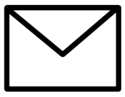


Figure 3.24: A BPMN message as defined in [121], the artifact being represented as a letter-shaped icon

Figure 3.24 shows the graphical representation of a message. Messages are represented in BPMN with an icon displaying a letter. Messages have particular relevance in the context of BPMN collaboration diagrams.

In summation, workflow models designed in BPMN can be executable but some elements of BPMN lack well-defined semantics and contain ambiguities. As a result, valid models in BPMN can also represent non-executable business process descriptions. Therefore, BPMN falls into the category of a semi-formal language, allowing its use in an IT-driven workflow environment as well as a visualization and discussion tool for informal settings. Since it includes a prominent graphical representation, it is also a graphical business process and workflow language.

The next section will assess the task-based business process and workflow modeling approach regarding their support for long-running autonomous business processes and evaluate them with regard to the three research goals, strategic-operation cohesion, workflow model agility and balance of global control and local autonomy.

## 3.8 Limitations of Task-based Business Process and Workflow Modeling Languages

Section 1.2 gives an introduction to long-running autonomous processes while also outlining research questions and goals to improve support for such business processes. Modeling languages are a key aspect in the design and implementation phases of the BPM lifecycle described in that section. For the first two phases of the lifecycle, three research goals are relevant when considering a language for long-running autonomous processes.

The first research goal is strategic-operational cohesion, which aims to strengthen the link between the operational aspects of a business process or workflow and the strategic business goals an organization designates for the process.

Task-based modeling languages concentrate strongly on the operational aspects of the business process: Tasks are defined and associated with a specific control flow which enforces an order on the tasks being executed, while allowing for a limited number of variants based on the modeled branches. This approach works very well for production business processes since the strategic goals are fairly clear even at the operational level (usually involving the production of goods). It also resembles how many naïve approaches to describing a production process are accomplished with a step-by-step instruction on how to assemble an item. However, when it comes to long-running autonomous processes, this task-based, strongly operational approach has some drawbacks, especially regarding workflow model agility and the autonomy of the workflow participants. This is demonstrated with the following example.

Figure 3.25 shows a simplified business process modeled in BPMN (as an example of a task-based language) for designing a new airplane model. This type of business process is a development process and therefore a typical example of a long-running autonomous business process involving autonomous experts from various departments and involving an execution time measured in years.

The process describes how three parts of an aircraft are developed, the fuselage, the wings and the engine. The first task of this process is the step of developing an initial prototype design for the fuselage. This is followed by a design review of the

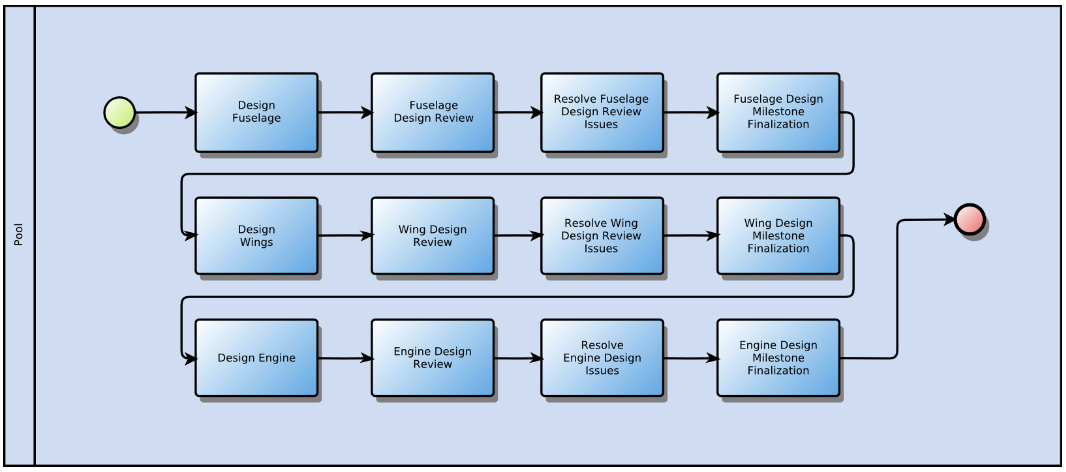

Pool
Design Fuselage
Fuselage Design Review
Resolve Fuselage Design Review Issues
Fuselage Design Milestone Finalization
Design Wings
Wing Design Review
Resolve Wing Design Review Issues
Wing Design Milestone Finalization
Design Engine
Engine Design Review
Resolve Engine Design Issues
Engine Design Milestone Finalization

Figure 3.25: Example workflow for designing an airplane

fuselage in which the participants discuss the prototype design and attempt to find issues which need to be addressed before the fuselage design is finalized. The third task involves the prototype designers resolving the issues raised during the design review by adapting their prototype design. In the fourth task, the modified design is finalized and committed to a formal document for later use during production.

The same basic steps are repeated for the wings and the engine. They likewise require an initial design phase, after which the prototype design undergoes a design review, where design issues are addressed and the prototype is finalized.

However, while it may be desirable for management to partition a development process in such a way so as to complete the specification for each part within a certain time frame, the three parts are ultimately interlinked and influence each other; in other words, the engine is incorperated into the wing which is attached to the fuselage. As a result, a design change in one part may result in requiring additional changes to the other two parts. For example, a design change on the engine may result in the engine being heavier, which then results in the need to strengthen the bolts used to attach the wing to the fuselage, which requires both of them to be changed to admit the heavier bolts.

However, if an already-finalized design is changed, it is required to undergo a further design review in order to ensure that the alteration does not negatively impact other design elements. As a result, during the design of one part it must become possible to revisit the design process of an already finalized part in order to accomodate the changed design.

Furthermore, as mentioned in section 1.2, the participants in such a development process often require a degree of autonomy. This means that the decision to start developing one part over another is often best left to the expertise of the workflow participants. This means that the workflow participants must be able to design the parts in any order while also allowing revisits as needed. Although seemingly simple requirements, modeling such a process flow in typical task-based business process languages becomes surprisingly difficult.

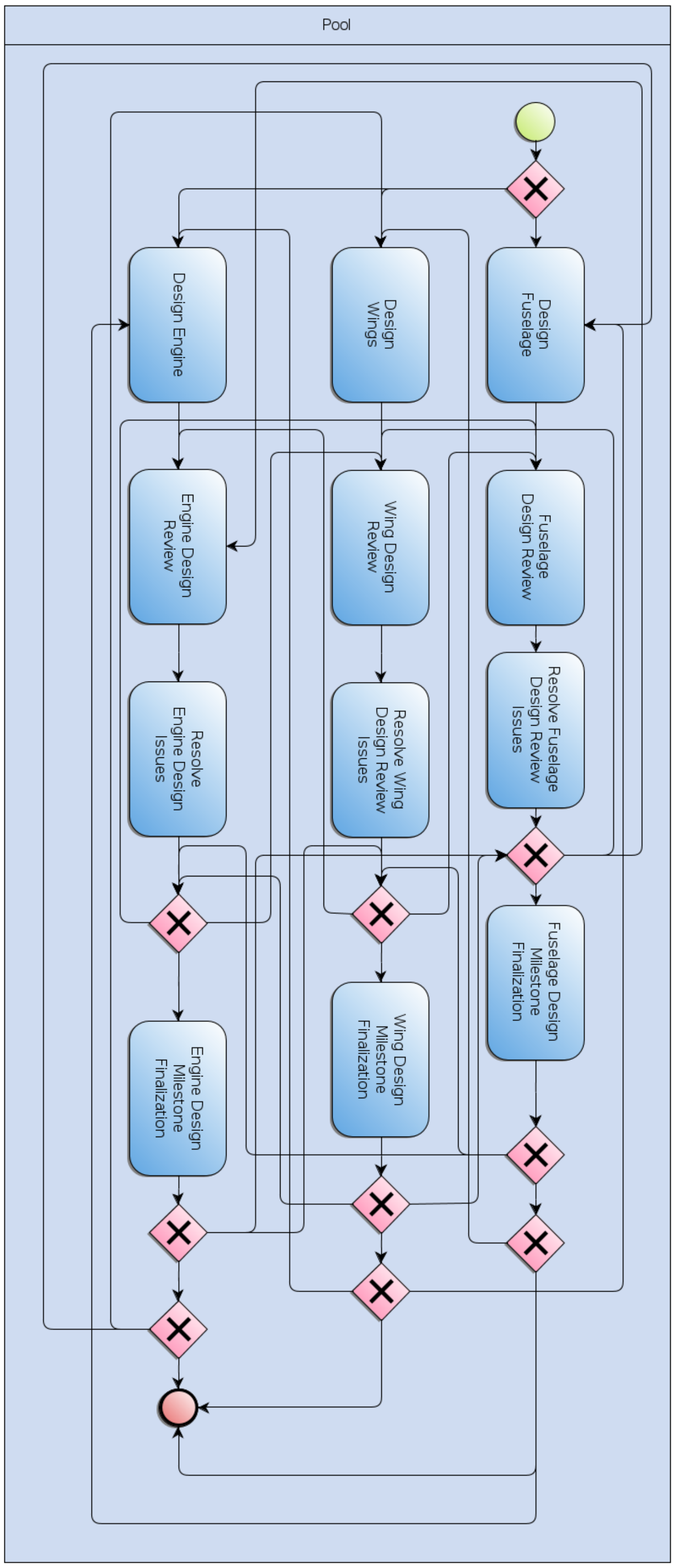

Pool
Design Fuselage
Design Wings
Design Engine
Fuselage Design Review
Wing Design Review
Engine Design Review
Resolve Fuselage Design Review Issues
Resolve Wing Design Review Issues
Resolve Engine Design Issues
Fuselage Design Milestone Finalization
Wing Design Milestone Finalization
Engine Design Milestone Finalization

Figure 3.26: Modification of the previous example process including design revisits after a milestone has passed

This difficulty is demonstrated in Figure 3.26, which attempts to modify the process previously shown in Figure 3.26, enhancing it with the necessary branches to allow choice of design order and revisits of design reviews. The first possibility to branch in the process begins right after the process has started, where the workflow participants have to decide which part is to be designed first.

The next XOR gateway is inserted to allow design revisits after the task resolving the issues and can allow either the continuation of the process if no revisit is necessary or accomodate a revisit of the review part of the two other design elements, which in this case would be either the wings or the engine.

After the finalization of the fuselage design, the process is met with another decision based on an XOR gateway. Here, the process can return to the gateway before the finalization step of the other two parts in case the finalization of the fuselage was a revisit, otherwise, the process continues as normal.

This gateway is immediately followed by another XOR gateway, which further enables the arbitrary and participant-driven design order. The participants of the process can decide here if they want to continue the process by designing the wings or the engine.

This pattern needs to be repeated for all three parts in order to allow the full flexibility required by the autonomous workflow participants. In summary, each part needs to be provided with the following three gateways and associated links in addition to the initial starting decision about which part to design first:

- After addressing the design issues but before the part finalization, each part of the airplane requires a gateway with outgoing sequence edges going to the design review stages of all other parts and incoming sequence edges from the gateway after the finalization of the parts as well as a sequence edge that continues to the finalization in order to accomodate revisits of design reviews.

- Another XOR gateway has to be provided after the finalization step of the part going to the previously mentioned gateway before the finalization step of all other parts of the airplane and an option for continuing the process. This allows for a return from design review visits.

- The design order is enabled by the last XOR gateway for each node that allows the start of the design process of all other parts. The outgoing sequence edges point towards the initial design stage for the other parts and have to be carefully guarded so they cannot be followed if an initial design already exists.

Despite its complexity, this example is a strongly simplified design process. Many development processes includes thousands of parts. In addition, each sequence of design steps for parts adds to the complexity of the design steps of other parts due to all the crosslinks and return branches between the design steps. Additionally, the process has to keep track of the statuf of what design revisits are necessary and where the process must return after a revisit in a manner that allows the correct decision at each branching point. This results in a task of data management, which is not shown in the example figures, and increasingly leaves the area of business data and is starting to become data kept not for business but technical reasons.

Furthermore, despite the enormous and quickly growing complexity of the branching mechanism for revisits, additional complexity incorporated into the design order which is necessary for sufficient workflow flexibility for long-running autonomous processes. The example as modeled in Figure 3.26 allows for revisiting the design review at multiple stages and allows for any design order of parts for the participants to follow during the process. For example, the participants can decide to start with the wing design if they consider this to be necessary and continue with the fuselage instead of the fixed order shown in Figure 3.25. As shown, if such an increased amount of local autonomy is necessary for a development process, the number of branches increases even further.

In conclusion, while task-based approaches to business process and workflow modeling are very intuitive for typical production workflows, they only maintain a medium connection to the strategic goals of the organization. Additionally, modeling the required local autonomy and workflow flexibility in task-based modeling languages is fairly difficult, complex and prone to error due to the quickly increasing complexity of branching elements within the model. As a result, a different approach is necessary to improve the support for long-running autonomous business processes and workflows.

## 3.9 Rule-based Workflow Modeling

Aside from task-based language approaches to business process and workflow modeling, a number of alternative modeling solutions have been used to model particular sets of business processes. In such systems, the workflow is defined by a set of rules which can be triggered by changes to a workflow state, which then causes a predefined procedure to be invoked. This procedure then performs the updates to the state or causes external service invocations.

A number of rule systems which can be used to model workflows are available, for example one such rule system commonly found and used in database system is the Event-Condition-Action (ECA) approach described in [112]. In this approach a defined event specifies when a rule should be triggered. This is followed by a condition which is then evaluated. If the condition evaluates to true, a predetermined action is invoked to perform the necessary steps. In database systems, these are usually updates of data, however, for a workflow system this can easily be extended to perform external actions.

ECA rule sets can be executed on rule engines using algorithms such as the Rete algorithm (see [47] and [46]), which would also provide a convenient execution platform for workflows modeled using this approach.

Since rule-based systems do not have a fixed control flow order and can perform any action associated with a rule at any point when the right conditions are met, approaches based on rules would help with the problem of workflow model agility.

For example, if part of a workflow needs to be revisited at a certain point, as shown in section 3.8, an action containing the revisit steps could be created and associated with an appropriate rule which is invoked when a revisit is required.

As a result, a number of attempts have been made to use rule systems for workflow modeling in some form, for example by directly modeling them as ECA rule sets as demonstrated in [60]. Another approach called DynaFlow not only uses rules to create a dynamic workflow model but also includes the use of rules as part of the workflow management system outside the actual workflow instances (see [113]). Alternatively instead of modeling the workflow itself using rules, another approach uses a rules system to modify an existing workflow if the rules in the systems are triggered (see [116]).

Finally, even task-based approaches often allow the use of rules to enhance the workflow in some way. For example, a database supporting an ECA rule system can be employed as part of the workflow, although, this would limit their application to data modifications. However, some task-based modeling languages also explicitly contain a rule modeling element to include rules in the process. While such an element by itself does not prevent the excessive complexity demonstrated in Figure 3.26, one could instead decompose the process into multiple independent processes capable of being triggered by a rule event, effectively replacing the action part of the rule with a workflow.

However, while rule-based approaches offer better support in terms of workflow model agility over task-based modeling languages, they have significant drawbacks in terms of the research goal of strategic-operation cohesion: The rules used in such workflow systems are very small parts of entire workflow models and are not associated in any way with the strategic planning of the organization. This severely impedes the possibility of justifying the actions taken by the system. For example, if a design review revisit is initiated by a rule that was triggered within the system, it is not easy to tell the strategic reason why that particular action was invoked. In fact, the strategic context given by a rule-based workflow system is even weaker than using a task-based workflow modeling language where it is sometimes possible to infer the strategic planning context of an action by its position within the control flow.

This criticism of a rule-based approach is part of a more general issue with rule-based programming approaches for development (see [101]): In this paper, Li identifies three major issues with rule-based programming approaches that limit their application in practice. The first issue is the maintainability of the resulting product, in the case of workflow systems, the workflow. Maintainability is often touted as an advantage of rule-based systems, however, in practice this cost potential is almost never realized because changing a rule system requires a knowledge engineer, often even from the group of experts who developed the system. Additionally, rule systems often do not allow for sufficient control over non-trivial applications and therefore such systems often only have a declarative form but not declarative content by including met rules that establish the necessary control.

The second issue is the testability of such a system. In a conventional program, certain language structures and conventions are available which make it easy to follow the control flow of an executing program for test purposes. While it is possible to write a difficult-to-test program, if good standards are followed during development, the subsequent control flow can be followed readily. In contrast, standards which establish such clear control flows are not well-supported when using rule-based approaches. Structures that are relatively simple to implement in a conventional language such as do-loops are very awkward to implement using a rule-based system.

Reliability of rule-based approaches is the third area identified by Li as an impediment. Most real-world application require high or at least sufficient reliability. While the correctness of many applications cannot be proven, Li notes that proving even the absence of errors in part using testing cannot be established when dealing with applications developed based on rules. This is due to the fact the given a set of rules and data, many rule systems do not always give a clear answer regarding the order of possible rule invocations and therefore lead to non-deterministic behavior of the implemented application. This non-deterministic behavior results in performed tests being unreliable predictors of real behavior since there is no way of knowing whether the performed test will give the same result each time it is executed.

Finally, from a business process modeling aspect there is another issue with rule-based approaches that ties in with the points made by Li regarding maintainability. As noted by Li, the development and maintenance of such a system must be performed by a knowledge engineer or similar technically versed person with extensive knowledge about the particular rule system used.

However, in business process management, experts from multiple domains are often part of the design process for a business process or workflow model. This not only includes technical persons without particular knowledge in rules systems but also domain experts from outside the technical sphere such as business or management experts. Moreso, even the participants of the business process being designed are part of this design process and are often deeply involved in the details of the business process model.

In contrast, rules in such systems are highly technical and require a specialist to understand both their purpose and their runtime effects. The resulting set of rules are completely inaccessible for persons with a non-technical background and prevents them from understanding the business process even though for many people from a non-technical background, such as management and strategic planners, it is vitally important to gain an insight in a business process.

This undermines one of the key goals of business process management, namely making the business process known throughout the organization, which is also the reason why many business process and workflow modeling languages offer a graphical representation for designing business process models. While a set of rules may eventually result in the intended behavior of the workflow, given enough domain experts and time, management and the business process participants are left unaware of the process and cannot contribute to further improvements.

## 3.10 Workflow Instance Agility with Adept2

The necessity for additional flexibility in workflow management resulted in another approach called Adept2 (see [136]) which allows the implemented model behavior of a workflow instance to change to accomodate an altered business situation. This approach is based on the notion that while workflow models often cover the common case, qualified workflow participants may need the ability to make autonomous changes in certain cases.

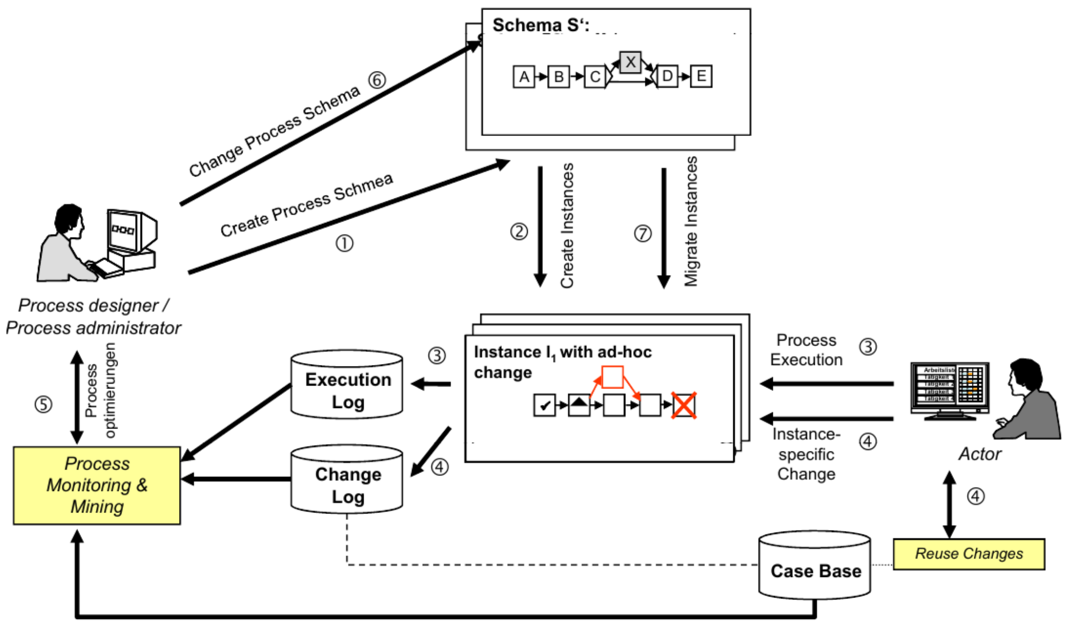


Figure 3.27: Lifecycle of changed workflow instances (from [160])

This means that the business process covers the most common case but unpredictable changes may necessitate a deviation from this standard case based on the judgement of a qualified participant. A common example for this sort of business process is the treatment of a patient by a medical professional, such as a nurse or a doctor. While there may exist a standardized treatment plan for certain medical conditions, the doctor or the nurse may notice unusual reactions or symptoms which require a deviation from standardized procedure such as ordering additional tests.

Consequently, participants or actors in such workflows need to be enabled to change the workflow instance after it has been enacted from the workflow model and is executing on the workflow management system as shown in Figure 3.27. The Adept2 approach generally follows the principles of task-based workflow management but proposes that actors who are participating in the workflow are granted the ability to change running workflow instances by inserting or deleting execution paths and tasks.

The system includes a number of checks intended to ensure a level of consistency and assist actors to make a desired change. Furthermore, if a certain change proves to be generally useful, there is a path for merging the change back into the general workflow model so it may be reused in future cases. This allows continuous improvement of workflows driven by participants over the lifetime of a workflow model.

The Adept2 approach represents an example of how workflow may be adapted in case of unpredictable changes in a business environment and therefore is a possible path for the excluded research goal of workflow instance agility as shown in section 1.2. Due to their unpredictability, contingencies for such changes cannot be included in workflow models beforehand. However, Adept2 allows alterations to the workflow instance once situation of alternate business environments surface.

However, this high level of runtime flexibility comes at a cost that cannot be easily reconciled with some of the other research goals and poses some issues in practice:

- As in the task-based approaches, there is no direct association of business goals with the tasks being performed. This is particularly critical in the case of workflow instance agility since there is no assurance that added tasks serve any particular business goal of the organization. Consequently, strategic-operational cohesion is not enforced and may be lost due to arbitrary changes.

- The approach of allowing workflow participants to modify workflow instances shifts the balance between global control and local autonomy strongly towards the autonomy aspects. Combined with the potential loss of strategic-operational cohesion and considering that workflow participants often have personal interests that diverge from the organization's interests (see section 2.3), the approach bears the risk of a loss of strategic control.

- Predictable changes to the business environment that could be included as structured contingencies either have to be address at runtime with the associated unnecessary loss of structuring or necessitate similar branch-heavy approaches as task-based approaches.

- The qualification of the workflow participants regarding workflow design is not necessarily guaranteed. While workflow participants are experts in their specific field, they are usually not experienced workflow designers. Since the modification approach requires them to modify the workflow model of a running workflow instance, they may not be able to fully understand the implication of the change or simply be unable to provide an ad-hoc modification that produces the desired results.

Nevertheless, Adept2 demonstrates an interesting approach for addressing the challenge of unpredictable changes or changed requirements in the business environment. However, it appears to be most useful for the area of ad-hoc workflows that center around a common case with participant-driven individual adaption for each instance. The next section will present a similar approach that is explicitly centered around case management and therefore provides a certain runtime flexibility.

## 3.11 Case Management Model and Notation (CMMN)

Another alternative to task-based modeling approaches is the Case Management Model and Notation (CMMN). Like BPMN, it was standardized by the Object Management Group (OMG) and is intended to represent standardized language for case management (see [122]) the same way BPMN is a standardized language for business processes and workflows.

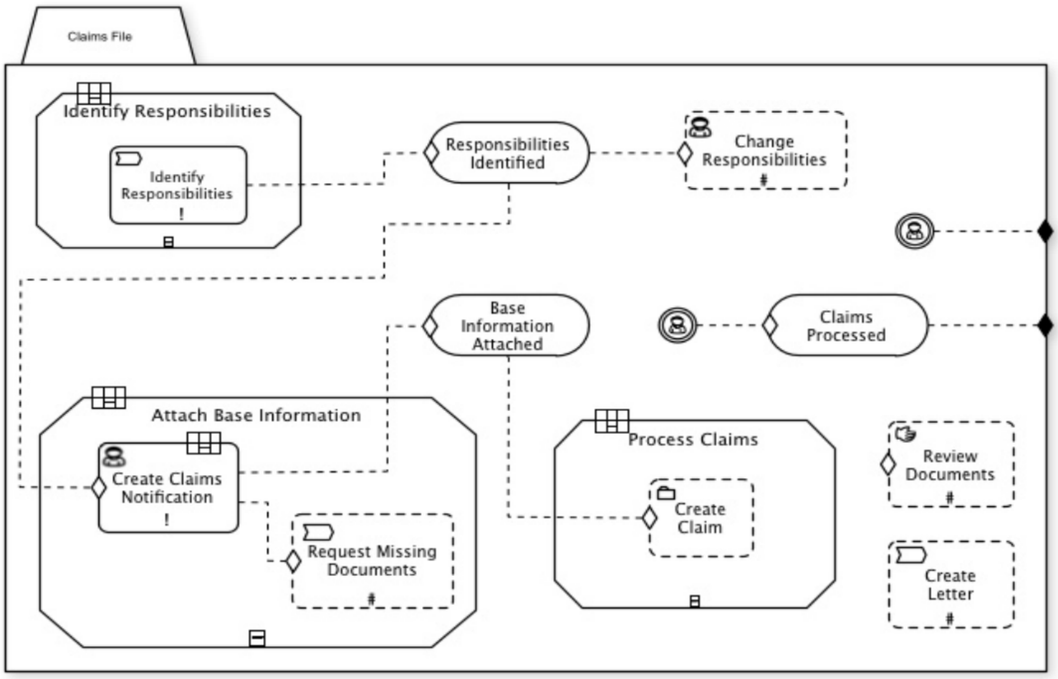

Claims File
Identify Responsibilities
Identify Responsibilities
Responsibilities Identified
Change Responsibilities
Base Information Attached
Claims Processed
Attach Base Information
Create Claims Notification
Request Missing Documents
Process Claims
Create Claim
Review Documents
Create Letter

Figure 3.28: Claims management example in CMMN from [122]

Case management derives from the legal and medical fields in handling cases for particular individuals. It focuses on a particular business situation and attempts to resolve it towards a desired outcome. The focus here is on an individual case instead of a generalized process, so while it may contain similar sets of actions, the use of those actions are dependent upon the individual circumstances.

In terms of business process categories as presented in section 1.1, case management is centered around the area of ad-hoc business processes, which are highly unique and customizable to the situation at hand. The basic building block of this approach, as shown in Figure 3.28, is the case which represents a proceeding involving actions regarding a subject in a particular situation to achieve a desired outcome (see [122]). The notion of a case was inspired by the legal and medical fields where this approach is common for handling business situations.

Unlike the structured approach of task-based modeling language, cases may be resolved in arbitrary ad-hoc manners and are not necessarily bound to strict execution orders. During the workflow modeling, the workflow designer can add tasks to the case. Two different types of tasks can be distinguished: Mandatory tasks, which are always part of the case and therefore the workflow execution and discretionary tasks which can be invoked by the workflow participants on a case-by-case basis at runtime. This is often done by a special participant called a case manager, though other models of participation are supported.

CMMN as an approach allows the inclusion of predictable changes in the business environment as part of the workflow model. However, it's case-based structure is primarily aimed at ad-hoc business processes. Similar to the Adept2 approach, it does not include a connection to the strategic planning and therefore does not provide a strong strategic-operational cohesion. This is often acceptable in ad-hoc processes since the complexity of such processes tend to be limited but it becomes more difficult with regard to the more complex area of collaborative business processes. However, while CMMN offer more limited workflow instance agility compared to Adept2, it's more structured approach provides a better balance between local autonomy and global control by offering the workflow participants a number of options in the form of discretionary tasks.

Overall, CMMN is a recent and interesting approach for the area of ad-hoc business processes which has some limitations regarding more complex collaborative business processes. In the next chapter, a goal-oriented approach for business process modeling is introduced which is specifically aimed at these collaborative business processes which ties the actions with the business goals of the process, providing strong strategic-operation cohesion and further strengthening global control while maintaining a high level of local autonomy.

Chapter 4

# Goal-oriented Business Process Modeling and GPMN

As shown in section 1.2, many product development processes are examples of long-running autonomous processes and therefore exhibit the requirements of additional workflow model agility and the need for strategic operational cohesions. These issues where also noted in research done by Burmeister at Daimler AG while attempting to model development processes which can be categorized as long-running autonomous business processes. As a first step towards solving the issues, the idea of using the business goals derived during the strategic planning phase (see chapter 2) for modeling such processes was proposed and programmatically implemented using agents (see [30]).

Using the business goals as modeling elements not only establishes a firm link between the business strategy and the operational execution of workflow, by providing clear reasons in the form of goals for all actions in the business process instance, but may also add flexibility to the execution of the workflow by separating the target objective from the means to reach it.

Programmatical implementation of business process models is less than ideal since it tends to exclude non-technical people from the design process (see chapter 2). As a result, a modeling approach was developed together with Whitestein AG in order to graphically model such workflows called GO-BPMN (see [31]), which is part of the Living Systems Process Suite and presented here in section 4.1. This approach centered around a conservative BPMN extension using an interpreter to represent business goals in process models.

Since the interpreter approach has a number of limitations (see section 5.4), a more complex approach which unifies both approaches by offering both graphical modeling and respresentation as well as agent-based semantics is part of this work and presented in section 4.2 called Goal-oriented Process Modeling Notation (GPMN).

Both concepts use the same basic approach for designing business processes and workflows using goals but use different means for enactment and GPMN adding additional semantic depth. In both approaches, the modeling starts with the main business goals being added to the process. Since these goals are often somewhat high-level and complex, they are successively broken down into smaller subgoals: For each of the main business goals, smaller subgoals are defined with the notion that reaching all of the subgoals implicitly causes the original goal to be reached as well. This approach can then repeated for the set of resulting subgoals as well, breaking them further down into smaller and smaller subgoals, creating a *goal hierarchy* with the original business goal as the root node.

This breakdown of goals is repeated until the leaves of the goal hierarchy describe goals that are sufficiently limited in scope so that they can be reached using a very simple set of instructions. These instructions are provided using plan elements, with one or more plans being attached to the goal as options for reaching the goal.

Since both approaches are based on the same premise of using business goals as elements, both concepts exhibit similarities in their approach for providing a goal-oriented business process language. However, in part due to the different semantics, there are some key differences between the two concepts.

In this chapter, the GO-BPMN approach is briefly introduced, followed by a detailed description of the GPMN approach. Finally, the two approaches are compared with regard to the modeling capabilities.

## 4.1 Goal-oriented BPMN (GO-BPMN)

Goal-oriented BPMN (GO-BPMN, see [31]) was the next attempt after the goal-context approach envisioned and programmatically tested at Daimler AG and was aimed at allowing the modeling of goal-oriented process in a graphical way in line with other business process modeling languages like BPMN. The GO-BPMN language was designed and implemented by Whitestein AG as a conservative extension of the BPMN language approach. Since, like GPMN, it is based on the idea of using business goals as model elements, it uses a similar set of modeling language elements centered around the use of business goals but with different execution semantic than GPMN and a different approach for enacting instances of the process models.

The GO-BPMN approach follows the goal-oriented pattern business process and workflow design, which is based on using the business goals of the process itself as defined in the strategic planning stage of business process management, as modeling elements in the language as well as a starting point and method for developing a business process model.

As shown in Figure 4.1, a process modeled in GPMN consists of a hierarchy of business goals, the root nodes of which represent the business goals for the particular process. These goals are then broken down into subgoals that represent the top goal. These subgoals are eventually simple enough to be implemented using plans, with multiple plans that can be attached to a goal as options for fulfilling the goal (e.g. EstimateCost and CalculateCost in the example).

Since GO-BPMN is considered to be an extension of BPMN, it follows a similar pattern with regard to goal and plan execution: When a workflow instance is running, a goal can be in one of six different states:

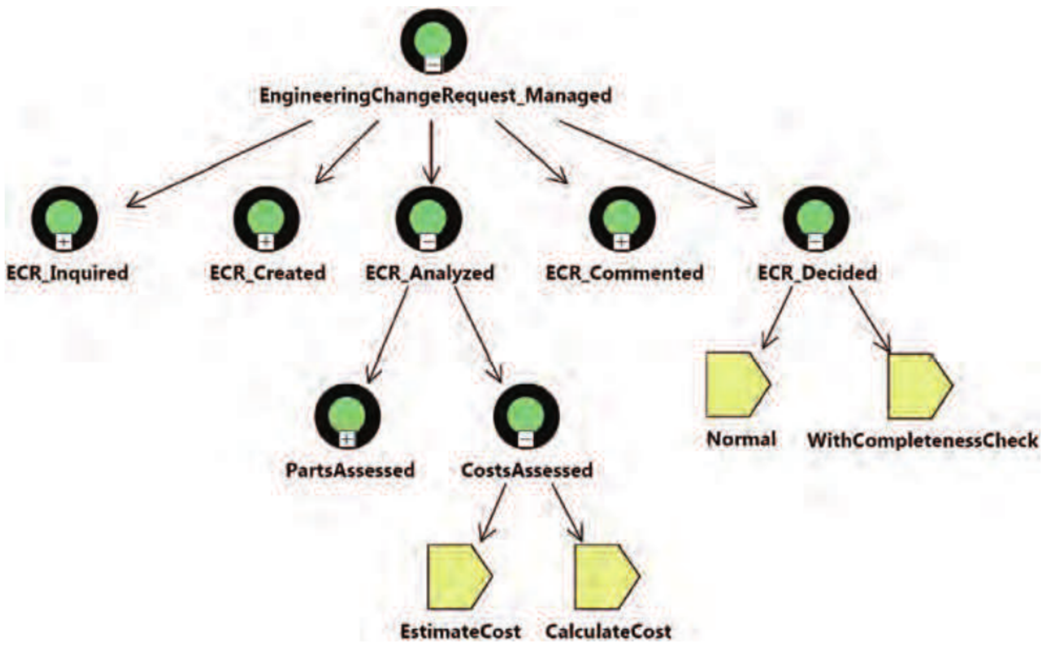


Figure 4.1: Example of a GO-BPMN process

- *Inactive*, if the goal has not been active before (default state).
- *Ready*, when a goal has been activated but is not currently executing.
- *Running*, when a goal is currently executing.
- *Failed*, if a goal was executing but has failed to achieve its objective.
- *Achieved*, if a goal executed and has managed to achieve its objective.
- *Deactivated*, if a goal had been activated but was later deactivated before achieving its objective.

Goals are therefore treated in a similar fashion to activities in BPMN process, which also have a certain execution state. Generally goals become activated through the edges used in the goal hierarchy, however, goals that are not currently in an active state can be activated or reactivated manually in plans similar to the approach chosen in GPMN.

GO-BPMN offers two kinds of goals, *achieve goals* that aim to meet a certain state and *maintain goals* that aim to maintain it beyond the point of having achieved it once. The achieve goal implements all of the states mentioned, however, maintain goals lack the achieved and deactivated states since they do not reflect their specific meaning in the process.

Plans in GO-BPMN are the means to achieve goals having no subgoals attached to them. This means that they are only acceptable to be attached to leaf goals of the goal hierarchy. The implementation of plans is done in BPMN which, during runtime, is directly executed using an interpreter approach. Plans can optionally be hidden from view to allow for a more high-level and goal-centered perspective which focuses on the objectives rather than the means of achieving them.

The workflow example in Figure 4.1 shows how a workflow designer can use the option of attaching multiple plans to a goal. The goal CostsAssessed has the objective of providing a cost figure as part of a larger analysis goal. In order to achieve this, the workflow designer has provided two different plans as means for achieving the goal: First, the CalculateCost plan which performs a precise cost calculation and second an estimation plan called EstimateCost to allow for a rough cost estimate.

The reason for the two options is that while a more precisely calculated cost figure is obviously superior to a rough estimate, it is not always desirable to calculate such a figure. For example, there may be too many uncertainties impeding a precise calculation of cost or there may be a time limitation which does not allow for a lengthy cost calculation. Additionally, not all cases where the workflow is used may require a precise cost figure, which may be difficult or itself costly to produce, with a rough estimate being ”good enough” for the purposes of that particular workflow instance.

The execution of both goals and plans is controlled using conditions. GO-BPMN offers four types of conditions to a process designer, however, they are usually restricted to certain elements in the language:

- Pre-conditions can be applied to achieve goals and only allow an achieve goal to be executed if they evaluate true.
- Deactivation conditions will deactivate achieve goals if the condition is met.
- Maintain conditions define the state that should be maintained by a maintain goal.
- Context conditions define when a plan is selectable as an option for achieving a goal.

The conditions are based on the business process state which contains named elements that can refer both to internal engine states as well as user-defined business states based on typed and named entries. This approach mirrors the concept of a process context in GPMN and fulfills a similar function within the process.

## 4.2 Goal-oriented Process Modeling Notation (GPMN)

The goal-oriented process modeling notation is a business process and workflow model language that was developed to facilitate the research goals of workflow model agility, strategic-operational cohesion and to address the need for a balance between the global control of a process and the local autonomy of business process participants in long-running autonomous processes (see [87] and [81]). It is based in part on earlier work on the goal-context method developed by Burmeister of Daimler AG (see [30]) and the graphical modeling approaches found in GO-BPMN (see [31]) but unifies the two concepts and enhances their overall abilities by making graphical workflow modeling available as well as leveraging more capabilities of the original programmatic goal-context approach.

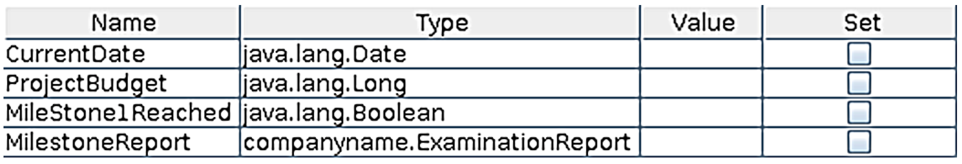
### 4.2.1 GPMN Process Context

GPMN centers around the idea of directly including business goals as first-class entities in the process models and representing the business environment and situation as the process *context*. The process context exists implicitly in GPMN processes and consists of a number *context elements* or *context entries* which are key/name-value pairs in which the name is both a description of the information contained in the value as well as a handle to access those values. The values of the context are statically typed based types available in the Java language. This includes custom and complex types which can be implemented as Java classes to help the workflow designer to provide complex business objects.

Furthermore, context entries can alternatively be defined as sets. Context sets also consist of a name and a static type but allow multiple values to be assigned to the entry during runtime such as multiple reports or business documents of the same type.

Name-value pairs within the context represent two different types of information regarding the business environment. First, it can contain business information that are part of the workflow state, such as business objects that are currently processed by the workflow or information temporarily stored by the workflow for later use. These values are modified by the workflow itself while it is executing.

In addition, the context also contains elements representing the business situation. This can include all kinds of business information that may impact the workflow in some fashion, for example, expected delivery dates provided by suppliers, warehouse contents and sales data. When modeling the context, it is key to represent the situation of the business accurately, however, in order to limit excessive information which needs to be observed and updated at runtime, it should be limited to information relevant to the business process.



| Name | Type | Value | Set |
|---|---|---|---|
| CurrentDate | java.lang.Date | | ☐ |
| ProjectBudget | java.lang.Long | | ☐ |
| MileStone1Reached | java.lang.Boolean | | ☐ |
| MilestoneReport | companyname.ExaminationReport | | ☐ |

Figure 4.2: Example of a GPMN process context

Figure 4.2 shows an short example of a GPMN context configuration. It contains four context entries which can be used in the business process. The first entry is the current date as it applies to the workflow instance and uses the predefined Java Date-class as the type. The second entry denotes the project budget and is typed as a value based on the Java Long class. The following entry is a flag based on the Boolean class, which marks whether the first milestone was reached in the process. The fourth entry is the milestone report which is presumably set once the first milestone is reached. The report is typed as a custom Java class which implements a domain-specific business object. None of the entries is marked as a set.

Entries can include an optional default value which is set when the process starts; however, this is optional and is only used if the entry must avoid providing an undefined value.

The values in the context can be accessed by conditions and plans (see sections 4.2.3 and 4.2.4). This can be accomplished using a notation which prefixes the name of an entry with "$", which causes a reference to the value object to be injected in place at runtime. For example, if one wishes to invoke the booleanValue() method of the Milestone1Reached entry, the term would be "$Milestone1Reached.booleanValue()".

The process context will influence the execution order and timing of the workflow instance and therefore represents a key part for the goal of workflow model agility: By including a model of the current business situation, the workflow model can then use that information to decide whether a predicted change of the business situation has occurred and chose the contingency included in the workflow by the designer to remedy this situation.

### 4.2.2 Graphical GPMN Elements

GPMN is a graphical business process and workflow modeling language in which a small number of elements are used to define the workflow semantics based on the process context. GPMN departs from the control flow-centric and task-based modeling and emphasizes the use of business goals. In addition, it also includes a plan element for concrete actions, a subprocess element for referring to subprocesses, activation edges to connect goals, plan edges to attach plans to goals and suppression edges to model the suppression of goals in certain situations.

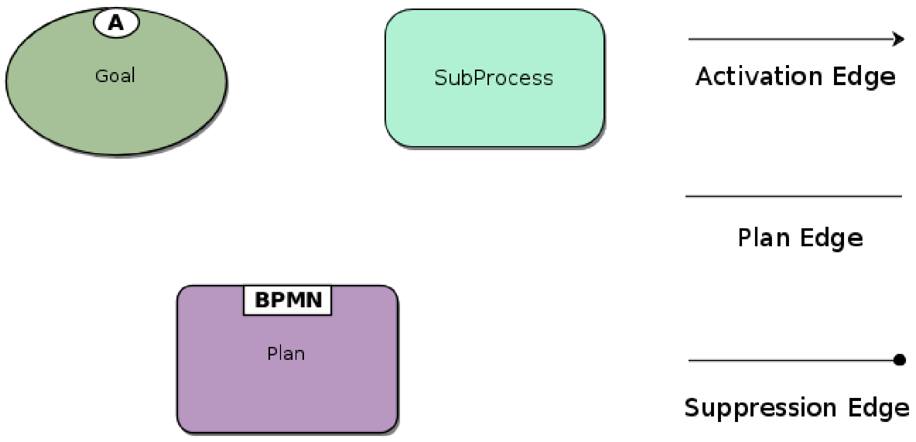


Figure 4.3: Elements of the goal-oriented process modeling notation

Figure 4.3 shows the elements of the language. The key element in the language is the goal element. Goals represent a certain state that may be desired by the workflow at some point during execution. Ideally, this element is used to directly represent the business goals of an organization's strategic planning.

Business processes typically have only a small number of high-level business goals that the process aims to achieve. For example, the typical production process may have the business goal of assembling a product. A research and development process may have business goals such as designing a new product or developing a production process for such a product.

While these goals accurately describe what a business process is expected to deliver, they tend to be high level and it is not easy to derive potential actions directly from them. Therefore, GPMN allows the model designer to break down business goals into subgoals, which are connected to the original goal using activation edges, creating a *goal hierarchy* directly representing the strategic goals of the organization for the business process.

The semantics of this connection is that the connected subgoals represent the complete set of goals that must be accomplished for the primary goal to be completed. In other words, a goal is considered to be achieved once all of its connected subgoals have been achieved. During execution, an active or adopted goal activates its subgoals, creating a goal instance of the subgoal (the top goals have their goals instance created during the start of the workflow). This basic behavior of activating subgoals can be modified in some cases as described in the next section.

The subgoals themselves again can be further divided into smaller goals that are attached to that subgoal. This approach lets the workflow designer start out with the high level business goals and successively break them down into smaller and smaller subgoals.

Once the designer has reached a point where the goals are sufficiently simple, concrete actions can be associated with them. This is done using *plans* which are attached to a goal using *plan edges*, which means that plan edges can only connect a goal with a plan, connecting two plans or two goals being excluded. Plans represent a set of instructions on how to accomplish a particular goal while the plan edge represents the association with that goal. This means that plans are possible courses of actions for the workflow to reach the desired state as defined by the associated goal. As a result, multiple plans can be attached to a single goal in order to offer multiple solutions for goal completion. Goals therefore represent target states for the workflow while plans are the means available to reach those states.

Due to the refinement of the goals into subgoals, the leaf nodes of the goal hierarchy are generally very simple goals where the means of reaching them can easily be expressed in very simple instructions. When working on the implementation towards a workflow model, these plans usually contain a reference to a BPMN workflow fragment, however, other options for expressing the plan are possible. A currently implemented alternative is the definition of the plan as a Java class.

*Suppression edges* offer a possibility to graphically model certain dependencies between goals and therefore can only be used to connect two goals. One thing that can occur in a goal-oriented workflow is that two goals are in conflict and, when pursued at the same time, may interfere with each other. For example, the goal of maintaining room temperature interferes with the goal of refreshing the air in the room. Suppression edges allow one active goal to temporarily suppress another active goal while it is active. In the given example, the goal for maintaining room temperature is suppressed while goal for refreshing the air is active (and executing its window opening plan). Once the suppressing goal has achieved its objective, the suppressed goal can be pursued again.

Subprocess elements can be used to modularize very deep goal hierarchies and reuse them in part in other processes. Semantically, they behave like goals and can be activated by other goals using activation edges, but do not allow the attachment of plans nor can they have further subgoals. Instead, if a subgoal is connected to a goal using an activation edge, it is considered equivalent to attaching the root goals of a referred subprocess to that goal.

It is also possible to attach plans to a goal and, in addition, connect subgoals using activation edges to the same goal. In terms of process semantics, this pattern treats the adoption of the attached goals as one of the options for achieving the main goal. In other words, once the main goal activates, it can either attempt to reach the desired state by executing one of the plans or, alternatively, adopt one of the attached goals.

Goals, subprocesses and plans can contain strongly typed parameters similar to the context entries, which are runtime arguments that can be passed to the goal instances and subprocess instances once they activate or to plan instances once they are executed. They can be implicitly passed from goal to subgoal or subprocess once activated or to a plan once executed from the parameters of the top goal. Upon successful execution they are mapped back into the previous goal or alternatively into the process context if a match can be found.

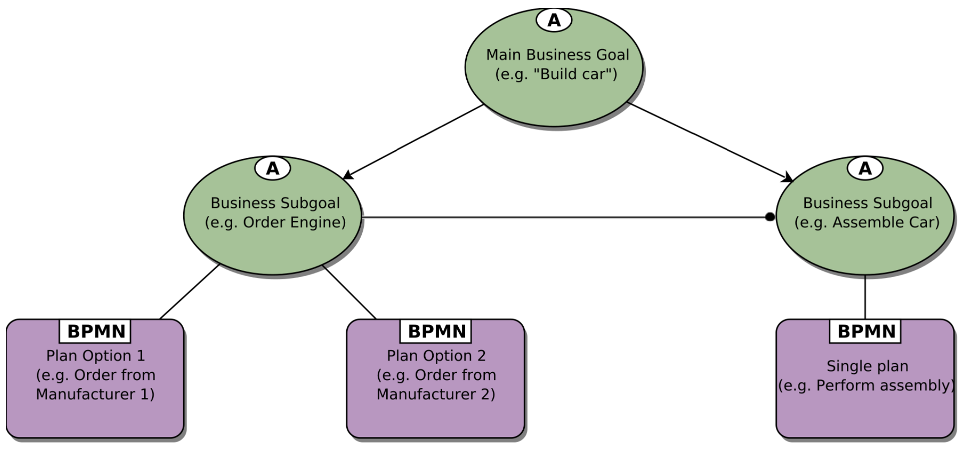


Figure 4.4: Simple example of a GPMN process model

Figure 4.4 shows a very simple example of a business process modeled in GPMN. The main business goal in this process is to build a car and is represented by the goal at the top of the goal hierarchy which has no incoming activation edges. While this example process only has a single main business goal, it is possible to model processes which pursue multiple business objectives at once by adding more root goals to the goal hierarchy.

In this simplified example, the main business goal is then split into two subgoals which must be achieved to achieve the main business goals. The first subgoal is to acquire an engine for the car. The second business goal is to assemble the car.

In this example, the goal hierarchy ends after only this simple split into two subgoals. In complex real world processes, the subgoals are often subdivided into further subgoals until sufficiently simple goals are reached. However, both the goal of ordering an engine as well as the goal of assembling the car is not further divided, instead plans are attached representing options for activities for the workflow.

The engine order goal has two plans attached, one which orders the engine from one particular supplier while the other one orders from a different supplier. Another alternative plan could be to produce the engine in-house. During execution, the two plans mean that the workflow has two options for accomplishing the goal. The basic unmodified semantic is to attempt one plan and failing that, try one of the remaining options. The car assembly on the other hand only has a single plan attached, meaning it's the only available means to reach the goal of assembling the car.

Since the car cannot be assembled without an engine, a suppression edge is used: When the process starts, both goals are adopted by the workflow, but the goal of ordering the engine is temporarily suppressing the goal to assemble the car. Only once the goal of ordering an engine has been accomplished can the goal of assembling the car be pursued.

This process represents a very basic example of a GPMN workflow, however, further details can be configured and changed. In order to further refine the semantics of the goals for example, additional information can be added to specify their exact behavior. The next part will give a description of the different goal kinds that are available and how they can be configured.

### 4.2.3 Conditions and Goal Kinds

Not all business goals that can be found in many business processes including long-running autonomous processes describe the same semantic idea. As a result, while attempting to model different kinds of such business processes, four different kinds of goals were found to be useful in different contexts (see [24], [87] and [81]). For example, in a business process for designing a product that is required to be compliant with certain norms, the goal that requires the compliance criteria to be met is not merely a goal that needs to be reached at one point in the development process. Instead product compliance must be maintained over the whole of the development and include all changes that happen during development.

Furthermore, there must be an option to include the business environment in the evaluation of goals to achieve workflow model agility. This is accomplished using the process context as described in section 4.2.1 in combination with different kinds of *goal conditions*. Goal conditions are statements that evaluate to either true or false and are currently expressed in Java syntax based on the values accessible in that context.

In addition to goal conditions, goals are also available in different *goal kinds*. Goal kinds define the basic semantics of the goal. Some goal conditions are only available to some of the goal kinds; however, the following conditions can be specified for all goal kinds:

- *Creation conditions* allow the activation or creation of a goal instance if the expression defining the condition evaluates true. If changes state multiple times, it can lead to the adoption of two or more goal instances at the same time which are both pursued independently based on the semantics of the model. An example use case for this condition could be included in a goal that is part of a goal hierarchy normally handling the extension of credit to a customer if so desired. The creation condition is useful for explicitly triggering that part of the goal hierarchy if a customer is unable to pay after the purchase but has previously declined applying for a credit.

- When *drop conditions* evaluate as true in a particular goal instance, that goal instance is terminated regardless of whether it has met its objective. In most cases, drop conditions specify under which circumstances it is no longer worthwhile to pursue a particular goal. In the example workflow shown in Figure 4.4, the goal for acquiring an engine could have a drop condition that triggers if an engine is already available in the warehouse, dropping the goal after it was adopted and thus avoiding ordering unnecessary items.

- *Context conditions* are semantically similar to suppression edges in that they prevent the workflow instance from pursuing a particular goal it has adopted, but allow for more complex modeling by specifying a condition based on the context. The price for the added modeling flexibility over the suppression edge is the lack of graphical representation. For example, instead of using a suppression edge, the goal for car assembly shown in Figure 4.4, the goal could include a context condition that inhibits the goal until an engine becomes available.

These conditions, which are common to all goals, allow workflow designers to shape the semantics of the goal. However, goals themselves are available in four different kinds, some of which allow additional conditions to be specified which influence the behavior of that particular goal kind.

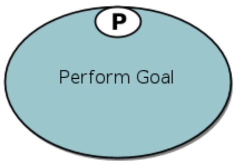


Figure 4.5: A GPMN Perform Goal

The first goal kind available to business process and workflow designers is the perform goal. It is represented as an oval but includes the letter „P" at the top to denote its kind (see Figure 4.5). A goal of the perform goal kind represents the idea of simply performing an action, in other words, the desire to have certain activities done. If multiple plans are attached to this goal, executing any of the plan is sufficient to consider goals of this kind to be finished, regardless of what occurs within the plan since only the performance of the activities is relevant to this goal kind. Beyond the three kinds of conditions previously stated, the perform goal kind does not have additional conditions.

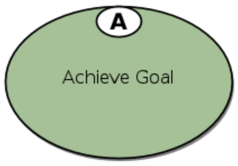


Figure 4.6: A GPMN Achieve Goal

The second goal kind and the most common kind in many workflows is the achieve goal. The achieve goal is usually what many people often intuitively perceive as a goal. Goals of this kind denote that something needs to be accomplished or a certain state be reached. In other words, achieve goals describe a desired state of the business environment as represented by the context. As a result, the achieve goal includes an additional condition type to describe this desired state:

- *Target conditions* denote a state of the business environment as represented by the process context which the goal aims to reach. Once the target condition evaluates true, the goal is considered to be successful and is terminated.

As a shorthand, the workflow designer can also omit entering a target condition for an achieve goal. In this case, the target condition is implicitly considered to be a successfully executed plan, or, in other words, if a plan executes and finishes without generating an error, an achieve goal without target condition is considered to be fulfilled.

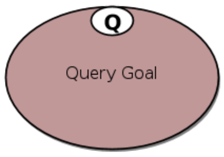


Figure 4.7: A GPMN Query Goal

Another goal kind in GPMN is the query goal. Query goals are a specific goal kind for information retrieval. Variables can be defined in the query goal which denote the information that the goal aims to acquire. The semantics are then similar to the achieve goal in that the goal is considered to be finished and a success once all variables have a value assigned to them. Like all goals, the attached plans or subgoals are executed, in the case of the query goal, this is done until all variables are assigned. Once a query subgoal has achieved its objective, the retrieved values are then transferred into the process context. Aside from the variable definitions, the query goal does not include any additional conditions beyond the three default goal conditions.

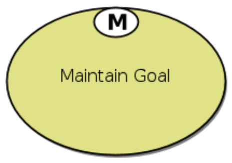


Figure 4.8: A GPMN Maintain Goal

The final goal kind is semantically the most complex type of goal. In some situations, it is the business goal of a process not only to reach a certain state in the business environment but continue to maintain a state afterwards in case a violation occurs at a later point. For example, a business process may go through a number of tests in order to reach a milestone in the process. Once the process has reached this milestone, it should ensure that later actions of the process do not impede the state reached in the milestone and, if it should occur, perform any needed measure to remedy the loss of conformance with the process milestone.

As a result, the maintain goal offers a condition to define the state that needs to be continuously observed and maintained as long as the goal is active:

- A maintain condition defines a process context state that not only needs to be reached but continuously observed while the goal is active. Should the maintain condition be violated, a maintain goal causes the process to execute attached plans or subgoals until the violation is remedied.

However, in some cases single boundary conditions are not enough to completely define the behavior of all different maintain goals. For example, a process may want to ensure that at least ten items of a particular type are in stock in the warehouse. If a maintain goal only specifies a maintain condition that the quantity of the items in stock must always exceed ten items and an attached plan only included a purchase order for a single item, the goal will activate whenever a single item is removed once the boundary condition has been reached once.

As a result, the maintain goal also allows for specification of a target condition similar to the achieve goal though the semantics are slightly different from the achieve goal. If a maintain goal specifies both a maintain condition and a target condition, the goal will, once activated, continuously observe the maintain condition. Once a violation occurs, measures such as plans and subgoals, are executed until the target condition is reached, after which the maintain goal returns to observing the maintain condition.

Goals also include a number of flags and configuration options that allow workflow designers to influence goal behavior as well as choices of the workflow regarding plans and subgoals that are available to a goal. Normally, all plans and subgoals are tried one after the other until the goal has been fulfilled. Previously attempted and failed plans (plans that did not reach the conditions specified by the goal) are excluded from further attempts. After all plans have been tried and the goal is not fulfilled, the goal has failed.

The following configuration options allow some configuration of this default behavior:

- The *retry* flag specifies that a goal should retry another plan or subgoal option after one has already failed. This flag defaults to a true state but can be set to false which causes a goal to fail after failing only a single plan or subgoal hierarchy.

- If the retry flag is set, a *retry delay* can be specified. In the default case, the workflow will try another plan or subgoal hierarchy immediately after failing one. If a retry delay is set, a delay is observed before another attempt to reach the goal is made.

- The *exclude* configuration enables workflow designers to specify when plans should be excluded as options for reaching a goal. The default setting is *when_tried*, which will cause a plan to be excluded as an option if it has been tried before, whether the plan has previously been successful in reaching the goal (for example, restoring a maintain condition) or if it failed. Another available option is *never*, which prevents the exclusion of plans. If combined with the retry flag, all the plans are continuously tried until the goal has been reached. The last two remaining options are *when_succeeded* and *when_failed*. These two options exclude plans as options when they have previously been successful or when they have previously failed.

- Normally, plans are tried in an unspecified but deterministic order. As an alternative, it is possible for workflows to pick a random plan from the available options. This behavior can be enabled by the workflow designer by setting the *randomselection* flag.

- The normal behavior of a GPMN workflow when considering an active goal with multiple plans or attached subgoal hierarchies is to try one of them after another until the goal is reached. However, sometimes it is desirable to try all of the available options at once, for example to quickly reach a result. This behavior can be enabled with the *posttoall* option, which causes all plans and subgoal hierarchies to be tried in parallel until the goal is reached.

As noted previously, goals can activate subgoals with activation edges. If multiple subgoals are attached to a goal, all the subgoals activate and are pursued in parallel unless somehow inhibited or dropped by suppression edges or conditions. However, a

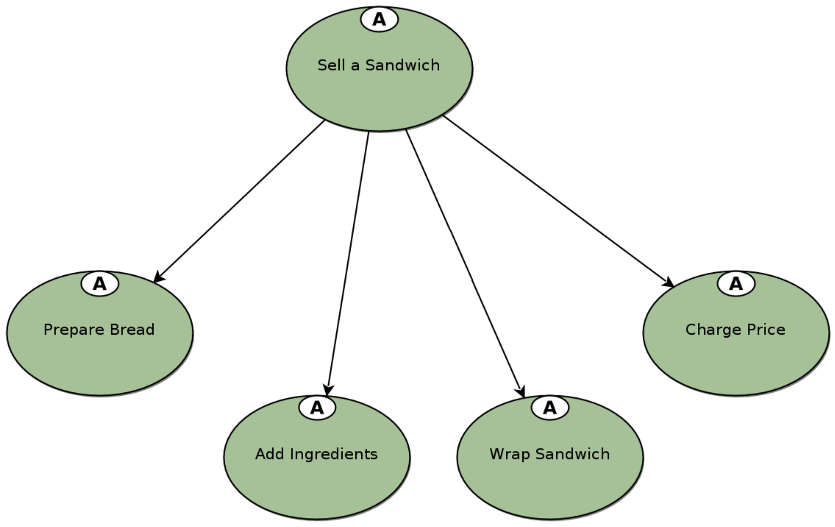


Figure 4.9: GPMN example process for preparing and selling a sandwich, plans are omitted for brevity

special but very common case of inhibited goals is the activation of goals in a specific order.

For example, in order to create and sell a sandwich, the bread must be sliced first before further ingredients can be added on top before the sandwich is wrapped and the price charged. In this case, one would start with a top level goal of selling a sandwich, then adding four goals for slicing, adding of ingredients, wrapping and charging as subgoals (see Figure 4.9).

However, to ensure the specific order of goals, one of two options would have to be used to establish the ordered behavior of the goals. The first option is to use context conditions for each of the subgoals. for example, the goal to add ingredients could include a context condition that evaluates to true once a cut bread is available. In this case the goal would be suppressed until the bread was cut. However, this approach has the disadvantage of lacking a graphical representation. In addition, the sequential activation of the goals needs to be coordinated using the process context, which is supposed to represent business information and should not, just like any workflow feature, be used to store technical information. While the information that cut bread is available may be relevant to the business, in this case the information only exists to facilitate a sequential subgoal activation order.

Alternatively, the subgoals could be connected using suppression edges which enforced the specific order. In this case, the cutting bread goal would have a suppression edge to the other three subgoals. The ingredient subgoal would have a suppression edge to the remaining two goals and so on (see Figure 4.10). Unfortunately, this modeling approach leads to a convoluted and increasingly complex graphical representation with the number of suppression edges growing quadratically as a triangular sequence: $E = \frac{(G-1)G}{2}$, where $E$ is the number of suppression edges and $G$ is the number of subgoals.

As a result, these approaches for modeling the sequential activation of subgoals

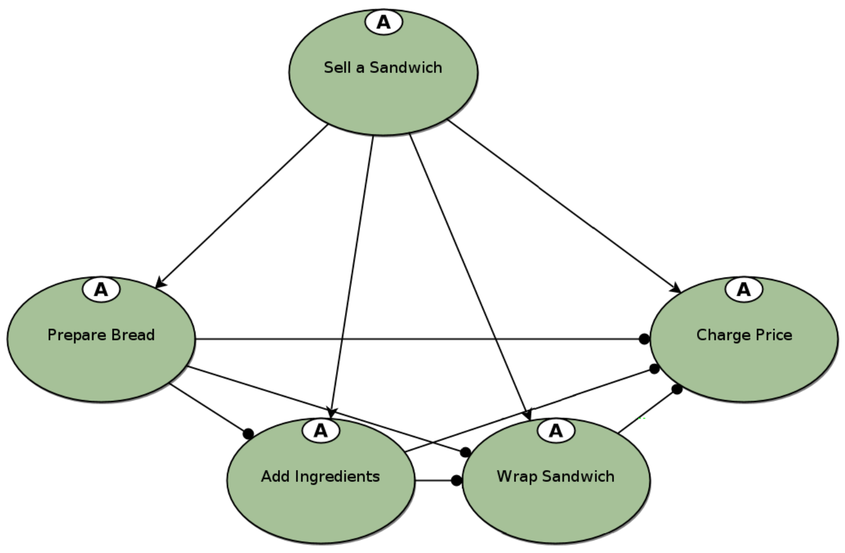


Figure 4.10: Sequential subgoal activation is enforced using suppression edges

is less than ideal, with the condition-based approach having a slight advantage in terms of complexity. However, this ordered pursuit of subgoals is a very common case (see [87]) and it is therefore useful to provide a more natural and compact option for modeling such sequential subgoal activations.

GPMN therefore supports a special mechanism for sequential subgoal activation called sequential goals. Any goal can be configured as a sequential goal, which is marked as such with the string "1..n" (see Figure 4.11). Sequential goals enforce a sequential order on subgoal activation, which is defined by the natural order of a set of numbers annotated on the outgoing activation edges. The order can be changed by changing the annotated numbers on the activation edges.

The next section will give a brief overview of plans and their configuration, which allows workflow designers to influence the options available to workflow instances.

### 4.2.4 Plans and Plan Configuration

In GPMN, plans represent the means for a workflow to reach goal objectives. They are attached to goals using plan edges. For a perform goal, a plan specifies the actions that need to be performed. When attached to achieve goals, they describe actions for the workflow to meet the target condition for the goal. Plans attached to query goals perform the necessary action to acquire the information defined by that goal, while plans attached to maintain goals either attempt to restore process context to a state where the maintain condition evaluates true or, if a target condition is attached, until that condition is reached.

In GPMN processes, plans are usually implemented as BPMN fragments which exist separately from the main process and are referenced by the GPMN plan element in the process. For example, one of the plans for purchasing an engine from the earlier GPMN example process in Figure 4.4 could be implemented as BPMN

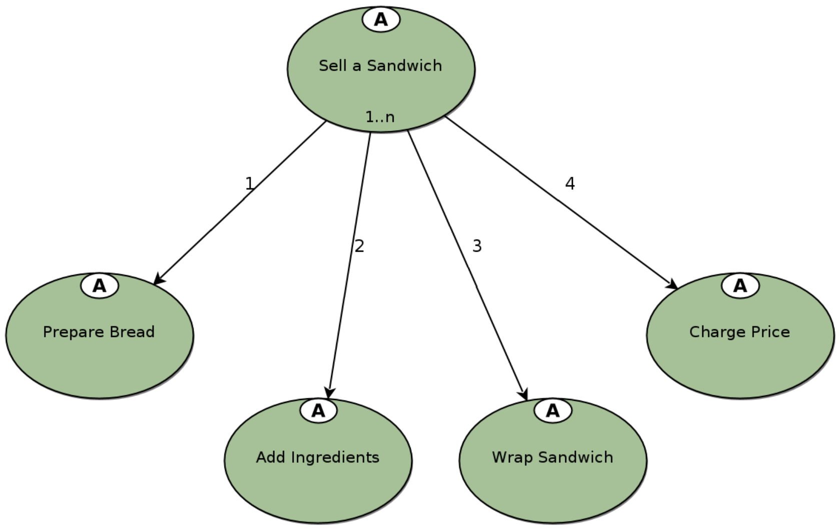


Figure 4.11: The "Sell a Sandwich" goal was converted to a sequential goal with the annotations on the activation edges defining activation order

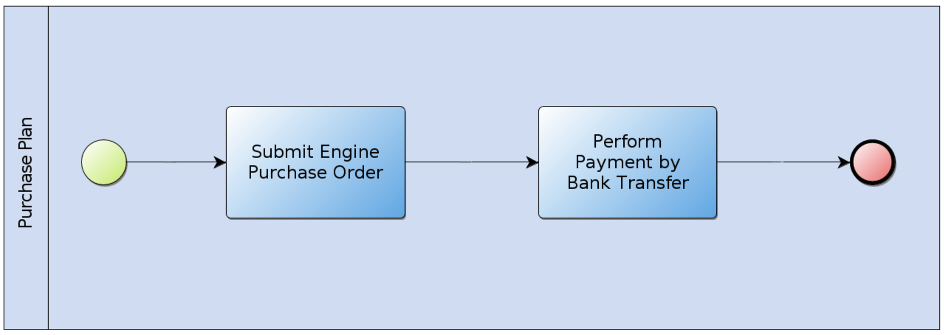

Purchase Plan
Submit Engine Purchase Order
Perform Payment by Bank Transfer

Figure 4.12: BPMN fragment used as a plan for purchasing an engine

fragment as shown in Figure 4.12. The fragment submits the purchase order to the manufacturer, then pays for the engine with a bank transfer. The plan for using the other manufacturer may, for example, include a different payment method such as credit cards or implement a different order submission.

Since the goal hierarchy already aims to minimize the scope of the goals as far as possible, the BPMN fragments implementing the plans are usually fairly simple as shown, however, the full complexity of BPMN as a language can be used to model BPMN plans if necessary.

As an alternative to using BPMN fragments as plans, GPMN allows referencing Java classes which are then used as a plan for reaching the attached goal. The referenced class must contain a special body method which implements the plan behavior. The plan is able to access the process context and parameters during execution. The implementation of plans as Java classes potentially allows for more flexibility compared to the use of BPMN fragments, but comes with two drawbacks; They have no graphical representation and are not immediately intelligible to users and designers who are not technically inclined.

As mentioned in section 4.2.2, multiple plans can be attached to goals in order to offer multiple options for reaching a goal. However, not all options are always applicable in all business situations. For example, if a manufacturer does not have any more production capacity or has ceased to produce a particular engine, the plan for ordering from that manufacturer should not be used when such a GPMN process instance is active. Therefore, plans can be configured with conditions similar to goals to describe when they are appropriate. The following two kinds of conditions are available for plans:

- A plan *precondition* defines whether a plan is an appropriate option for reaching a goal based on the current process context. If the precondition does not evaluate true, the plan is excluded when the decision is made to select a plan. However, once a plan has been chosen, it will be executed regardless of whether the precondition invalidates during the execution of the plan. This condition is useful when a plan requires a certain state to have a chance of successful execution. For example, a quick purchase approval may only be viable if a pre-approved budget exists, otherwise, a lengthier check may have to be performed using a plan without that precondition.

- The *context condition* of a plan on the other hand denotes the condition that must be true at all times for a plan to be viable. Unlike the precondition, this means that if the context condition invalidates during plan execution, the plan is no longer viable and is aborted in favor of an alternative. This condition is helpful when the business environment reduces the chance of a plan's success to zero. For example, requesting data through a network download (instead of, say, acquiring it by physical transfer of a data medium) may only be useful if network access is available. Once the condition changes, it is no longer useful to pursue the plan.

These conditions can be used in combination with goal flags like posttoall and randomselection to shape the options available to the business process based on the process context. For example, a posttoall flag set to true would normally cause all the plans attached to a goal to start executing in parallel, however, if some of the plans are excluded through preconditions, only a subset may execute.

Additionally, plans can activate any part of the goal hierarchy independently of the activations by means of the activation edges. For example, if a plan requires performing parts of the process again under some circumstances, it can manually activate a subtree of the business process by invoking a method in case of Java plans or using a special task in case of BPMN plans which causes the process to adopt the specified goal regardless of whether it has been activated before as part of the hierarchy. In fact, using this explicit adoption can be used to adopt a subtree of the hierarchy multiple times, thus creating multiple goal instances which can optionally be configured using the goal parameters.

The next section will provide the formal GPMN meta-model which formally specifies the available elements in GPMN as well as their potential interconnections. The section will also summarize the modeling options of GPMN as part of the presentation of the meta-model.

## 4.3 GPMN Meta-Model

The previous section provided an introduction to the goal-oriented process modeling notation (GPMN) and its graphical and non-graphical elements. GPMN is based on the idea of using business goals as a first-class citizen and executable part for modeling business processes and workflows.

Figure 4.13 shows the meta-model of GPMN and gives an overview of the available elements in the language. Each GPMN process can include one or more *activatables*, which can be activated during runtime by the workflow creating an instance of that activatable and denoting some desired state for the process to reach. Activatables also include parameters containing additional business information for the particular activated instance.

Activations of activatables are largely defined by *activation edges*. The default behavior is that activatables with no incoming activation edges are automatically activated once the workflow or business process is enacted. Activatables with incoming activation edges are activated once the source of the incoming activation edge becomes active.

The most common type of activatable are *goals*, which are the only activatable which can have outgoing activation edges. All goals can optionally be assigned three different kinds of *conditions*: The creation conditions, defining the state in which the goal exceptionally activated in addition the previous rules. Once a goal instance is active, it can become inhibited in case the *context condition* is invalid. Finally, the *drop condition* determines if the goal has become obsolete and should no longer be followed. In addition, goal instances can be inhibited by other goal instances if they are connected with a *suppression edge.*

Four kinds of goals are available for modeling: The *perform goal* expressing the desire to execute an action, the *achieve goal* which aims to achieve a particular state, the *query goal*, which describes the need for information and the *maintain goal* requiring the process to maintain a certain state. The achieve goal includes a *target condition* for the desired state and the maintain condition can include both a *maintain condition* describing the state to be maintained and the *target condition* for a state that needs to be reached once the maintain condition is violated.

The second kind of activatable is the subprocess. Subprocesses represent modularized parts of processes with potential use in multiple business processes. They can be attached to other goals using activation edges and denote a shorthand of attaching the whole subprocess goal hierarchy to the goal with the outgoing activation edge.

Concrete actions of the workflow are bound to goals as options for reaching the goal by means of *plan* elements. Plans can be implemented in two ways, as a BPMN Plan using a BPMN process fragment or as a Java class where the plan's actions are

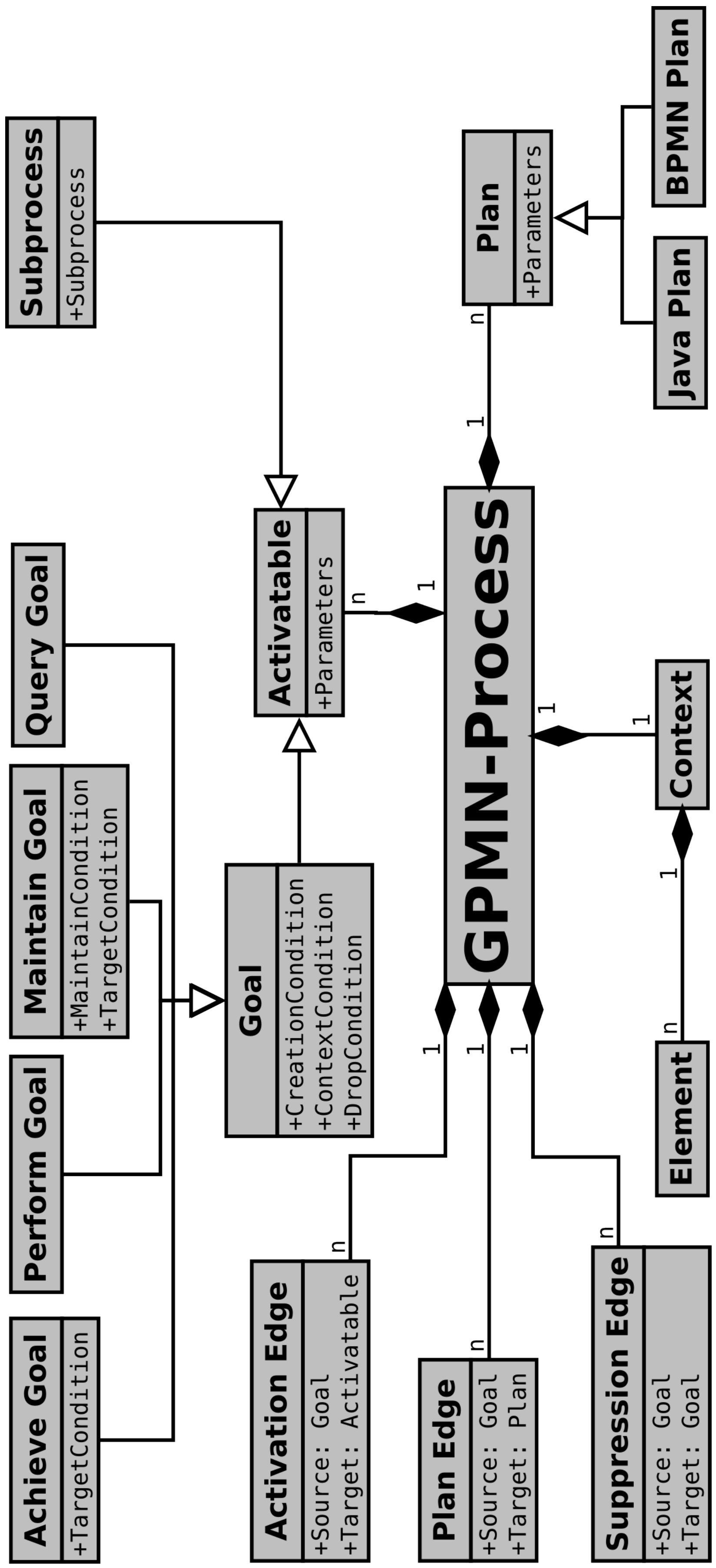


Figure 4.13: The meta-model of the GPMN language

described in a method within the class. Plans become options of action for goals by attaching them to the goal with a *plan edge*. They also including parameters which are passed from the goals attached to them using plan edges.

Finally, every GPMN business process and workflow contains exactly one process *context*, unlike all other GPMN elements which can be included in processes multiple times. The context represents the business situation, including all relevant internal and external factors for the process. It consists of multiple context elements or entries which are named as well as model specific aspects of the business environment. The values of the context elements can change at any time while a GPMN process is executing and influences both goals and plans through the previously described conditions. The context can include values that reference business objects internal to the business process or represent facts or information from outside the business process as part of the larger business environment of the organization which may have an impact on its execution.

This summarizes the meta-model of the goal-oriented process modeling notation. The next section will give a detailed overview of the execution semantic when executing GPMN instances followed by a section providing an overview of the XML-based storage format for GPMN business processes and workflows, which is then followed by an evaluation of GPMN with regard to the objectives of supporting long-term autonomous processes.

# Chapter 5

# Detailed GPMN Semantics and Model Format

This chapter provides a more in-depth overview of the semantics of GPMN during execution. In addition, a storage format for saving editable GPMN models to disk is defined. Finally, key differences of the semantic approaches of GPMN and GO-BPMN are compared using a number of examples in the last part of this chapter.

## 5.1 Goal and Plan Execution in GPMN

Business process and workflow instances based on GPMN models follow a strict distinction between the model elements and runtime instances of the elements. This means, for example, that when a goal is activated, a *goal instance* of the goal is generated which is based on the goal element in the model but has a distinct runtime state. The same approach applies to plans in that once a plan is selected for execution, a plan instance is generated, executed and destroyed once the plan execution has finished.

A consequence of this approach is that multiple instance of both goals and plans based on the same model element can exist in any given business process or workflow instance. While activation edges and plan edges generally only initiate the generation of a single goal or plan instance, additional instances can be created through other mechanisms, for example by a triggered creation condition or programmatic activation of a goal within a plan.

This distinction is important because it differentiates two different concepts about both goals and plans: Goal elements which are part of the process model represent objectives that may be considered worthwhile for the organization to follow under some circumstances, while a goal instance means that the process instance has now taken the goal into consideration. The same applies for plans, where a plan element in the model is a set of action that can be followed, while a plan instance is the same set of actions plus the accompanying runtime state which is being executed at a given moment as part of a business process instance.

This differs from approaches like BPMN where a task is a purely procedural construct which is either being executed or not without the explicit creation of runtime instances.

In GPMN, goals represent desirable states for the business process and business environment as represented by the process context, while the plans contain explicit actions to be performed. As a result, a distinction is made on how they are treated while a GPMN business process is enacted.

The first issue that arises deals with the question which of the available goal instances should be pursued at a given moment. With the creation of a goal instance, the business process instance has included it as one of the objectives it needs to achieve. However, the states goals try to reach may contradict each other and one state has to be reached first before an attempt can be made to reach the other. Therefore, a GPMN business process instance must make a decision which of the available goal instances should be suspended for a given time and which can be actively pursued using am approach called *goal deliberation*.

During goal deliberation, the business process instances evaluate the current goal instances based both on runtime and model information. Some of the available goal instance may be inhibited from being pursued and can be excluded. For example, if a goal has an outgoing suppression edge which is connected to a second goal, all goal instances of the second goal are inhibited while one or more goal instances of the first goal are available.

Another reason for goal instances to be inhibited is the violation of its context condition. In this case the goal instance is retained but not actively pursued until the context condition becomes valid.

Once the business process instance has decided which goal instances to actively pursue, a decision has to be made about the means of reaching the set of actively chosen goals. The means for reaching goals are defined in the model as plan elements; however, not all plans are equally suitable or worthwhile to use under a given set of circumstances.

The decision about which plan to execute in order to pursue goals is derived by a technique called *means-end-reasoning* (see [20]). During this cycle the business process considers the plans attached to the goal of the actively-pursued goal instances and derives which of them should be executed. This step involves the evaluation of available plans for each of the goals, some of which may not be available due to conditions based on the process context. If multiple plans are available that are not excluded, they will be tried either in definition order or at random until the goal is reached or until no more plans are excluded. The exclusion of plans after they have been tried can be controlled by the model designer through the exclude configuration in the goal as described in section 4.2.3.

## 5.2 GPMN Intermediate Format

In order to store business process and workflow models designed in the goal-oriented process modeling notation, a storage format is required. This is necessary for loading and saving functionality as part of business process and workflow design tools but also as a step for refinement into executable workflows by enabling the stored models to be utilized by a workflow engine. As a result, the storage format should be machine-readable; however, for error-checking and debugging by humans, it is helpful if the format is also human-readable as long as the machine-readability is not impeded.

A common compromise for storing information in a format that is both machine- and human-readable is the extensible markup language (XML, see [166]), which is also used as the default format for other workflow languages like BPMN (see section 3.7) and BPEL (see section 3.5). Since this choice seems to provide a reasonable compromise, the storage format chosen for GPMN is also based on XML (see [81]). Since the format also represents the intermediate step between a goal-based business process model and an executable workflow, the format is call the GPMN intermediate format and uses the file extension ".gpmn".

XML is able to represents arbitrary types of data and therefore allows a great deal of freedom in creating XML-conformant documents. However, in order to allow reasonable interpretation of the resulting documents, a structure definition has to be provided. With regard to XML, this task is usually accomplished using an XML schema language, which is a document specifying the XML elements which are valid within the context of the document, thereby providing a structure for tools and interpreters to process the format.

Currently there are a number of schema languages available to model XML schema formats such as the common Standard Generalized Markup Language (SGML, see [61]) and the W3C XML Schema (XSD, see [107]). The GPMN intermediate format, however, is defined in a schema format called Relax NG (see [34]), but uses the data types as defined in the specification of XSD (see [14]). The full specification of the GPMN intermediate format is listed in Figures 5.1, 5.2 and 5.3).

The format uses its own XML namespace which is usually denoted as "gpmn", which is the pattern followed in this text, but a different qualifier may be associated with the namespace URL if needed. All tags associated with the format are part of that namespace. The model starts with a `<gpmn:gpmn>`, which denotes the beginning and, with the corresponding closing tag, the end of a GPMN model.

Many business process and workflow language make a distinction between the semantic aspects of the model, which defines the properties of the business process or workflow that are necessary in order to understand the execution semantics of the workflow and the visual aspects of the model which are necessary for the graphical representation of the process model.

The advantage of this approach is that the part necessary for graphical representation can be ignored by a workflow engine which loads the process model and may even be omitted to save space in files intended for execution in cases where the file is only used for enactment rather than modeling and design.

In most cases, both aspects are generally included in the same file but it can also be split up in two different files such as in an older iteration of the Eclipse

BPMN modeling tool (see [51]). However, the disadvantage of the two-file approach is that it makes it harder to ensure all necessary data is available to applications requiring them since one of the files may get lost. Furthermore, there exists an implicit reference between the two files which may be confusing to users and lead them to mistakenly delete one or the other. As a result, a newer version of the Eclipse tool (see [52]) has abandoned this approach in favor of the standardized BPMN 2.0 format, which includes both aspects of processes in the same file.

Since the approach of distinguishing the two aspects, the GPMN intermediate format also distinguishes between these two aspects of the model and separates the business model, which describes the structures relevant to the business process itself and the visual aspect, which describes visual features of the model such as the size and positioning of goal objects and plans. However, both aspects are contained in the same file.

First, the description of the business aspect of the process model is started with the `<gpmn:gpmnmodel>` tag. The first optional tag that can be defined within the `<gpmn:gpmnmodel>` tag is the `<gpmn:modelname>` tag, which wraps the name of the model as chosen by the process designer. The model name can be followed with a human-readable description of the process which is surrounded with the `<gpmn:modeldescription>` tag.

The first element of the business part of the GPMN intermediate format specifies the process context. The context section is started and closed with the `<gpmn:context>` tag. Within the context tag, context entries can be defined. This is accomplished using the `<gpmn:contextentry>` tag, which includes a string-typed `name` attribute for specifying the name of context entries as well as a `type` attribute which uses a string to specify the Java type this entry represents. Both of these attributes are mandatory and must be included to define an entry.

As an option, a `set` attribute can be included in the context entry tag in order to define the entry as a set of multiple values by setting the attribute to true. The default state for an entry is false for this attribute if it is omitted. Lastly, the tag itself can surround a Java expression which evaluates to an initial value for the entry, however, this is also optional. If no initial value is specified, the value evaluates to null.

The next section of the format describes the goals, which is started and closed with the `<gpmn:goals>` tag. This section includes all the goals in the process. Each goal is started with a `<gpmn:goal>` tag. Each goal tag includes a number of mandatory and optional XML attributes defining properties of the specific goal and also specifies which kind of goal the entry represents (see also section 4.2.3):

- The first attribute is the `id` attribute, which is a unique identifier expressed as a string value. This value identifies the goal as an object within the process and is used to reference the goal in other parts of the process.

- The `name` attribute sets the name of the goal. This attribute has no semantic effect but provides a human-readable name for the goal identifying its purpose.

- A `goalkind` attribute is mandatory when specifying a goal, which defines the kind of the goal and its semantics. The attribute is a string value which is restricted to the values "perform" for perform goals, "achieve" for achieve goals, "query" for query goals and "maintain" for maintain goals.

- The optional `retry` attribute, which defines the retry flag state. If omitted, the state of the flag defaults to true.

- The retry delay is defined by the optional `retrydelay` attribute, it defaults to zero in cases where it is omitted which is interpreted as instantaneous retries.

- The optional `randomselection` attribute indicates whether the goal selects plans at random or in the order as they are defined in the GPMN intermediate format file. The default value for this attribute is false, meaning that the defined order is chosen.

- The `exclude` attribute, defining how plans are excluded with a default value of "when_tried". Other valid values are "when_failed", "when_succeeded" and "never".

- In order to indicate whether a goal has a sequential activation semantic, a `sequential` attribute can be set to true or false, with false being the default value.

- Finally, the optional `posttoall` flag indicates whether plans attached to the goal are tried sequentially or are executed in parallel depending on the conditions. This flag has a default value of false, setting the sequential plan activation as the default.

As previously described in section 4.2.3, goal behavior can be further shaped using conditions. These can be described in the GPMN intermediate format as optional tags within the goal tag. Although not all conditions are valid for all goal kinds, the ones that can be applied to all goal kinds can be defined within a goal tag using one of the following optional tags:

- The goal creation condition as previously described can be specified by adding a `<gpmn:creationcondition>` tag within the goal tag. The tag contains an expression as a string which defines the condition when the goal should be activated independently from the activation edges.

- The condition defining when to drop an activated goal is defined using the `<gpmn:dropcondition>` tag, the condition itself again specified as a string within the tag.

- Finally, the context condition can be specified the same way as the previous conditions using the `<gpmn:contextcondition>` tag.

Some goal kinds allow for additional conditions to be defined. For example, both the achieve goals and the maintain goal allow the specification of a target condition using a `<gpmn:targetcondition>` tag. The maintain goal kind also allows the definition of a maintain condition using the `<gpmn:maintaincondition>` tag.

Parameters for goal instances can be declared by including a goal parameter section with the goal using the `<gpmn:goalparameters>` tag. Within that section, individual parameters are specified with `<gpmn:parameter>` tags. The parameter tag, similar to the context entries, must include a `name` and `type` attribute for the name and type of the parameter respectively and can optionally include an expression within the parameter tag to define an initial value.

The next section after the definition of the goals is the section defining the subprocesses. It is surrounded with the `<gpmn:subprocesses>` tag, with each subprocess defined in the process being declared using the tag `<gpmn:subprocess>`. This tag includes a mandatory string-based `id` attribute with a unique identifier for referencing and an optional `name` attribute entering descriptive strings as names for the subprocess. The subprocess itself is referenced within the subprocess tag as a string with a suitable path to the subprocess.

Following the subprocess section is the plan section, which is started and finished with the `<gpmn:plans>` tag. Within this plan section, individual plans are defined with `<gpmn:plan>` tags, which, like the goals, each include a mandatory `id` attribute with a string-based unique identifier which is used as a handle for the plan to be referenced in other parts of the process. An optional `name` attribute assigns a human-readable name to the plans the same way as goals are named. The plan tag itself then includes a reference to the actual plan implementation. This can either be a BPMN file or, alternatively, a Java class, the former being implicitly identified by the ".bpmn" suffix of the reference.

The segment following the plans section starts with the `<gpmn:activationedges>` tag and defines the activation edges in the process. Each activation edge is specified with a `<gpmn:activationedge>` tag, which includes two obligatory attributes, a `source` attribute setting the source and a `target` attribute for setting the target of the edge. Both attributes are strings in which the only valid values are unique identifiers of goals for the source and unique identifiers of goals or subprocesses for the target. The edge also includes an `id` attribute with its own unique identifier.

If the source of the activation edge is a sequential goal, the integer value assigned to the order attribute is used to define the activation order of the sequential goal. The integers provided in the order attributes of all activation edges with the same sequential goal source must be a total order and, as a result, identical numbers for two of such edges is invalid.

Subsequent to the activation edges is the section with the plan edges which is surrounded with the `<gpmn:planedges>` tag. Each plan edge in the process is detailed with a `<gpmn:planedge>` tag within that section. The source and the target of the plan edge are denoted identically to the activation edges with the attributes `source` and `target`, each specifying a unique identifier string. However, unlike activation edges, the source identifier must refer to a goal while the target identifier must refer to a plan. Like the activation edges, the plan edge tag also has an `id` attribute providing the edge with a unique identifier.

Both activation edges and sequence edges also allow the inclusion of a `name` attribute in order to specify a string used as a descriptive name for a specific activation or plan option.

The section on plan edges segues into the section that concerns the suppression edges, which uses the same pattern as the previous edge sections by surrounding the section with a `<gpmn:suppressionedges>` tag and detailing individual suppression edges with `<gpmn:suppressionedge>` tags. Each of the tags also have the `source` and `target` attributes for referencing the source and target objects for the edge, however, in both cases only goal references are valid for both target and source.

With these sections, the definition of the business aspects of a GPMN process are sufficiently defined to convey the semantics of the process and the `<gpmn:gpmnmodel>` tag is closed. Nevertheless, in order to provide a graphical representation of the process, some additional information is required which does not carry business semantics and is therefore not used for execution of GPMN processes. This is defined in the section following the definition of the business aspect and it is surrounded with the `<gpmn:visualmodel>` tag to denote the visual aspects of the GPMN process.

This segment is divided into sections for goals, plans and edges, surrounded with the `<gpmn:vgoals>`, `<gpmn:vplans>` and `<gpmn:vedges>` tags respectively. The goal and plan sections reference the goals and plans defined in the business section by specifying their unique identifiers in the `id` attribute of the `<gpmn:vgoal>` and `<gpmn:vplan>` tags. Both tags also include four additional double floating point attributes, an `x` attribute for the position of the object on the x-axis, a `y` attribute for the position on the y axis, a `w` attribute to define the width and an `h` attribute to define the height of the object.

Since the edges already define a source and a target in the business section and therefore already have a defined start and end point visually, this results in the visual section for the edges being optional. However, it is sometimes useful to add routing points to define a path for the edge instead of the edge being drawn straight from the source to the target. Therefore, in each of the `<gpmn:vedge>` tags, which reference their business counterpart using their unique identifier in the id attribute, can include `<gpmn:waypoint>` tags which define the intermediate waypoints for the edge using its `x` and `y` attribute to define the x- and y-position respectively.

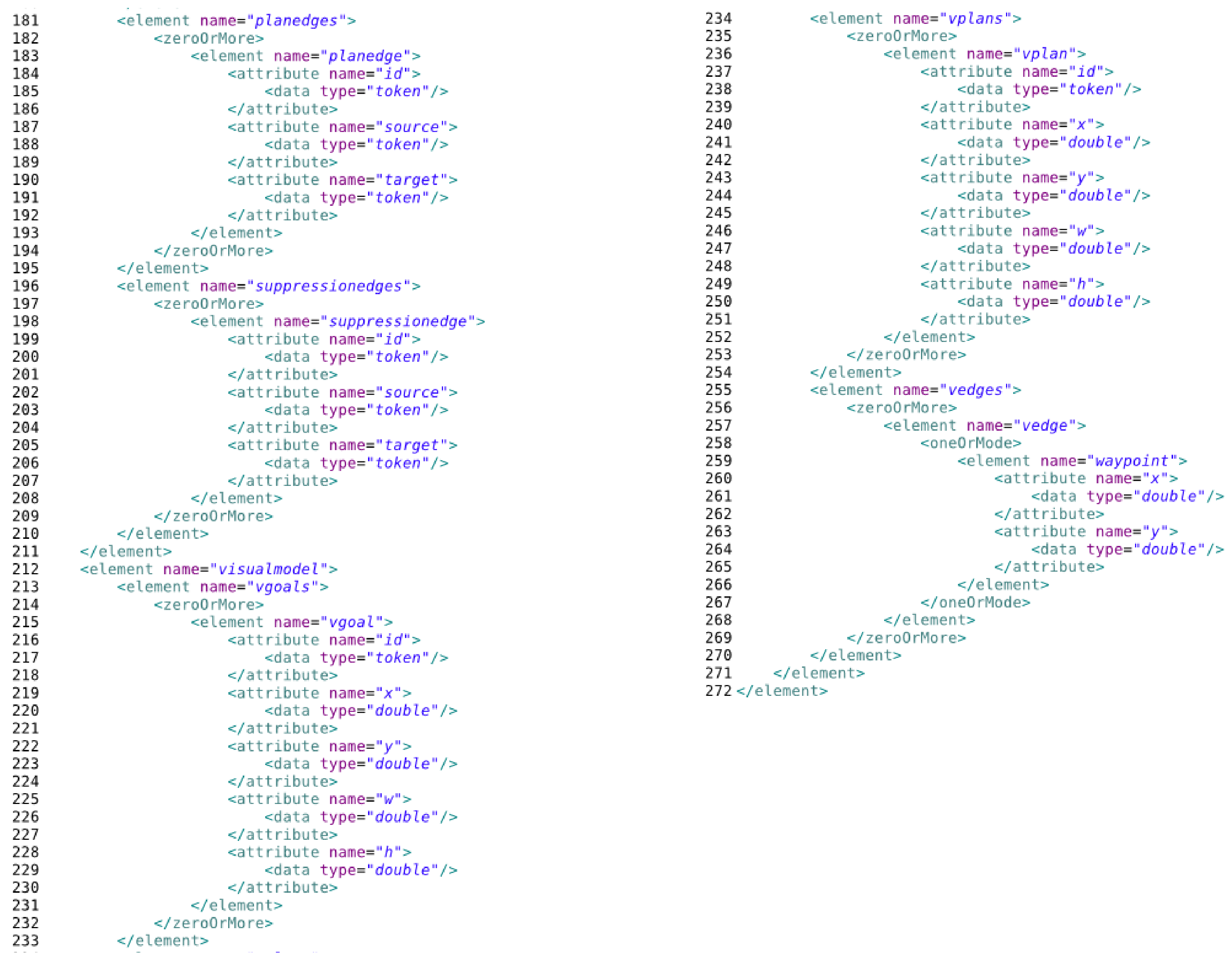
GPMN models using the GPMN intermediate format are stored as text files encoded with UTF-8 (see [169]) with the file name suffix ".gpmn2" in order to distinguis it from an earlier and non-final iteration of the format. The next section will evaluate GPMN with regard to the research goals that were set in section 1.2 and compare it with other business process and workflow modeling languages presented in earlier sections.

```
<?xml version="1.0" encoding="UTF-8"?>
<element name="gpmn" xmlns="http://relaxng.org/ns/structure/1.0"
                     datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes"
                     ns="http://jadex.sourceforge.net/gpmn">
    <element name="gpmnmodel">
        <optional>
            <element name="modelname">
                <data type="normalizedString"/>
            </element>
        </optional>
        <optional>
            <element name="modeldescription">
                <data type="string"/>
            </element>
        </optional>
        <element name="context">
            <zeroOrMore>
                <element name="contextentry">
                    <attribute name="name">
                        <data type="token"/>
                    </attribute>
                    <attribute name="type">
                        <data type="token"/>
                    </attribute>
                    <optional>
                        <attribute name="set">
                            <data type="boolean"/>
                        </attribute>
                    </optional>
                    <data type="string"/>
                </element>
            </zeroOrMore>
        </element>
        <element name="goals">
            <zeroOrMore>
                <element name="goal">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <optional>
                        <attribute name="name">
                            <data type="normalizedString"/>
                        </attribute>
                    </optional>
                    <attribute name="goalkind">
                        <data type="token"/>
                    </attribute>
                    <optional>
                        <attribute name="retry">
                            <data type="boolean"/>
                        </attribute>
                    </optional>
                    <optional>
                        <attribute name="retrydelay">
                            <data type="long"/>
                        </attribute>
                    </optional>
                    <optional>
                        <attribute name="randomselection">
                            <data type="boolean"/>
                        </attribute>
                    </optional>
                    <optional>
                        <attribute name="exclude">
                            <data type="token"/>
                        </attribute>
                    </optional>
                    <optional>
                        <attribute name="sequential">
                            <data type="boolean"/>
                        </attribute>
                    </optional>
                    <optional>
                        <attribute name="posttoall">
                            <data type="boolean"/>
                        </attribute>
                    </optional>
                    <optional>
                        <element name="creationcondition">
                            <data type="normalizedString"/>
                        </element>
                    </optional>
                    <optional>
                        <element name="dropcondition">
                            <data type="normalizedString"/>
                        </element>
                    </optional>
                    <optional>
                        <element name="contextcondition">
                            <data type="normalizedString"/>
                        </element>
                    </optional>
```

Figure 5.1: First part of the Relax NG definition of the GPMN intermediate format

```
                        <element name="targetcondition">
                            <data type="normalizedString"/>
                        </element>
                    </optional>
                    <optional>
                        <element name="maintaincondition">
                            <data type="normalizedString"/>
                        </element>
                    </optional>
                    <optional>
                        <element name="goalparameters">
                            <zeroOrMore>
                                <element name="parameter">
                                    <attribute name="name">
                                        <data type="token"/>
                                    </attribute>
                                    <attribute name="type">
                                        <data type="token"/>
                                    </attribute>
                                    <data type="string"/>
                                </element>
                            </zeroOrMore>
                        </element>
                    </optional>
                </element>
            </zeroOrMore>
        </element>
        <element name="subprocesses">
            <zeroOrMore>
                <element name="subprocess">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <optional>
                        <attribute name="name">
                            <data type="normalizedString"/>
                        </attribute>
                    </optional>
                    <data type="normalizedString"/>
                </element>
            </zeroOrMore>
        </element>
        <element name="plans">
            <zeroOrMore>
                <element name="plan">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <optional>
                        <attribute name="name">
                            <data type="normalizedString"/>
                        </attribute>
                    </optional>
                    <optional>
                        <element name="planparameters">
                            <zeroOrMore>
                                <element name="parameter">
                                    <attribute name="name">
                                        <data type="token"/>
                                    </attribute>
                                    <attribute name="type">
                                        <data type="token"/>
                                    </attribute>
                                    <data type="string"/>
                                </element>
                            </zeroOrMore>
                        </element>
                    </optional>
                    <data type="normalizedString"/>
                </element>
            </zeroOrMore>
        </element>
        <element name="activationedges">
            <zeroOrMore>
                <element name="activationedge">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <attribute name="source">
                        <data type="token"/>
                    </attribute>
                    <attribute name="target">
                        <data type="token"/>
                    </attribute>
                </element>
            </zeroOrMore>
        </element>
```

Figure 5.2: Second part of the Relax NG definition of the GPMN intermediate format



```
        <element name="planedges">
            <zeroOrMore>
                <element name="planedge">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <attribute name="source">
                        <data type="token"/>
                    </attribute>
                    <attribute name="target">
                        <data type="token"/>
                    </attribute>
                </element>
            </zeroOrMore>
        </element>
        <element name="suppressionedges">
            <zeroOrMore>
                <element name="suppressionedge">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <attribute name="source">
                        <data type="token"/>
                    </attribute>
                    <attribute name="target">
                        <data type="token"/>
                    </attribute>
                </element>
            </zeroOrMore>
        </element>
    </element>
    <element name="visualmodel">
        <element name="vgoals">
            <zeroOrMore>
                <element name="vgoal">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <attribute name="x">
                        <data type="double"/>
                    </attribute>
                    <attribute name="y">
                        <data type="double"/>
                    </attribute>
                    <attribute name="w">
                        <data type="double"/>
                    </attribute>
                    <attribute name="h">
                        <data type="double"/>
                    </attribute>
                </element>
            </zeroOrMore>
        </element>
        <element name="vplans">
            <zeroOrMore>
                <element name="vplan">
                    <attribute name="id">
                        <data type="token"/>
                    </attribute>
                    <attribute name="x">
                        <data type="double"/>
                    </attribute>
                    <attribute name="y">
                        <data type="double"/>
                    </attribute>
                    <attribute name="w">
                        <data type="double"/>
                    </attribute>
                    <attribute name="h">
                        <data type="double"/>
                    </attribute>
                </element>
            </zeroOrMore>
        </element>
        <element name="vedges">
            <zeroOrMore>
                <element name="vedge">
                    <oneOrMode>
                        <element name="waypoint">
                            <attribute name="x">
                                <data type="double"/>
                            </attribute>
                            <attribute name="y">
                                <data type="double"/>
                            </attribute>
                        </element>
                    </oneOrMode>
                </element>
            </zeroOrMore>
        </element>
    </element>
</element>
```

Figure 5.3: Final part of the Relax NG definition of the GPMN intermediate format

## 5.3 Modeling with GPMN

In section 2.7 three research goals, strategic-operational cohesion, workflow model agility and balance of local autonomy and global control were presented for the design phase of the business process management lifecycle in which the business process modeling language represents a key aspect. Strategic-operational cohesion ties the more loosely organized action in a long-running autonomous process to strategic planning. Increased local autonomy by participants allow them to retain their independence which has to be balanced against the need for global control in order to reach the defined business goals. Workflow model agility enables workflow designers to easily include contingencies for predictable changes in the business environment that can occur over the long execution time of the process.

If the language is used in the implementation phase as a workflow modeling language, two of these three research goals, the balance of global control and local autonomy as well as the workflow model agility, also affect the implementation phase.

Strategic-operational cohesion can be difficult to establish in many business process and workflow modeling languages. For example, in section 3.8 it was shown how many task-based modeling languages, while clearly showing what is being done, only implicitly show the reasons for particular actions and therefore only have a weak link to strategic planning. It is sometimes maintained through non-executable meta-models or annotations, however, this connection is not part of the execution semantics and tends to be neglected over time once the processes get reengineered.

In GPMN on the other hand, strategic planning is present in the model itself through the representation of business goals as elements of the business process model. Furthermore, in workflows designed in GPMN, a link to strategic planning is always present by evaluating the active goal instances and how they related to the root goals representing the key business goals for the process.

Sets of actions in GPMN are wrapped in plans which have a defined association with certain goals which are part of the overall goal hierarchy. Therefore, actions taken by a business process or workflow instance not only carry information about what is being done at the moment but also offer a well-defined business reason for why they are being performed at the time.

The difficulties of allowing workflow model agility and a greater amount of local autonomy for the autonomous workflow participants in long-running autonomous processes was demonstrated for task-based business process and workflow modeling language in section 3.8 by using an example of a development process. The process involved the development of an airplane and revolved around the development of three parts required for the plane as shown in the linear approach in Figure 3.25.

The workflow model agility required that when one part is designed in a way that requires an additional design review of a previously designed and finalized part. These reiterations of previously performed parts of the workflow were coined as revisits and required that a control flow path not only to the part of the workflow needing to be revisited but also a return path when the revisit has concluded.

Furthermore, increased autonomy in case of this sample development process means that the workflow participants must be able to chose the order in which the parts are designed based on their expertise. This means that not only an initial choice must be selectable after process enactment but after the finalization of each part the following part must be selectable from the remaining ones in order to allow arbitrary orders.

As was demonstrated in Figure 3.26, it is possible to design a workflow with both the required flexibility and the provision of revisits in task-based modeling languages. However, it was also shown that such processes have an excessive amount of branching. While in simple example processes such as these the inclusion of the required branching, despite being fairly confusing to non-technical users and designers of business processes, may still seem manageable, more complicated processes with more parts being involved quickly outstrips the usefulness of the language to clearly represent the business process.

In GPMN, a different approach to branching can be chosen to allow the required flexibility and autonomy as demonstrated in Figure 5.4. In this business process model, the process design is started by including the business goal of the process as the root element of the GPMN process goal hierarchy, which in case of this example is the achieve goal described as "Finish Airplane Design".

Since the goal is considered accomplished when schematics for all parts of the airplane have been developed, this naturally leads to the three subgoals of the main business goal of the process. The achieve goal "Generate Fuselage Design" is responsible for finishing the fuselage design while the "Generate Wing Design" and "Generate Engine Design" achieve goals aim to provide the design for the wings and the engine respectively.

In the original process, each of the parts had four tasks associated with them: The initial task for developing the initial design of the part followed by three tasks for completing the developed design: First, a design review is performed to find design issue with the current design of the part, then the issues found during the design review are addressed by changing the design and ultimately the part design is finalized in the last task for each part.

In the GPMN version of this process, these steps are broken into two goals. In case of the fuselage, the first goal is the "Design Fuselage" goal which aims to provide the initial design of that part similar to the task providing the initial design in the process in the task-based approach. A plan is provided for the goal in order to provide the necessary steps for generating the initial design of the part.

The design review along with the design adaption and finalization are the responsibility of the second goal, which in case of the fuselage is called "Maintain Finalized Fuselage Design". Similar to the goal "Design Fuselage", a plan is attached to perform the necessary tasks for this part of the process.

Since a design cannot be finalized before the initial design has been provided, the previous goal named "Generate Fuselage Design" is defined as a sequential goal with the outgoing activation edge going towards the initial design goal being assigned the order "1" and the activation edge going to the goal for maintaining the finalized

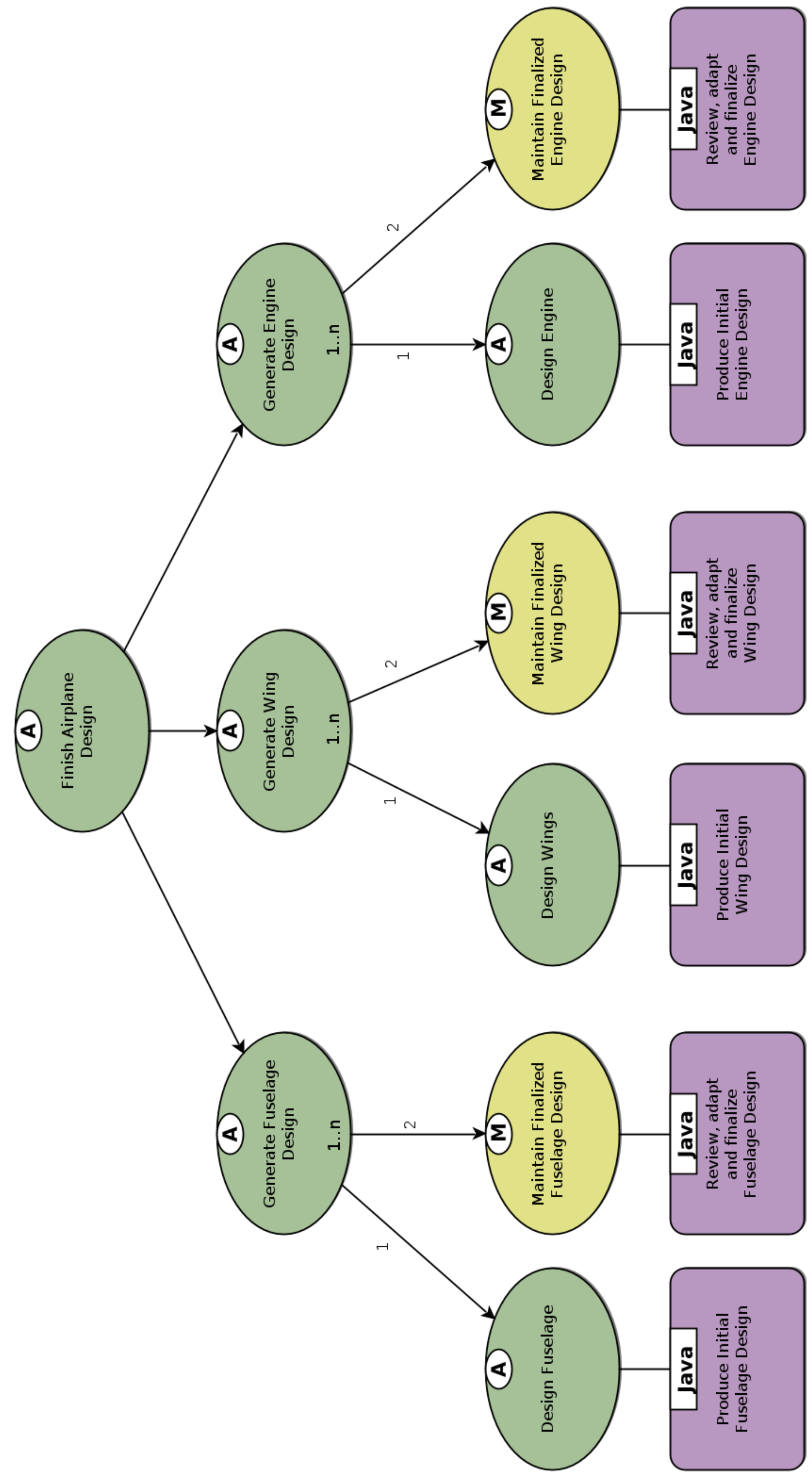

A
Finish Airplane Design
A
Generate Fuselage Design
1..n
A
Generate Wing Design
1..n
A
Generate Engine Design
1..n
1
2
1
2
1
2
A
Design Fuselage
M
Maintain Finalized Fuselage Design
A
Design Wings
M
Maintain Finalized Wing Design
A
Design Engine
M
Maintain Finalized Engine Design
Java
Produce Initial Fuselage Design
Java
Review, adapt and finalize Fuselage Design
Java
Produce Initial Wing Design
Java
Review, adapt and finalize Wing Design
Java
Produce Initial Engine Design
Java
Review, adapt and finalize Engine Design

Figure 5.4: Airplane design process modeled using GPMN

design being assigned the order ”2”. This means that the goal for maintaining the design is only activated after the goal for the initial design has been achieved (i.e. an initial design has been generated).

When the business process is enacted, the root goal is activated which in turn activates the achieve goals aiming to create the design for all three parts. Since the goals are sequential, they initiate the first goal in the sequence which, in each case, is the creation of the initial design of the part. At this point, the goals of the process is to create an initial part design and the workflow participants are able to chose which of the parts to design first.

At any point, the initial design of any other part can be initiated, allowing for parallel work on those designs which is a flexibility feature not provided even by the heavily branched task-based model shown in Figure 3.26.

Once the initial design for a part is provided through the plan, the attached achieve goal instance will have met its target condition and terminate successfully. As a result, the sequential goal that aims to generate the finalized design will proceed to the next goal in its sequential order which, in case of the fuselage, is the goal labeled ”Maintain Finalized Fuselage Design”.

This newly activated goal is a maintain goal, the maintain condition mandating the existence of a finalized design will be monitored after activation. Since the initial design has just been finished and no design review has been performed, this maintain condition is violated immediately and the goal will cause the process to proceed with the attached plan to restore the condition, which contains three elements; the design review, design adaption and finalization tasks.

However, after the maintain condition has been accomplished, unlike an achieve goal the maintain goal instance will stay active and continue to monitor the condition. This pattern of a maintain condition will allow for the revisits of the design review part of the process: If any other part of the process causes the maintain condition to be invalidated again, the goal will again use the plan with the design review process fragment to restore the finalized design.

As a result, this relatively structured and clear process design allows the same and even greater flexibility than the task-based approach with a large amount of branching and therefore demonstrate the advantages of the goal-oriented approach of GPMN in long-running autonomous processes such as the one used as the basis for the example.

The next chapter will proceed with an exposition of necessary infrastructure and tools required to turn GPMN-based business process models into executable workflows. It will further elaborate on the underlying engine for executing GPMN workflows as well as provide a background on the infrastructure supporting them.

## 5.4 Comparison of GO-BPMN and GPMN

As noted in the beginning of this chapter, both GPMN and GO-BPMN are based on the goal-context approach conceived at Daimler AG and therefore follow the same approach of using business goals as modeling elements in their business process and workflow languages. However, the approaches use different semantic bases for the modeling language: GO-BPMN is understood as a relatively conservative extension to BPMN and therefore follows a similar state machine-driven approach. In contrast, GPMN follows an approach of separating model elements from instances and uses a two-step process of goal deliberation and means-end reasoning to process the GPMN models during execution.

These two approaches result in some differences regarding the modeling capabilities and resulting execution flexibility of processes modeled in each language. This section will point out major modeling differences with a focus on the added flexibility of the GPMN approach over the more conservative interpretation used by GO-BPMN.

### 5.4.1 Goal Instantiation

In GO-BPMN workflow instances, goals which are part of the process are unitary instances that can acquire certain states such as inactive, when the goal has not been activated, active, when the goal is active, or running, when the goal is not only actively pursued but its plans are being executed. The goal itself however is not reified beyond its existence within the process model.

In contrast, the GPMN semantic requires creation of goal instances from the goal element in the process once a goal has been activated. These goal instances can be individually configured through parameters and are pursued individually in parallel with constraints being evaluated for each instance.

Both GPMN and GO-BPMN allow the manual activation and reactivation of goals within the goal hierarchy by use of library calls/tasks within the plans of the process. However, in GO-BPMN, the goal can either be active or inactive/finished while in GPMN multiple instances can be created at once.

Figure 5.5 shows an example process demonstrating how goal instances increase the usefulness of the reactivation of goals within plans. The process has a single root goal of preparing a pizza which is divided into two subgoals, one for acquiring and preparing a topping and another for assembling the final product.

The branch that prepares an ingredient can be configured with a parameter but when no parameter is set, defaults to preparing the dough. The rest of the branch is divided into two steps, one for ingredient retrieval either from storage or by supermarket purchase and the second step of preparing the ingredient by opening its container and optionally using the blender.

After the initial ingredient (the dough) has been prepared, the process continues with the second step by activating the goal "Assemble and Finish Pizza". This part is also split into two steps. In the goal activated first, the toppings for the pizza are assembled while the second goal aims to finish the pizza by baking it.

The plan to assemble toppings for the pizza can now reuse the subtree of the goal hierarchy that deals with ingredient preparation through the use of goal instances: The plan reactivates the "Acquire Prepared Ingredients" goal for each topping but

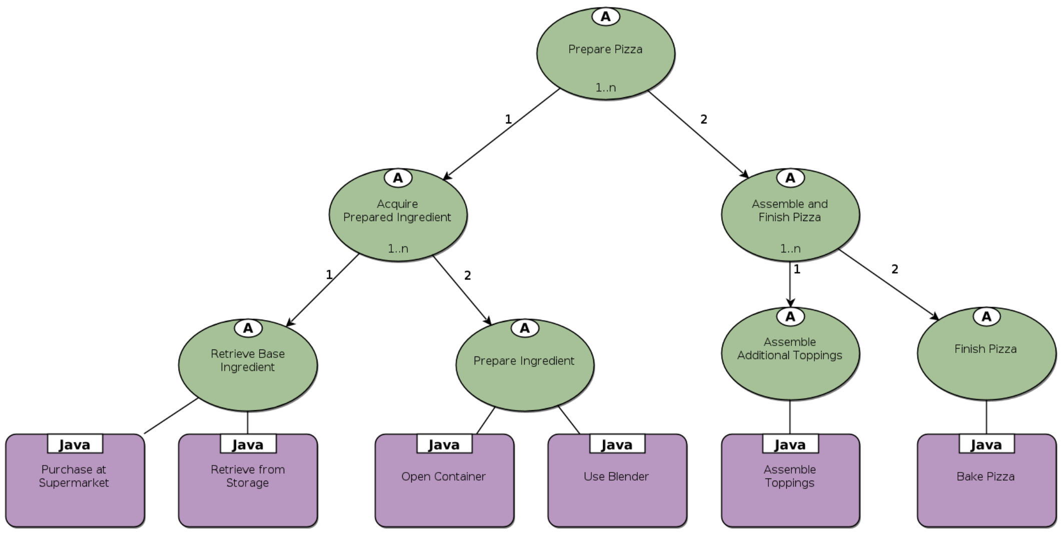


Figure 5.5: Example of a process that employs the use of goal instances by reactivating the goal branch starting with the "Acquire Prepared Toppings" goal

configures each instance to acquire and prepare a specific topping instead of defaulting to the goal. These goals are then pursued in parallel, making use of the available plans as necessary until all toppings are assembled and prepared. The final pizza is then baked as part of the second step and the process terminates.

This process demonstrates how goal instances can be employed to reuse parts of the goal hierarchy in different capacities to their default, adding additional modeling flexibility to GPMN.

### 5.4.2 Allowing Goal Subtrees as Plan Alternatives

In GO-BPMN, plans that can be used to fulfill goals can be attached to leaf goals of the goal hierarchy. This approach covers the common case found when decomposing the main business goals and eventually arriving at subgoals with a very small scope as leaf goals. However, GPMN also allows the attachments of plans to goals other than leaf goals as shorthand alternatives to fulfilling a goal if a condition on the plan allows for it.

Figure 5.6 demonstrates how this added modeling flexibility can contribute more concise process models. The figure shows a goal hierarchy which could either be used stand-alone or as part of a larger goal hierarchy. The root goal of the hierarchy is the "Provide Components" goal, which expresses that the process aims to make certain components available to the rest of the process.

The shown hierarchy offers two possibilities to achieve the goal. First, a plan for simply retrieving the components from the warehouse is available. However, this plan is guarded by a condition which requires that components be stocked in the warehouse.

The alternative is a goal subtree that will attempt to purchase the components, which again has the subgoals of first retrieving offers from suppliers, then ordering

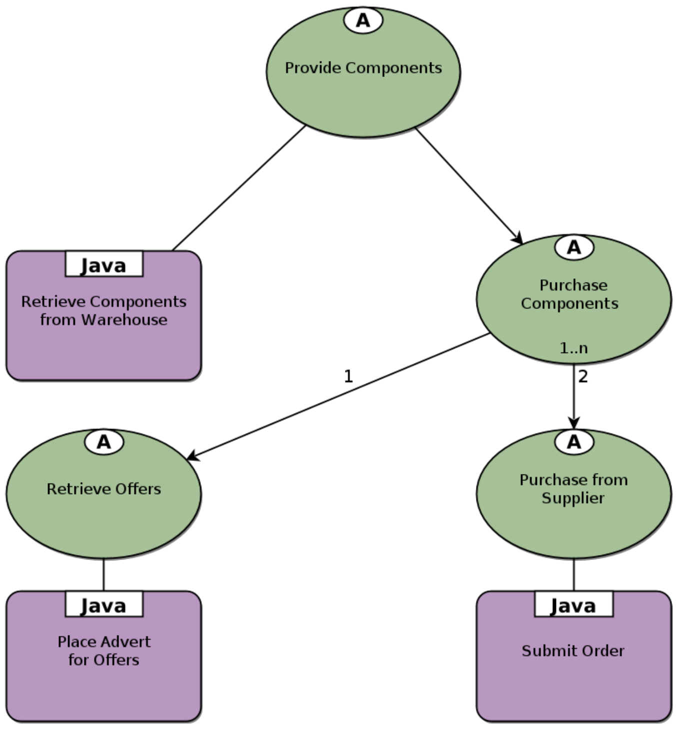


Figure 5.6: GPMN allows the attachment of plans to goals which are not leaf goals, allowing for additional model flexibility

the components based on the best offer received. The ”Provide Components” goal therefore has both a plan attached with a plan edge and a subgoal with an activation edge.

The workflow process treats either one as an option for achieving the main goal, so if the option to retrieve the components from the warehouse is inhibited by the precondition that components are unavailable in the warehouse, the process will activate the ”Purchase Components” goal instead. This allows the workflow designer to mix and match plans and subgoal hierarchies as required by the workflow.

### 5.4.3 Goal Deliberation with Suppression Edges

Goals sometimes stand in conflict with each other or an issue needs to be resolve before other goals can be resumed. For this type of issue when modeling goal-oriented business processes or workflows, GPMN offers a special visual modeling element called suppression edges to both semantically and graphically model precedences between goals.

Figure 5.7 demonstrates the use of suppression edges in a simple example business process. The main business process consists of the main goal of constructing a shopping mall which is divided into three subgoals: An achieve goal of constructing the parking lot, an achieve goal of constructing the main building and a maintain goal to ensure the workers a safe environment. Each of the goals have a plan attached which works towards accomplishing the attached goal or restoring the maintain condition

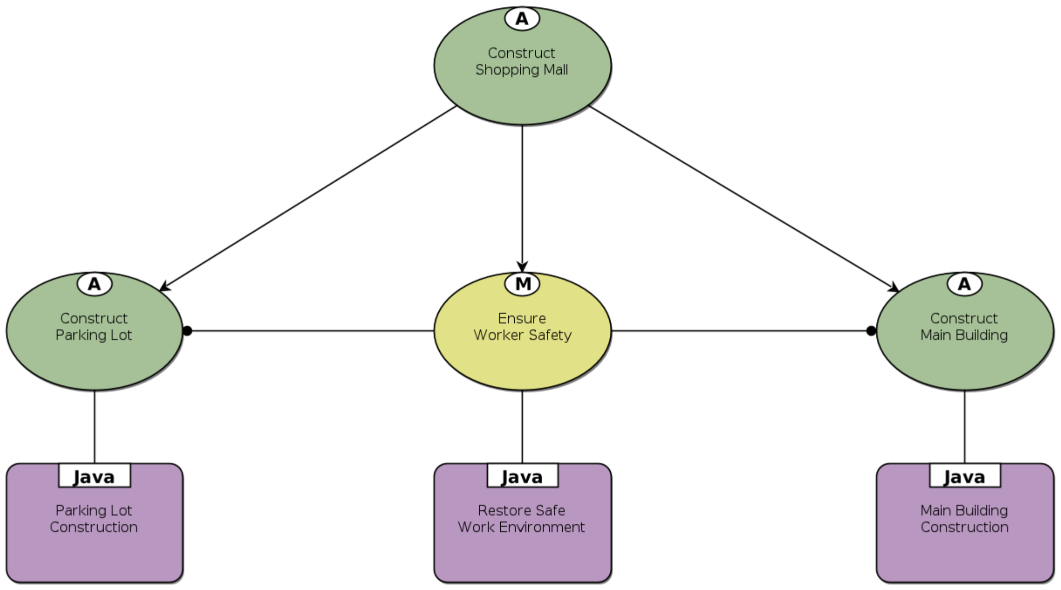


Figure 5.7: GPMN offers Suppression edges to temporarily suppress active goals while a potentially conflicting goal takes precedence

on the maintain goal.

If a safety issue arises, the maintain condition of the maintain goal becomes invalid and the goal will now be actively pursued in order to restore a safe working environment. However, while a safe work environment is being restored, work cannot continue without risking the health of the construction workers.

This is where the suppression edge can assist in modeling such situations: Once the maintain goal is actively pursued, the suppression edges ensure that the two achieve goals aimed at the construction work are suppressed. This means that their plan execution is interrupted while a safe work environment is being restored.

While it is possible to model similar patterns in GO-BPMN solely using the GO-BPMN goal preconditions, it has a number of disadvantages to the use of suppression edges:

- Suppression edges are an explicit and graphical element that clearly denote relationships between goals even to non-technical people involved in the process. Conditions on the other hand are implicitly attached and have to be formulated by a specialist.

- Goals in GO-BPMN only have preconditions and once committed to the execution state cannot be prevented from finishing plan execution. This allows the initial establishment of a safe work environment but not work stoppage and recovery once construction has begun. Suppression edges on the other hand allow the temporary suppression of already actively pursued goals when it becomes necessary (see also section 5.4.4).

- Suppression edges can be effectively combined with specialized goal kinds like the maintain goal, which only suppresses the connected goals on violations of the maintain condition.

As a result, while it is possible to devise a clever use of explicit context variables, conditions and goal deactivations to achieve a similar sort of effect in GO-BPMN, it is a technically involved solution which results in the solution being incomprehensive to non-technical people. Furthermore, the process context is supposed to represent the business situation rather than being used as a technical coordination mechanism between goals and plans. Finally, such a solution essentially disables the default execution semantics of GO-BPMN and awkwardly replaces it with a custom implemented semantics.

The next section will discuss the advantages of GPMN's explicit goal deliberation and means-end reasoning and demonstrates how this helps developing business process models.

### 5.4.4 Continuous Goal Deliberation and Means-End Reasoning

GO-BPMN, as a conservative extension of BPMN, follows the task-centered execution semantics of BPMN, where both goals and plans are treated as a special kind of BPMN element to be executed. This means once the engine has committed to "executing" a goal, this decision is final and cannot be reverted. This is the reason why goals in GO-BPMN use preconditions instead of context conditions used by GPMN: Before a goal is executed, the precondition is checked. Once the precondition is cleared, the goal becomes active and is pursued until dropped.

This behavior is in contrast to the behavior in GPMN in which context conditions are used on goals. The context condition itself does not prevent the creation of a goal instance and therefore the goal immediately reifies as a goal instance and denotes that the process now has the desire for a certain state, but may be prohibited from pursuing it due to a context condition. Once the context condition becomes valid, the goal can be actively pursued. However, should the situation change, the context condition can become invalid again and the goal will again be temporarily suspended.

With plans, GPMN offers both kinds of conditions, the precondition which can exclude plans from the initial plan options and the context condition which can cause an active plan to be aborted, with an optional plan rollback being performed. As a result, the condition and context handling is somewhat more dynamic in GPMN when compared to GO-BPMN.

The next section will explain the additional goal kinds available in GPMN compared to GO-BPMN and present some situations where they might be useful and assist in workflow modeling.

### 5.4.5 Additional Goal Kinds

Both GPMN and GO-BPMN offer the achieve goal as a goal kind that denotes the desire to reach a particular state as well as the maintain goal which is used to express the desire to not only reach but maintain a business state over an extended period of time. However, GPMN offers two additional goal kinds which are useful for certain business situations:

- The perform goal, which denotes the simple desire to perform an action regardless of outcome. This goal is usually used in situation where the action is regularly and repeatedly performed during the process.
- The query goal, which expresses the desire to acquire certain information if the information is not already available.

While both goal kinds can be emulated in GO-BPMN using the achieve goal kind, it requires the use of the business context to implement the additional functionality of these goal kinds, thus introducing technical data into the business context which is supposed to contain only business information.

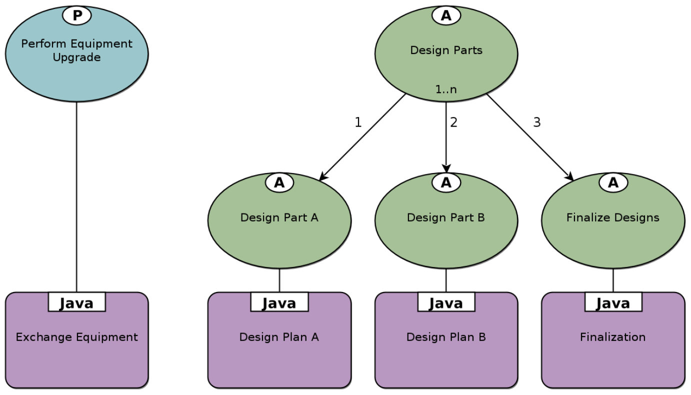


Figure 5.8: Example process employing a perform goal

Figure 5.8 shows a simple example process which demonstrates the use of the perform goal kind. The process consists of two hierarchies, one hierarchy which aims at producing certain parts using a set of equipment.

The hierarchy for designing parts consists of a sequential achieve goal hierarchy, with a common main goal aiming to produce parts and a sequential set of subgoals for designing each part and finalizing their designs. However, since this represents a long-running process, the equipment used to design each part may need to be upgraded at regular intervals to maintain the state of the art.

The other hierarchy consists only of a single perform goal which is responsible for these regular equipment upgrades which uses an upgrade plan attached to the goal to perform the upgrades. The goal has the retry flag enabled and is configured with a specific retry delay, causing the workflow engine to retry the plan execution at the intervals defined by the retry delay. Unlike the achieve goal, however, the perform goal has no state it aims to achieve.

If an achieve goal were used, the achieve goal is terminated once the achieve condition is met, which means in practice that the attached plan is only retried once since its successful execution denotes success for achieve goals without a specific achieve condition. However, the perform goal has no achieve condition, which results in this configuration running the upgrade plan regularly due to the retry flag.

The use of the query goal kind on the other hand is demonstrate in the example shown in Figure 5.9. This sample business process concerns itself with the fulfillment of a customer order, where the placement of an order is confirmed to the customer, the order is shipped and finally a confirmation of the order shipment is also sent to the customer.

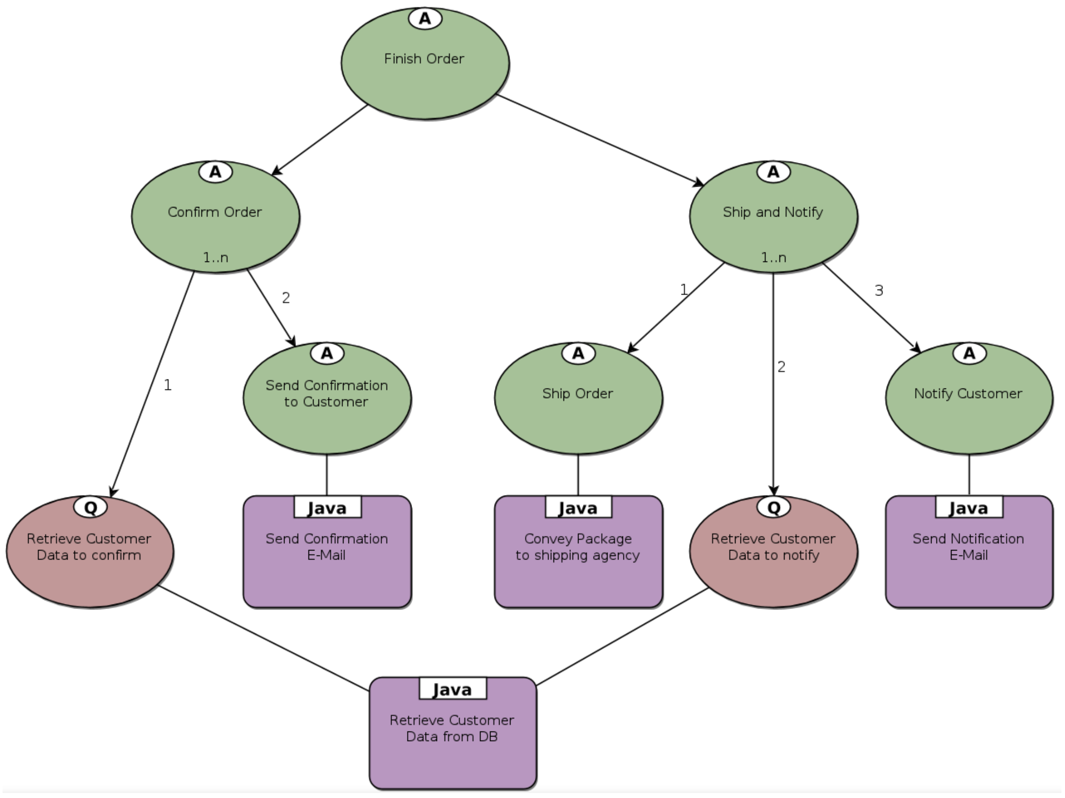

A
Finish Order
A
Confirm Order
1..n
A
Ship and Notify
1..n
1
2
1
2
3
A
Send Confirmation to Customer
A
Ship Order
A
Notify Customer
Q
Retrieve Customer Data to confirm
Java
Send Confirmation E-Mail
Java
Convey Package to shipping agency
Q
Retrieve Customer Data to notify
Java
Send Notification E-Mail
Java
Retrieve Customer Data from DB

Figure 5.9: Example process employing a query goal

The goal hierarchy is split in two branches, one branch to confirm the order to the customer and the other one to ship the order and notify the customer about the shipment. Before the customer can be sent the confirmation, however, pertinent information about the customer such as name and address has to be available. For this purpose, the query goal kind is used: Before the confirmation e-mail is send, the query goal ensures that the customer information is available.

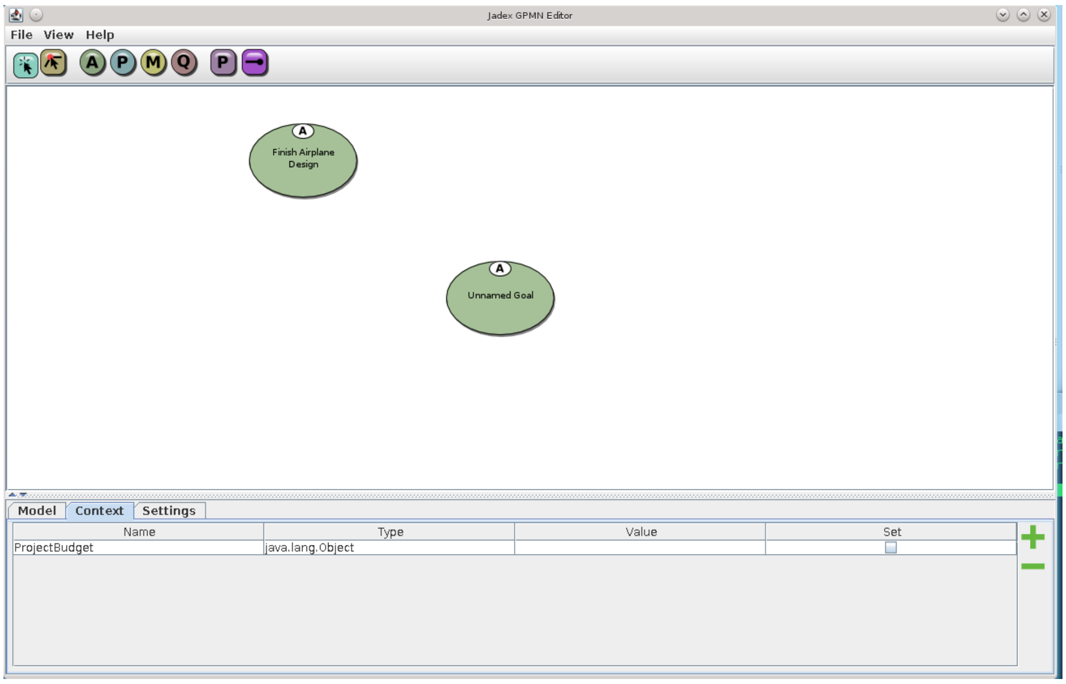
Another query goal is used before the shipment notification is send in order to ensure the availability of the customer information. In fact, both goals share the same plan to retrieve this information.

However, the advantage of using the query goal is that it only performs a plan if the information to be queried is not already available. This means that it does not matter which of the query goals becomes active first since the other one will terminate successfully if the information is available without issuing an additional database request.

GPMN therefore offers some additional business process modeling flexibility and expressiveness in certain situation by offering these additional goal kinds. The next section will introduce the modeling tools which are used to model both the GPMN business processes themselves as well as the BPMN fragments that can be used as GPMN plans.

## 5.5 Implementation of a GPMN-based Editor Toolset

While it is possible to use business process languages without IT systems by drawing or specifying business processes on paper, they are typical designed using specific modeling applications, often as part of a business process or workflow management system, which supports designing models graphically. This becomes especially relevant if the model should be enhanced with execution detail to become an execute workflow model.



| Name | Type | Value | Set |
|---|---|---|---|
| ProjectBudget | java.lang.Object | | ☐ |

Figure 5.10: Screenshot of the GPMN Editor which allows a workflow designer to model GPMN business processes and workflows

If the resulting business model is eventually converted to a workflow model, the need for such a tool becomes clear since it can automatically generate a workflow model in a machine-readable format which can be used by a suitable workflow engine or interpreter.

As a result, two editors for both GPMN and BPMN have been developed to support modeling the goal-oriented process themselves as well as the BPMN fragments that implement the GPMN plans (see [81]).

The GPMN editor as shown in Figure 5.10 provides a business process or workflow designer with the tooling to develop GPMN process models. The user interface is separated into three parts: The top area contains a toolbar where GPMN design elements like the four goal types can be selected. It also contains a tool for selecting

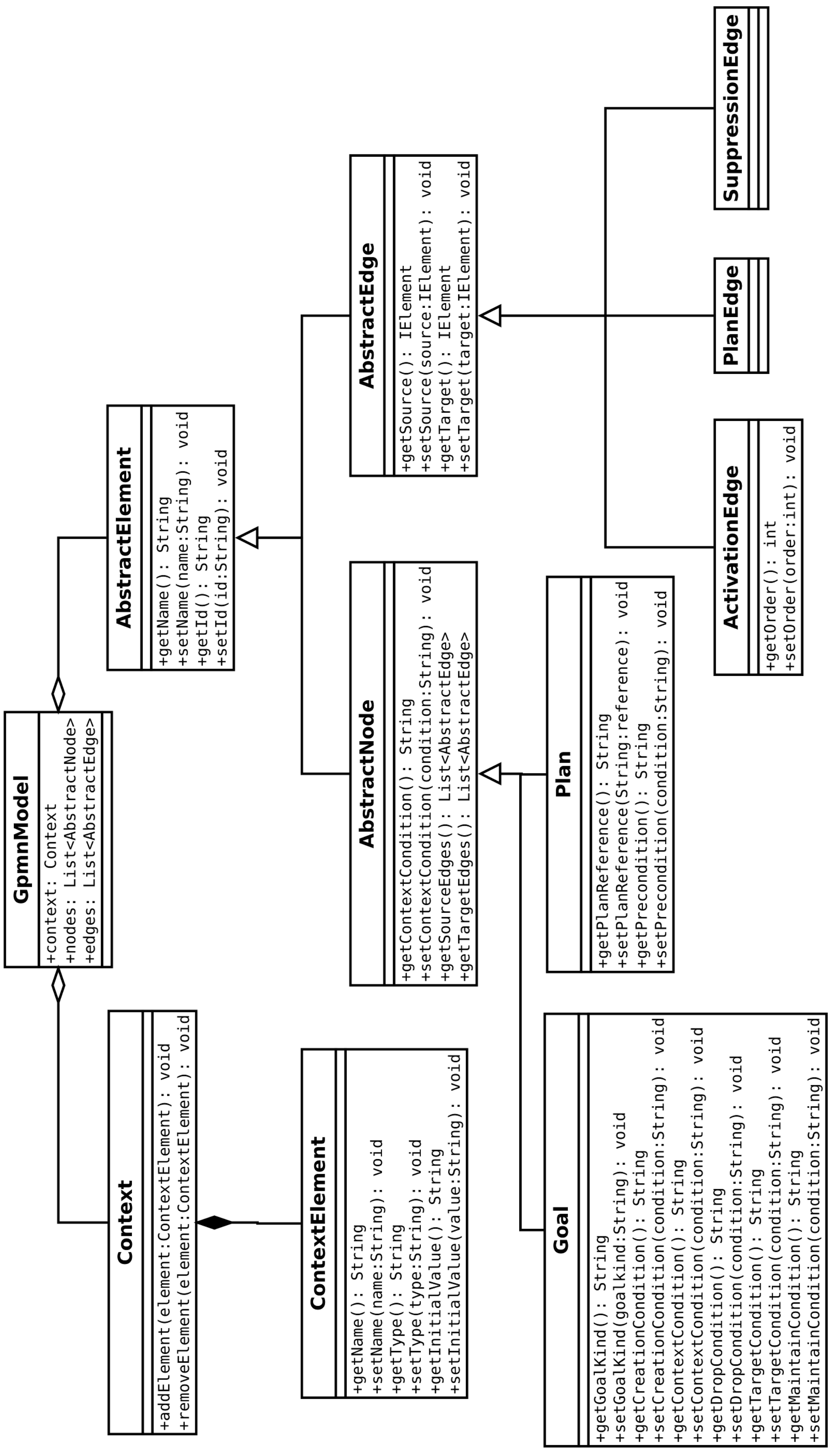

GpmnModel
+context: Context
+nodes: List<AbstractNode>
+edges: List<AbstractEdge>
AbstractElement
+getName(): String
+setName(name:String): void
+getId(): String
+setId(id:String): void
Context
+addElement(element:ContextElement): void
+removeElement(element:ContextElement): void
ContextElement
+getName(): String
+setName(name:String): void
+getType(): String
+setType(type:String): void
+getInitialValue(): String
+setInitialValue(value:String): void
AbstractNode
+getContextCondition(): String
+setContextCondition(condition:String): void
+getSourceEdges(): List<AbstractEdge>
+getTargetEdges(): List<AbstractEdge>
AbstractEdge
+getSource(): IElement
+setSource(source:IElement): void
+getTarget(): IElement
+setTarget(target:IElement): void
Plan
+getPlanReference(): String
+setPlanReference(String:reference): void
+getPrecondition(): String
+setPrecondition(condition:String): void
Goal
+getGoalKind(): String
+setGoalKind(goalkind:String): void
+getCreationCondition(): String
+setCreationCondition(condition:String): void
+getContextCondition(): String
+setContextCondition(condition:String): void
+getDropCondition(): String
+setDropCondition(condition:String): void
+getTargetCondition(): String
+setTargetCondition(condition:String): void
+getMaintainCondition(): String
+setMaintainCondition(condition:String): void
ActivationEdge
+getOrder(): int
+setOrder(order:int): void
PlanEdge
SuppressionEdge

Figure 5.11: The business side class hierarchy as used by the GPMN editor

GPMN elements and a tool for adding waypoints to edges.

The central area contains the GPMN goal hierarchy and plans. This can be used to add and delete GPMN elements and connect them with edges. Edge connections can be accomplished by simply dragging a line between two elements. The editor will then select the right type of edge to be inserted between the elements. For example, if a line is dragged between two goals, an activation edge is inserted while dragging a line between goals and plans inserts plan edges.

Since suppression edges are also inserted between goals, a separate tool is available in the toolbar which, when selected, causes the editor to draw suppression edges instead of activation edges between goals.

Like the GPMN intermediate and BPMN 2.0 formats, the GPMN editor also distinguishes between the visual and business aspects of process models with the business side of the model representing the business semantics of the workflow and the visual side containing the representation of the graphical elements such as sizes and positions.

Each graphical element in a GPMN workflow is therefore represented by a visual and a business object with the visual one holding a reference to their business aspect counterpart. However, both models represent graphs and therefore contain both nodes and edges. Since many objects on the business side of the model contain similar information, the business aspect of the model uses a class hierarchy to minimize implementation effort as shown in Figure 5.11.

The business side of the model is represented by the GpmnModel class, which contains all of the semantic elements of a GPMN business process or workflow. The elements are sorted into a three containers, the context containing the GPMN process context and two lists, one containing the nodes of the model and another containing the edges.

The root of the hierarchy for both the node and edge elements is the AbstractElement class, which denotes any GPMN element which can be added to the GPMN model aside from context. The class itself is abstract and therefore cannot be used to create objects by itself. However, it contains the common information held by both edges and nodes. which both are assigned a unique identifier and a name.

The abstract class AbstractNode represents a node in the model. It holds containers for all incoming and outgoing edges to the particular node, which simplifies tasks like identifying the root nodes of the goal hierarchy. In addition, both goals and plans contain a context condition. While the semantics of both are slightly different, the actual type and value range (Java expressions) are identical. Therefore, in order to reduce code duplication, the AbstractNode class also manages the context condition for both plans and goals.

Plans are represented by the Plan class, which inherits the functionality of both the AbstractNode and AbstractElement classes. It enhances them by adding plan-specific functionality. This includes setting a precondition for the plan, a feature which is not available in goals. It also adds a plan references which contains the path to the implementation of the plan such as a BPMN fragment model or a Java class.

While GPMN currently supports four different goal kinds, the current implementation of the model represents all of them using the same Goal class. This was done since the goals share most of their properties. For example, all goals contain creation, context and drop conditions, which the goal class manages. The only specific conditions are the target condition and the maintain condition, the former of which is used by the maintain and achieve goal kind while the latter is used by the maintain goal kind only.

The current approach is to allow the Goal class to contain all possible conditions and setting them to a null reference in goal kinds where they do not apply. In order to allow the distinction between goal kinds, the Goal class simply carries the information about its kind as data within the class. This also allows relatively quick conversions of goal kinds even after a goal has been added to the model, however, if more goal kinds with greater distinctions are added at a later point, differentiating between goals based on classes with all of them inheriting the basic conditions from an AbstractGoal class may be warranted.

The edges in GPMN are considered to be directed and therefore contain a well-defined source and target. Therefore, the base class AbstractEdge which is used to represent all of the edges in the model contains references to both the source of the edge, which is the node where the edge originates, and the target, which is a refernce to the node where the edge ends.

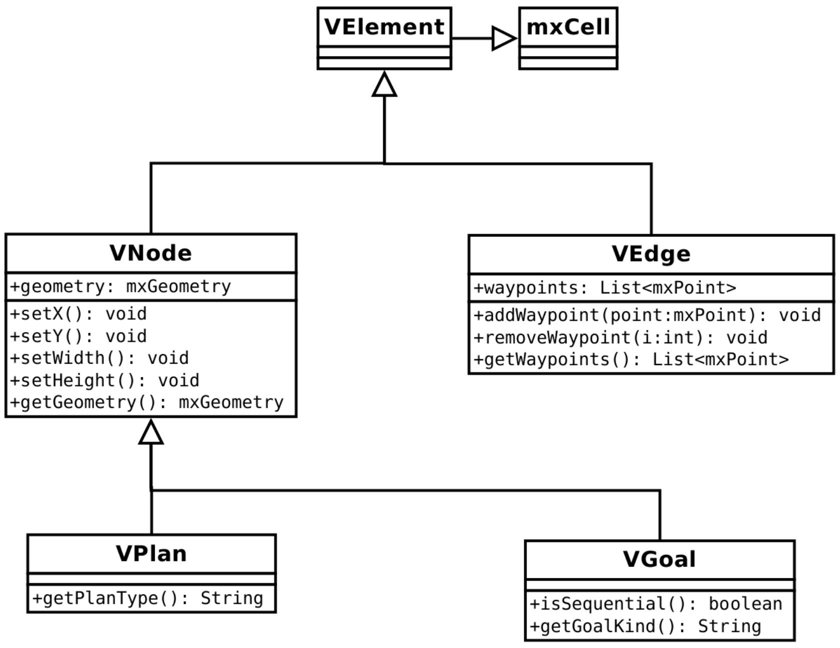


Figure 5.12: Class hierarchy of the GPMN visual model

The AbstractEdge class is then further differentiated in the subclasses ActivationEdge, PlanEdge and SuppressionEdge which differentiate the type of the edge. Additionally, the activation edge also contains information about the order in case the source of the edge is a sequential goal.

The last part of the business side of the model is the GPMN context, which is represented by the context class. Unlike the previous element, the GPMN context does not have a visual representation and therefore no element that corresponds to it in the visual model. Only one context is included in each GPMN model, which in turn is based on a list of elements that are modeled with the ContextElement class. The ContextElement class then holds the information about each entry in the context, such as name of the entry, its type and potentially an expression that produces an initial value for the context entry.

The graphical side of the editor is based on a graphical modeling framework called JGraphX (see [103]), which help to display and edit graph-based data. As a result, the visual model of the editor uses the basic graphical element class of the framework called mxCell as the base class for its hierarchy (see Figure 5.12). The base class for the visual elements is the VElement class, which primary includes initialization code that configures the base mxClass element with the settings common to all visual elements used in GPMN.

Edges are modeled in the visual part of the model using the VEdge class regardless of the type of underlying edge. This is due to the fact that the visual part of edges centers around the optional set of waypoints a model designer can add to them which is identical in function in all edge types.

In contrast, the nodes are first modeled using the abstract VNode class, which contains the common information about both goals and plans like the location in the model using x- and y-coordinates as well as a representation of the size of the object using width and height. Both pieces of data are stored in an object of the mxGeometry type which can be used directly in the JGraphX framework to display the node.

However, in order to represent both goals and plans, separate subclasses of the VNode class are available. While they do not store addition information over the VNode class itself, they offer functionality to quickly access business information about the object that is necessary to draw the node without having to acquire the underlying business object (e.g. objects of the Goal or Plan class) first.

For the VPlan class this includes the plan type, which refers to the option of including BPMN fragments and Java-based classes as implementations and is required to display the terms "BPMN" or "Java" in the graphical representation of the object.

The VGoal on the other hand includes accessor methods for acquiring the goal type to display the correct letter in the top oval of the goal and to chose the right coloring scheme for the graphical goal as well. Additionally, the VGoal class can also determine if the underlying business model aspect declares the goal it represents as a sequential goal. If true, it results in the sequential goal marker being drawn in the graphical representation.

The GPMN intermediate format presented in section 5.2 is the default storage format for models designed in the editor. The loading and saving of models is implemented using an extensible system as shown in Figure 5.13. Since the editor

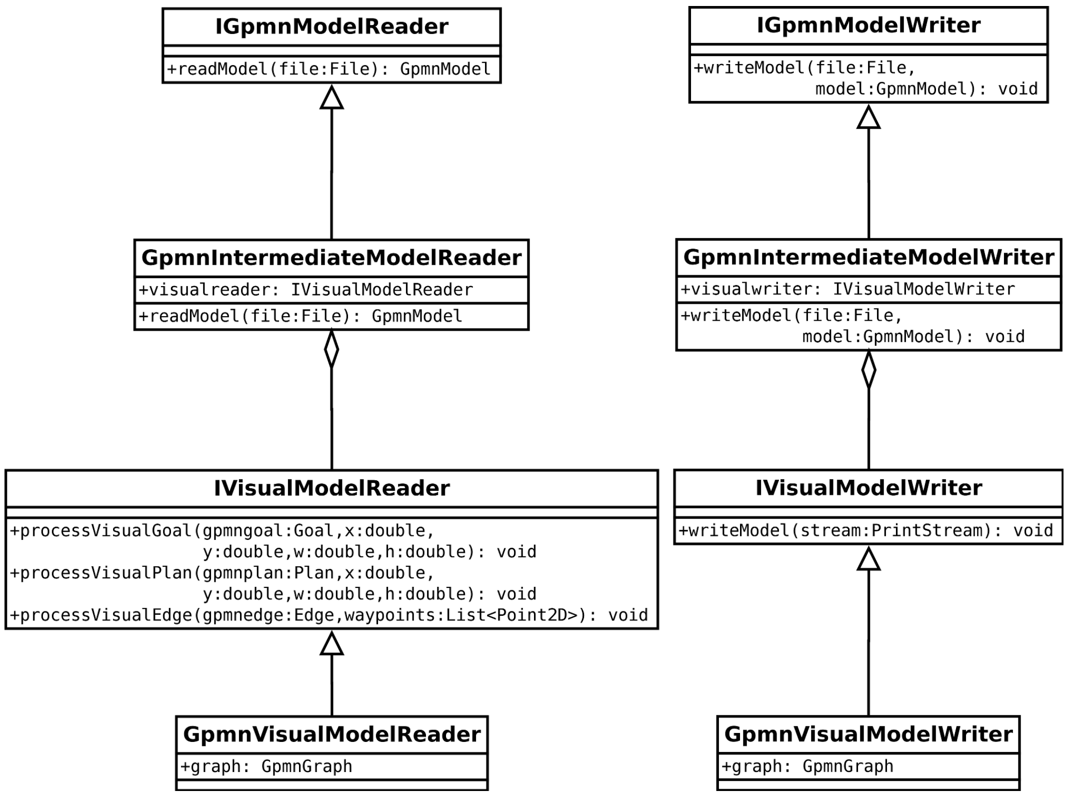

IGpmnModelReader
+readModel(file:File): GpmnModel
IGpmnModelWriter
+writeModel(file:File, model:GpmnModel): void
GpmnIntermediateModelReader
+visualreader: IVisualModelReader
+readModel(file:File): GpmnModel
GpmnIntermediateModelWriter
+visualwriter: IVisualModelWriter
+writeModel(file:File, model:GpmnModel): void
IVisualModelReader
+processVisualGoal(gpmngoal:Goal,x:double, y:double,w:double,h:double): void
+processVisualPlan(gpmnplan:Plan,x:double, y:double,w:double,h:double): void
+processVisualEdge(gpmnedge:Edge,waypoints:List<Point2D>): void
IVisualModelWriter
+writeModel(stream:PrintStream): void
GpmnVisualModelReader
+graph: GpmnGraph
GpmnVisualModelWriter
+graph: GpmnGraph

Figure 5.13: Extensible input/output system for the editor to save models in the GPMN intermediate format (see section 5.2)

may be required to store and load models in different formats, the editor only deals with the interfaces IGpmnModelReader and IGpmnModelWriter for reading and writing GPMN models respectively. This approach allows the implementation of different storage formats by simply providing a reader and writer that implement those interfaces.

The reader supporting the GPMN intermediate format is implemented as Gpmn-IntermediateModelReader, while the corresponding writer implementation is called GpmnIntermediateModelWriter.

While reading the visual part of the model is necessary for displaying the model in the editor, reading and writing this part is optional with regard to the reading and writing part since a visual part may not be necessary or even available in alternative model formats. As a result, the responsibility for reading and writing the visual part of the model is moved to a separate reader and writer which are represented in the main reader and writer using their respective interfaces IVisualModelReader and IVisualModelWriter.

When reading and writing the GPMN intermediate format in the context of the editor, the implementations used for reading and writing the visual part of the model are GpmnVisualModelReader and GpmnVisualModelWriter which implement the interfaces mentioned above.

The visual readers and writers must be created and assigned to the base model readers and writers if the visual part of the model should be considered which is then used to assign the field available in both the GpmnIntermediateModelReader and GpmnIntermediateModelWriter for the visual readers and writers. If this field is not set, it defaults to the null reference and the visual part of the model is ignored.

This approach also allows the use of alternative readers and writers for the visual part of the model. The current implementations are based on the visual approach of the JGraphX framework, however, if an alternative graphical framework is used in a different tool, an appropriate visual reader and writer can be supplied while still using the standard reader and writer for the business part of the model.

Since the visual part of the model is simply appended to the format, the interface for the visual writer is relatively simple, containing only the method writeModel() which expects the opened stream of the target that is being written which usually consists of a file. The visual writer is then expected to insert the visual aspects of the model into the target using the stream. The main writer finishes writing the model by adding the final tags for the overall model, then closing the stream.

This simple interface is in contrast to the interface of the visual reader. Here, the main reader will assist in parsing the file and invoking one of the available methods in the visual reader whenever it encounters the description of such a visual element.

The two methods involving nodes in the graph are processVisualGoal() for processing the visual aspects of a goal element and processVisualPlan() for visual aspects of plan elements. Both receive the corresponding business-side object of the Goal and Plan type respectively and the position and size of the node.

The processVisualEdge() method is responsible generating the visual edges within the model. It also receives the business-side object of the type Edge but has to differentiate between the different type of edges available. Additionally, it optionally receives a list of waypoints which are used to route the visual edge, overriding the default routing behavior.

The implementations of the visual readers and writers, GpmnVisualModelReader and GpmnVisualModelWriter, include a reference to an object of the GpmnGraph type. This class is a subclass of the mxGraph class from the JGraphX framework, which is the base class for modeling visual graphs within the framework. When the visual part of the model is read, the generated visual aspects are directly inserted into that graph. The writer on the other hand uses this object to extract the visual information written into the model.

If a GPMN process model uses BPMN fragments to implement plans, it also requires a tool for modeling such BPMN fragments. The BPMN editor (see Figure 5.14) is structured in a similar fashion to the GPMN editor, separating business and visual layers of the process model. The implementation currently supports most of the BPMN elements with defined execution semantics.

Unlike other available BPMN editors such as offered by Eclipse (see [52]) and Activiti (see [4]), the editor supports extensions that help the models integrate in the runtime environment used by GPMN and enable additional functionality offered by the BPMN interpreter offered by that environment (see chapter 6).

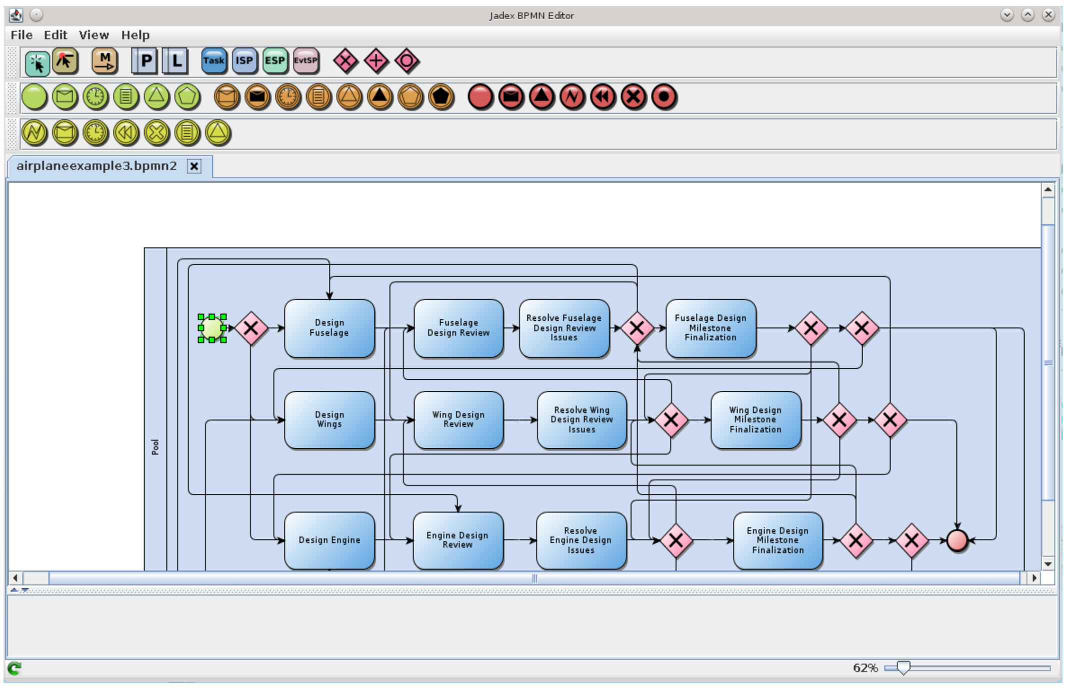

Jadex BPMN Editor
File Edit View Help
Task ISP ESP EvtSP
airplaneexample3.bpmn2
Pool
Design Fuselage
Fuselage Design Review
Resolve Fuselage Design Review Issues
Fuselage Design Milestone Finalization
Design Wings
Wing Design Review
Resolve Wing Design Review Issues
Wing Design Milestone Finalization
Design Engine
Engine Design Review
Resolve Engine Design Issues
Engine Design Milestone Finalization
62%

Figure 5.14: The BPMN editor allows the design of BPMN fragments used as plans in GPMN process or design standalone BPMN processes

Like its GPMN counterpart, the BPMN editor is also based on the JGraphX framework and follows the same approach of a separation between the visual and business part of the business process model. The readers and writers are also split between a pluggable visual part and a base business part with each being referenced through interfaces. The default, and currently only implementation of both, currently generates the BPMN 2.0 XML format as defined in [121].

Both editors together allow the creation of GPMN models and associated BPMN fragments used as plans. The next section will discuss options for executing workflows created by those editors and introduce the infrastructure used to implement the enhanced workflow management system used for dealing with the special requirements of long-running autonomous workflows.

# Chapter 6

# GPMN Workflow Execution

The previous chapter has shown how the Goal-oriented Process Modeling Notation (GPMN) can be used to design a business process model during the design phase and, using the editor tools, can be enhanced with plan implementations using the editor tools to generate a workflow model in the implementation phase.

The next phase in the BPM lifecycle (see section 2.7) deals with the execution of such workflow models. This phase involves multiple parts including the creation and actual execution of workflow instances, user interactions typically through the delivery of work items and interaction with predefined services. As noted in chapter 2.4, this step is part of the responsibilities of a workflow management system.

The creation of workflow instances and their execution is usually accomplished by a part of a workflow management system called the workflow engine. Since all other parts of a workflow management system depend on the workflow engine executing workflow instances, it is the first issue that needs to be addressed to provide the tools necessary for the execution phase. Additionally, in order to provide the other parts of the workflow management system and to allow access to other systems involved in the workflow such as database systems, web servers or machinery, a suitable middleware is useful to assist the workflow management system to accomplish this task. As a result, this chapter will deal with both issues by proposing a suitable solution for a workflow engine and selecting a workable middleware solution for implementing the remaining parts of the workflow management system.

## 6.1 Workflow Engines

The execution of workflow instances within a workflow management system is accomplished through the use of a workflow engine. The workflow engine is responsible both for creating runnable workflow instances from the workflow model by providing a runtime state as well as offering the means to perform the actions specified by the workflow at the appropriate time during execution.

Implementations of workflow engines usually follow one of two approaches to execute or enact workflow instances:

- *Model Conversion*: Before a workflow instance is created, the workflow model is converted to a different model using an available executable language for which an interpreter is available. This interpreter then provides the workflow instance with the runtime state and uses the converted model to execute the workflow.

- *Direct Interpretation*: The workflow model is directly represented in-memory. An interpreter capable of executing the language directly, and providing the runtime state, is used to create and execute the workflow instance.

Model conversion is a common approach for executing BPMN models by first converting the model to a BPEL model, then using an execution environment that includes a BPEL interpreter to execute the converted BPEL model. Popular implementations supporting this approach include workflow management systems like the Oracle BPEL Process Manager (see [75]), Microsoft BizTalk (see [74]), Apache Orchestration Director Engine (see [49]), SAP Exchange Infrastructure (SAP XI, see [2]) and IBM WebSphere Process Server (see [73]).

The advantage of this approach is that the workflow engine becomes reusable for different workflow model languages: Each additional modeling language has to provide a model converter and is then able to use both the engine and infrastructure already available for the targeted language such as BPEL. Additionally, BPMN 1.0 lacked clearly defined semantics so the conversion to a language with well-defined execution semantics like BPEL essentially attaches a specific interpretation of the original language model. Model conversion therefore allows a quick adoption of alternative interpretation should this become necessary.

The disadvantage is that complex conversions with a focus on execution tend to lose information about the original workflow model. Consequently, the resulting BPEL workflow after conversion from a BPMN model may not provide sufficient information about the original model, thus causing tasks like debugging to become more difficult since errors occur in the converted model used for execution which is difficult to link back to specific elements of the model created by the workflow designer. Additionally, workflow models that allow automated proving of properties (e.g. petri nets) may invalidate such proofs unless the model converter itself is proven correct.

The alternative to model conversion is direct interpretation in which the model and its elements are directly represented in the workflow engine as programming structures which are then executed by an interpreter capable of interpreting the workflow language. The advantage is that the association with the original workflow model remains clear and any event such as errors can be directly associated with elements in the model.

Conversely, the disadvantage is that the interpreter is specifically developed to interpret a particular workflow modeling language and cannot easily be reused for other languages. This generally limits such workflow management systems to a specific type of language without additional support by other language interpreters.

Nevertheless, with the release of BPMN 2.0, this approach has become popular in more recent workflow management systems that employ BPMN as the primary language, which often avoid implementing support for BPEL completely. Common modern examples of this approach include the workflow engines integrated in workflow management systems like Activiti (see [4]) which was specifically developed for this approach and JBoss jBPM (see [90]), which started out with model conversion but switched to direct BPMN 2.0 interpretation when it became available.

Generally, an interpreter tends to be the more adaptable approach since it is tailored towards the specific language which helps to include support for more specialized language features, ensures the integrity of the relationship between a runtime instance and the business process model. Nevertheless, a model conversion approach is considerably quicker to implement if an execution engine is already available and the model language and execution language are reasonably similar.

However, while GO-BPMN is at least adopted specifically as a conservative extension to BPMN and therefore has similar execution semantics, the additional complexity of GPMN with goal and plan instantiation as well as the goal deliberation and means-end reasoning cycles would require a non-trivial translation if a workflow engine for a task-based language is used.

An alternative option would be the use of a rule-based workflow engine. For this approach, the GPMN intermediate model would be converted to a set of rules which is then used in a rule engine to achieve the behavior intended by the workflow model. However, this approach would, aside from the obvious advantage during modeling, have similar disadvantages to a pure rule-based modeling approach as described in section 3.9:

- The model itself could be expressed in GPMN. This would reduce the development and maintenance difficulties of the rule-based modeling approach, thus providing an advantage in the modeling part of workflow design.

- It is non-trivial to maintain a connection between converted rules and the model elements. Therefore, information about the model elements such as goals is likely to be lost. As a result, strategic-operational cohesion would become unattainable at runtime.

- As with any model conversion approach, debugging becomes more difficult. This disadvantage is emphasized by the loss of strategic-operation cohesion. This disadvantage ties into the issue of testability of rule-based systems raised by Li (see [101]).

- Monitoring such processes would become harder if the engine is used as part of a workflow management system.

As a result, conversion to a rule-based system is less than ideal especially considering the research goals for long-running autonomous processes during runtime. However, considering the autonomous behavior of the business processes, a promising alternative is agent technology since software agents are also expected to exhibit autonomous behavior. Furthermore, a particular agent architecture called Belief-Desire-Intention

(BDI) agents is interesting since it also includes the use of goals similar to goal-oriented business processes.

The next section will provide a short introduction to agent technology and the BDI approach, followed by an explanation as to how this approach can be used to execute GPMN workflows.

## 6.2 Agent Technology

In the previous chapters, business processes, business process models, workflows and workflow models were introduced and the goal-oriented language GPMN was determined to be a good choice for modeling long-running autonomous processes. In addition to merely executing the resulting workflow models, one also has to consider the desired autonomous behavior that such workflows are expected to show. This makes software agents an interesting technology for executing GPMN workflows since it shares similar goals in this regard.

In this section, the concept of software agents is introduced, its relationship with long-running autonomous business processes is highlighted and the specific technology of BDI (Belief-Desire-Intention) approach is explained in greater detail since it offers a solution for executing long-running autonomous processes that is particularly suited to workflow models design in the GPMN workflow language.

### 6.2.1 Definition of Agents

Definitions of agents in general and software agents as agents that are part of a software system vary greatly and a vast number of different types of software agents have been proposed. As such, there is no rigorous and generally agreed-upon definition of software agents. Nevertheless, a number of themes and agreement on certain aspects of software agents can be derived from literature and convey a soft but general sense of the nature of software agents by providing a number of possible properties of entities that result in it to be considered a software agent. A number of different views on the term ”software agent” are discussed by Frankling and Graesser in [54].

Despite the diversity of views on software agents, a number of attempts to define software agents are available. One of the more widely accepted definition of an *autonomous agent* is offered by Maes in [106]:

> Autonomous agents are computational systems that inhabit some complex, dynamic environment, sense and act autonomously in this environment, and by doing so realize a set of goals or tasks for which they are designed.

Here, the emphasis is on three aspects; the autonomous actions of the agent, a dynamic environment and its interaction with this environment, which are potentiall useful properties for long-running, autonomous business process as well since they also necessitate autonomy and need to interact with a (business) environment.

This view on agents is also supported by possibly the most widely accepted definition an agent by Woolridge and Jennings in [91] and in [163]:

> An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives.

Again the definition emphasizes the autonomy of the agent and an environment in which the agent can interact, however, the definition of the environment is slightly more open and undefined. It also omits the obligations to realize goals and tasks, making this definition more general but also less clear. However, due to being more inclusive, this definition was also used as the basis for the definition used in [104]:

> An agent is a computer program capable of flexible and autonomous action in a dynamic environment, usually an environment containing other agents.

In this definition, the capability of the agent's actions include flexibility, another property desirable for long-running, autonomous business processes. In addition, this definition also notes that it is common for a system to include multiple agents. These sorts of systems are called *multi-agent systems* and are defined by allowing multiple separate agents to interact with the environment (see [163] and [138].

Most widely accepted definitions contain the two aspects of an independent system, called the agent, capable of autonomous action and an environment which this system can affect. These aspects are also present in [105], [65] and [138].

Additional properties agents can have which are relevant in the context of this work, but are not necessarily a property of all types of agents, can be found in[115]. Here, agent can be considered either to be *reactive* or *deliberative*. Reactive agents only perform actions based on some external triggers. These can be both triggers based on perceived changes in the environment or perceived actions performed by other agents.

The reaction can either be predictable and repeatable if the trigger is repeated, or they can differ for each trigger event, even if the trigger itself is identical. The latter requires more sophistication on part of the agent and necessitates an internal state (see also [138]); however, agents with such a behavior are still considered to be reactive.

In contrast, agents can also be deliberative. This means that, unlike reactive agents, they do not depend on external triggers but can take a *proactive* role which allows the agent to initiate actions by itself based on its internal state without an external trigger being perceived by the agent.

For this behavior to be meaningful, agents are often motivated by an internal objective or goal. These type of agents are call *goal-oriented*, since they have internal goals which they pursue and use to decide on the action they are performing. Similar to the more sophisticated reactive agents, deliberative and goal-oriented agents usually maintain an internal state. This state may not only include their own internal properties but can also contain a model of the environment and defined goals to help them plan their actions (see also [96]).

Wooldridge and Jennings, while also noting the difficulty in defining agents, provide the following properties to clarify the notion of agents (from [164]) and define some core properties of agents:

- *autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state [32];
- *social ability*: agents interact with other agents (and possibly humans) via some kind of *agent-communication language [56];*
- *reactivity*: agents perceive their environment (which may be the natural world, a user through a graphical user interface, a collection of other agents, the Internet, or perhaps all of these combined), and respond in a timely fashion to changes that occur in it;
- *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by *taking the initiative.*

In addition, both Wooldridge [163] and Russel et al. [138] give some properties regarding the environment. The first property is the *accessibility* of the environment. Accessibility is the degree in which an agent can perceive changes in the environment. In order for an agent to interact meaningfully with its environment, at least some changes such as changing states or exchange of messages must be perceivable by the agent.

However, the perception of the environment by the agent does not necessarily have to be complete or encompass the full state of the environment. The perception can be limited to a subset of the environment. For example, an agent representing an animal in a simulated physical environment may be limited in its perception to a certain distance and angle. By limiting the agent's perception in such a way and developing the agent based on such restrictions, the behavior of the agent may resemble the targeted behavior more closely.

Another environmental property is the question whether the state of the environment is *deterministic*. An environment is considered to be deterministic if changes to its state can be predicted by the agent. This includes environments where the agent affects the environment but is able to predict the influence of its own action on the environment. Non-deterministic environments are environments that are either influenced by changes outside the agent or if the results of actions performed by the agent cannot be predicted.

A property which is in partial contradiction to the definition by Maes in [106], environments can be either *static* and *dynamic*. Static environment do not experience state changes unless acted upon by the agent. Dynamic environments, on the other hand, can change regardless of the actions of the agent. This results in certain interdependencies with other properties.

For example, a multi-agent system filled with non-trivial agents will cause an environment to become dynamic, since the behavior of other agents cannot be predicted by any particular agent since their behavior is autonomous. The exception to this is a system with simple reactive agents without an internal state, except, at most, one more complex agent, because that agent could predict the behavior of the trivial reactive agents.

This also highlights how the determinism of the environment depends on the perception of a particular agent in multi-agent systems: The environment described above is deterministic for the agent able to predict the behavior of the trivial reactive agents, but it is non-deterministic for the reactive agents since they cannot predict the behavior of the more complex agent.

In addition, a deterministic environment tends to also be static, since the only deterministic and dynamic environment is only possible if the dynamic behavior of the environment is entirely predictable by the agent. Similar to the example above, this is only the case if the dynamic behavior of the environment is trivial.

As a result of most dynamic environments being non-deterministic, the behavior of the overall system results in increased overall complexity and greater difficulty in predicting its actions. It is therefore harder for a system designer to plan in advance, resulting in greater design complexity for development. In addition, it often prohibits analytical behavior analysis and increased effort for validating system behaviors.

### 6.2.2 Agents in Software Development

Software agents are an approach for perceiving systems or part of systems. In particular, a software agent is a piece of software that is regarded as having a certain degree of independence or autonomy to decide or act by itself rather than behaving like a simple object as used in typical object-oriented software development (see [69]), blindly and mechanically performing based on external input.

However, this is merely a choice by a developer to perceive and consequently treat a system in such a manner because it is *useful* to them in the context of his work and helps them to represent real world or abstract concepts in software. By itself, it does not offer any technical advantage over other approaches such as object-orientation, i.e. any software system can be developed with either technique, but both convey development advantages in particular contexts.

This view of agents as merely a point of view is supported by Shoham in [145] as follows:

> It is perfectly coherent to treat a light switch as a (very cooperative) agent with the capability of transmitting current at will, who invariably transmits current when it believes that we want it transmitted and not otherwise; flicking the switch is simply our way of communicating our desires.

This example shows that even a relatively primitive mechanical object can be perceived as an agent, despite the fact that it is more natural to view a light switch as a fairly primitive mechanism since its capabilities of autonomous action is quite limited, though one could interpret a mechanical fault in the light switch as the agent's refusal to perform its function. Shoham argues in [145]:

> [...] it does not buy us anything, since we essentially understand the mechanism sufficiently to have a simpler, mechanistic description of its behaviour.

In fact, it can be argued that this interpretation is actually harmful in that it actively impedes the resolution of potential problems: Interpreting a faulty light switch as an agent refusing to conduct current in contradiction to a request offers no useful recourse to the fault, since there is no way to "persuade" or "negotiate" with the agent in a manner that will result in it to change its behavior.

In contrast, interpreting the malfunctioning light switch as a simple mechanism intuitively provides an option to remedy the lack of functionality by identifying the flaw in the mechanism and perform physical changes to it to restore function.

As a result, since agent-oriented development does not add actual capabilities to the system itself, it should be regarded as "yet another tool" for the developer to employ. Perceiving a system as an agent or multi-agent system during development is only useful to the extend that it aids the developer to design and implement the system. If it helps a system designer to think about the system in a manner that eases understanding, it should be used. If a different approach, such as object-orientation presents itself as more suitable, it should be preferred.

### 6.2.3 Agent Architectures and BDI Agents

While the previous segment gave an introduction to *agent theories*, which attempts to give a *specification* of what an agent is and what properties define it and how to reason about them, this section will give a brief overview of *agent architectures* (cf. [164] and [144]).

Agent architectures attempt to provide a path to turn agent theories into a practical implementation of agents. Since the goal is to implement the desired properties of agents derived from agent theories, this includes the question of how to structure agents in order to best fulfill those properties. In case of software agents, this addresses the software architecture of the agents.

Specific implementations of agent architectures often also specify an *agent language*. The agent language specifies how an agent is to be programmed, i.e. which language primitives are used to describe the agent and the agent's behavior and provides a path for executing agents such as compilation or interpretation.

Since there is no universally accepted definition of what an agent is (cf. section 6.2.1), a large variety of agent architectures, accompanying languages, frameworks and execution platforms have been developed and described in literature.

Of the available agent architectures, the Belief-Desire-Intention (BDI) approach appears promising for the execution of GPMN workflow models since it contains a notion of goals similar to the one used in the goal-oriented GPMN workflow models.

BDI theory was first approached by Bratman (see [17]) as a means for describing the rational behavior of human agents. Rather than an approach emulating the emergence of rational behavior from a biologically inspired model, BDI uses abstract and mentalistic concepts to describe rational reasoning. The approach chosen for this offers three major concepts used to describe rational behavior as follows:

- *Beliefs* describe the knowledge an agent has about the world. Since this knowledge is usually acquired by the agent itself, beliefs may not be an accurate, current and complete representation of the world. The agent also assumes that all of the acquired beliefs are true. If new information becomes available, the beliefs may be changed to reflect this circumstance. Uncertain knowledge with probabilities are not modeled in this approach.

- *Desires* represent states of either the agent or its environment which the agent considers to be advantageous and is willing to work towards achieving them. Multiple desires of an agent merely have to represent desirable states for the agent with no consistency between them being required. It is completely acceptable for an agent to hold contradicting desires.

- *Intention* include all the actions that an agent has currently committed to performing in the future in order to affect changes towards the desires. This concept is more concrete than the desires in that it does not merely describe the target but the path towards it. Bratman includes in this the concept of plans which describe a set of actions that represent a step towards carrying out an intention and therefore may be part of a partial solution for the whole.

BDI follows a reasoning cycle which, similar to GPMN, is divided in two steps, deliberation between the desires and means-end reasoning to determine the intention. Deliberation is necessary in order to resolve conflicts between conflicting desires since the intent must not contain contradictory action. This is due to the fact that executing contradictory actions leads to cyclic and erratic actions which represent an irrational behavior that stands in opposition of the goal of rational reasoning.

Once the deliberation cycle resolves conflicting desires, means-end reasoning is used to find a rational solution for reaching the desires. As a result, the means-end reasoning determines the intention by evaluating the available actions with regard to the desires resulting from the deliberation cycle.

Since BDI lends itself towards an easily intelligible yet reasonably complex model for reasoning and rational behavior, BDI theory was adopted by multiple software agent platforms which use this approach as the basis for their respective agent architectures. Recent examples of such platforms include e.g. JACK Intelligent Agents (see [71]), Jason using the AgentSpeak language (see [16] and [15]), BDI4JADE (see [117]) as an extension to the more basic JADE agent architecture (see [8] and [9]) and Jadex (see [18], [133] and [22]).

Most BDI software agent platforms implement BDI theory as part of an agent model with an execution semantic which includes the following concepts to represent the BDI ideas:

- The *belief base* represents the agglomeration of the beliefs of the agents in the sense of the beliefs in BDI theory. This concept is usually implemented as a typed data store.
- The desires of BDI theory are expressed on BDI software agent platforms as *goals* with configurable interrelations and conditions.
- Finally, intention is not usually explicitely expressed. Instead, intention is implemented by *plans* which fulfill the same role as the plans defined by Bratman by representing steps towards or parts of a whole intention.

As a result of the use of similar concepts such as the autonomy of agents and specific BDI aspects such as goals, the BDI approach represents a good target of a model conversion approach towards executing GPMN workflow models and thus the agent platform itself could serve both as a middleware and an execution environment as well as a workflow engine for GPMN workflows.

For the implementation of the model conversion, the Jadex platform with its BDI execution semantic was chosen as the targeted execution engine. In terms of BDI execution, there is no particular reason for a particular BDI platform. However, Jadex also offers alternative pluggable execution semantics as part of its active component model (see [127]and [128]) which also includes a BPMN interpreter for executing BPMN 2.0 compatible workflows. Since BPMN is easier to understand for workflow designers, it is desirable to have a BPMN interpreter available in order to allow the execution of plans implemented in BPMN as part of the overall GPMN workflow.

Additionally, Jadex conveys certain advantages as a middleware platform which helps to implement a workflow management system that offers better support for long-running autonomous workflows as further discussed in chapters 7 and 8.

## 6.3 Jadex BDI Agents

The Jadex Active Components platforms offers a number of different agent architectures and other interal architectures by including plugins called kernels, which provide the execution semantics for these architectures. Among those kernels included in Jadex, two are interesting options for executing GPMN workflows: First, the BPMN kernel which comes with a BPMN 2.0 model loader and a BPMN interpreter for executing BPMN workflows. Second, the BDI kernel which can execute BDI agents, which has an execution semantic (see [130]) that could be used as an engine for the GPMN workflow models.

When performing the deliberation cycle, BDI systems must follow a strategy which cannot be overly complex if the deliberation is to be done in realtime. Jadex offers a realtime approach as its deliberation strategy for goals called easy deliberation, which allows the definition of constraints in the model which the deliberation cycle can follow to determine viable goals (see [129]). The main constraints available are constraints on cardinality which acts on the created goal instance and goal inhibitions, which declares a goal to be inhibited by another goal.

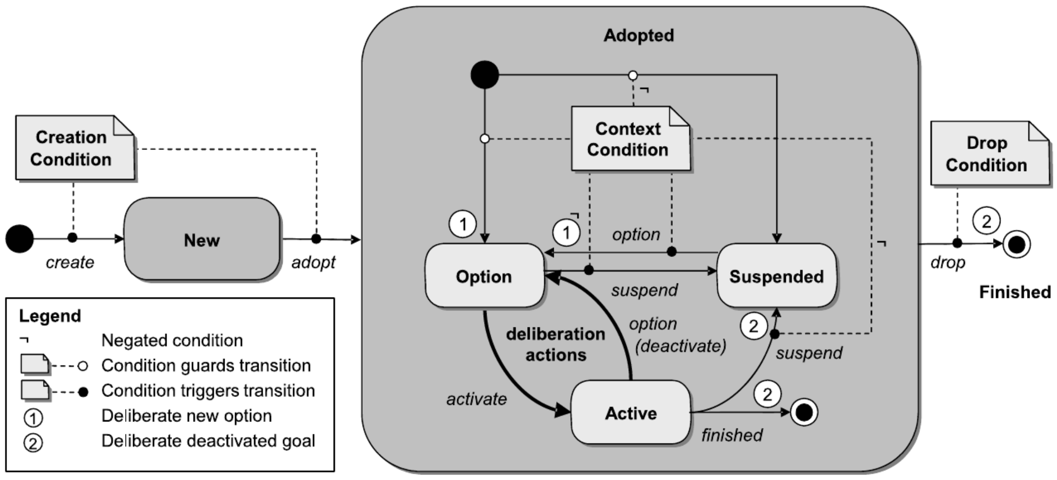


Figure 6.1: Goal life cycle as used by the Jadex BDI kernel (from [129])

Jadex follows a life cycle for goal instances as shown in Figure 6.1). In this life cycle, goal instances are first created and then adopted based on a trigger (e.g. a creation condition), after which this goal instance is included as part of the set of adopted goal instances. Once a goal instance is adopted, it can enter three different states.

When a goal is first adopted, it enters the *option* state, which means that the goal is not restricted and is available to be considered during goal deliberation. If the deliberation cycle selects a goal instance, it enters the *active* state in which the agent will actively use means-end reasoning to reach the goal. If the goal instance reaches its desired state, it terminates. A deliberation cycle can also remove the active state from a goal and return it to the pool of goals in the option state.

If a goal instance becomes restricted, for example because its context condition becomes invalid, the goal enters the *suspended* state. While suspended, the goal does not participate in the deliberation cycle until the reason for the suspension disappears such as the context condition becoming valid again. However, even if the goal was in the active state when it was suspended, it will first enter the option state again and must be subsequently reselected by the deliberation cycle before it can enter the active state again. Finally, if the drop condition of a goal instance evaluates to true, the goal instance immediately terminates. This happens regardless of the state it is in while it is still adopted, including being in the active state.

During the active state, the means-end reasoning cycle is used to determine plans that can be executed to reach the goal. Plans can be declared in the BDI agent model and are associated with goals by including a *plan trigger* referring to a goal. Plans can be reused by multiple goals by including multiple triggers.

By default the means-end reasoning in Jadex is straightforward: Associated plans are executed in declared order until the goal is reached. Plans can therefore contribute partial solutions which will be completed by another plan. Alternatively, all associated plans can be executed in parallel. Furthermore, Jadex offers meta-level goals to modify and extend the means-end reasoning cycle, however, they are currently not used by GPMN.

A plan which is currently being executed can be aborted at any time. For example, if a goal has been reached, it is no longer necessary to execute the associated plans and they are terminated. Naturally, this also applies if a goal has conclusively failed. Failure by plans on the other hand are considered to be normal and the next plan is tried if one has failed without the goal itself failing. Plans can also be retried if configured in such a manner.

Plan execution is also terminated while the goal instance is still adopted. Since plans are only allowed to be executed while a goal instance is in the active state. If the deliberation cycle deactivates the goal instance and it enters the option state or if the goal becomes restricted and is suspended, all plans associated with that goal instance that are currently executing are aborted. The abortion of plans is facilitated further by allowing plans to include actions that are executed in case a plan aborts, thus allowing the plan to perform a rollback.

The next section will provide the concept and implementation details for converting the GPMN workflow models into Jadex BDI agents and matching up the Jadex BDI semantics with the GPMN approach for workflow execution using the GPMN editor previously described in section 5.5 as a key element for this process.

## 6.4 GPMN Model Conversion to BDI Agents

In order to execute GPMN workflow models as Jadex BDI agents, the model has to be converted to the Jadex BDI agent model format. The Jadex BDI format, like GPMN, is based on XML but is geared more towards modeling BDI agents rather than goal-oriented workflows and includes concept currently not supported by GPMN such as meta-goals. However, Jadex BDI agents use similar concepts to GPMN and uses two reasoning cycles to determine the actions a particular agent executes and therefore would represent a good execution platform for GPMN workflows as well.

Two conversion directions could be performed, converting GPMN workflows to Jadex BDI agents or converting Jadex BDI agents to GPMN workflows. The former is necessary for workflow execution, the latter would only be useful if modifications are performed on the BDI agent after conversion and a round-trip engineering cycle needs to be established. Since modeling of workflows is expected to be performed on the GPMN level 1and the primary goal is to execute GPMN workflows and considering the fact that Jadex BDI agents include concepts that cannot be represented in GPMN, the focus will be on the conversion of workflows to agents.

Figure 6.2 is the conversion chart used to convert GPMN workflow models to Jadex BDI agent models. The first concept that needs to be converted is the GPMN workflow context. The context represents the business situation as well as the workflow state. Provided the runtime system will supply the agent with current information about the business environment, the workflow context can be represented in the BDI agent as its belief base with individual context entries being the beliefs or

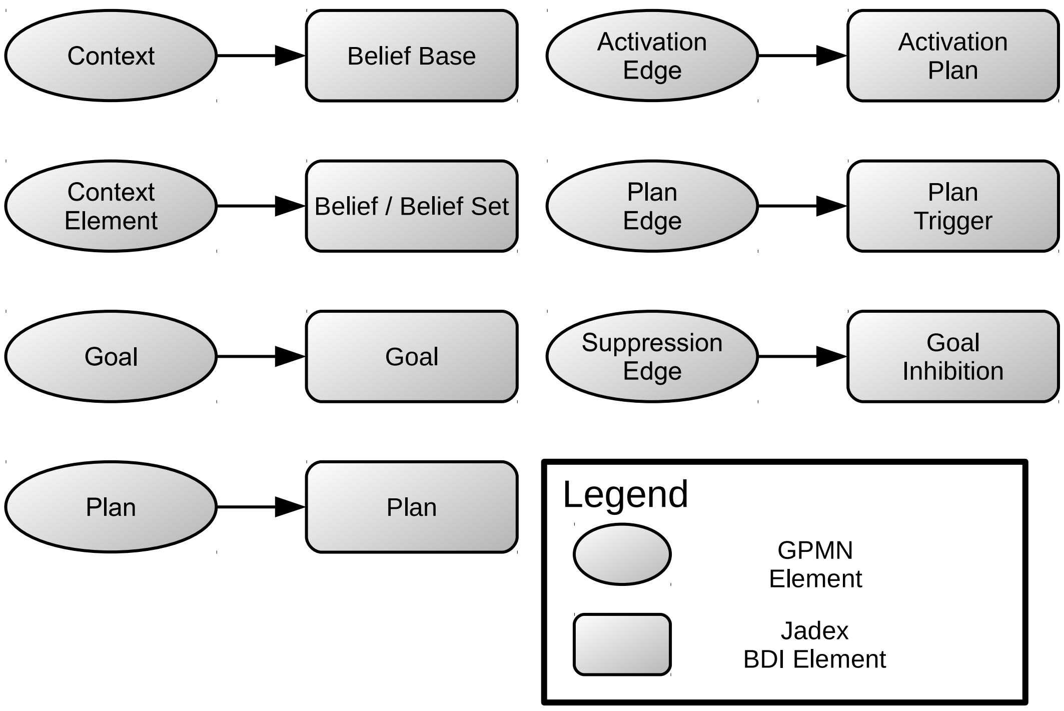


Figure 6.2: Model conversion chart for converting the elements of a GPMN workflow model to a Jadex BDI agent model

belief sets of the agent.

Goals in GPMN workflows can be represented directly as goals of the BDI agents. As with GPMN, multiple goal kinds are available (see [23]), including goal kinds matching the ones used in GPMN workflows. GPMN plans can be implemented as plans in the BDI agents. In Jadex, plans for BDI agents reference Java classes containing the code implementation of the plan, so an extension is necessary to support referencing BPMN fragments as plans (see section 6.5).

```
<plan name="alarm_plan">
    <body class="AlarmPlan" />
    <trigger>
        <goal ref="alarm"/>
    </trigger>
</plan>
```

Figure 6.3: Declaration of a trigger for a plan, referencing a simultaneously declared ”alarm” goal

Plans in GPMN are attached to goals using plan edges, however, Jadex BDI agents are not graph models and therefore do not include edges as a concept. Instead, plans refer to goals by including plan triggers with a reference to the name of the goal in the declaration of the plan itself (see Figure 6.3). Multiple references can be included in the plan declaration, allowing reuse of the plan for multiple goals, the same of which can be achieved in GPMN by connecting a single plan with multiple goals using multiple plan edges. Therefore, plan triggers are, despite being singlelined references instead of edges, capable of representing the plan edges of GPMN workflows.

```
<maintaingoal name="MaintainBatteryLoaded">
  [omitted parameter and condition specs. for brevity]
  <deliberation>
     <inhibits ref="AchieveCleanupWaste"/>
     <inhibits ref="PerformLookForWaste"/>
     <inhibits ref="PerformPatrol"/>
  </deliberation>
</maintaingoal>
```

Figure 6.4: Declaration of a maintain goal with three inhibition references (from [129])

Suppression edges in GPMN indicate which goals may suppress other goals during goal deliberation by forming an edge connection in the model. In Jadex BDI agents, the same effect can be reached by including an inhibition reference with the declaration (see Figure 6.4) of the goal that needs to suppress other goals in the GPMN model. Therefore, for each outgoing suppression edge, the declaration of the source goal has to include an inhibition reference to the target goal. If multiple outgoing suppression edges are present, multiple inhibition declarations are required.

The final GPMN concept, activation edges involves a more complex solution to implement using Jadex BDI agents. In GPMN, activation edges are directed edges which form a direct connection between goals, indicating that a goal activates the connected goals as subgoals. For this, two kinds of semantics are available: The default behavior is to activate all of the subgoals at once, pursuing them in parallel. As an alternative, GPMN allows an activating goal to be declared as a sequential goal, in which its subgoals are activated in an order depending on declared order values annotated on the activation edges.

This GPMN concept has no close equivalent in BDI agents. In fact, BDI agents do not allow the referral of goals from other goals at all. While plans are capable of activating goals both as top level goals or as subgoals by calling a library method, goals include no executable code on their own and are therefore incapable of directly activating subgoals. Therefore, when constructing goal hierarchies with subgoals as modeled in GPMN, BDI agents require the implementing developer to alternate between goals which trigger plans which in turn can activate subgoals. The goal activation semantic in Jadex BDI agents is therefore part of the business logic of the agents rather than an innate feature like the activation edges and sequential and parallel goal types used in GPMN.

As a result, when converting from a GPMN workflow to a BDI agent model, an *activation plan* must be injected between the top goal and its connected subgoals which explicitly activates the subgoals by calling the library method. Since this plan only has a limited functionality, it can be provided as a technical feature by the conversion process.

The plan can be included as part of the support library for GPMN workflows for Jadex and is prewritten with the necessary goal activation semantic as defined by the GPMN activation edges. Since the plan is provided by the library, the workflow designer is not required to provide any code.

The next section provides details about the implementation of the model conversion and execution of GPMN workflows as Jadex BDI agents. The resulting solution will provide the workflow engine for GPMN processes and a platform for the implementation of the rest of the workflow management system.

## 6.5 Implementation Aspects of GPMN Workflow Execution

The previous section demonstrated that GPMN workflow models can be converted to Jadex BDI agents by transforming the GPMN workflow elements into elements of BDI agents in order to use the BDI reasoning system as the workflow engine for such processes. When considering the implementation of a model conversion process, the first issue that arises is the question of where the conversion process should be performed.

The conversion of GPMN workflow to Jadex BDI agents can be accomplished in two different parts of the system: The GPMN workflow editor could perform the conversion process and provide an executable Jadex BDI agent which can be executed directly using Jadex. Alternatively, Jadex could be provided with an additional kernel which reads a GPMN intermediate model XML file and converts it to a Jadex BDI agent before execution.

In this case, performing the conversion process as part of the editor conveys some advantages over conversion of the workflow model as part of the execution platform:

- Model conversion as part of the editor introduces a clear boundary and division of responsibilities between the editor and Jadex as the execution platform: The editor handles the workflow-related details and then prepares a suitable executable model to Jadex. This also lets the editor alert the right person, the workflow designer, should issues arise during the conversion process. If the conversion is performed as part of the loading process before execution, not only can this be confusing to users by requesting the execution of a workflow resulting in an executing BDI agent, but the workflow management system administrator is unlikely to know enough model details to correct any conversion issues.

- The editor provides a pluggable reader/writer architecture that allows a special writer to be implemented which performs the conversion process by writing a Jadex BDI agent model output. While Jadex also offers pluggable extensions for providing additional component functionality called ”kernels”, this option is technically more involved than the implementation of a writer plugin for the editor.

- Performing the conversion within the editor is akin to ahead-of-time compilation of a program and offers a similar advantage: Once the conversion is performed, the runtime system can immediately begin executing instead of first performing the conversion process, possibly for each new instance of the process. A caching mechanism could prevent multiple conversions of the same model but at the cost of memory. These issues are avoided altogether by including the conversion in the editor.

Therefore, the model conversion has been implemented as part of the GPMN editor (see section 5.5) by implementing another model writer using the writer interface as shown in Figure 5.13. Since Jadex BDI agents do not include a graphical representation, no visual model writer was implemented and it defaults no a null reference, which means that the visual part of the model is omitted.

```java
public void body()
{
    String[]    subgoals    = (String[]) getParameterSet("subgoals").getValues();
    String  mode    = (String) getParameter("mode").getValue();

    if("parallel".equals(mode))
    {
        IGoal[] goals   = new IGoal[subgoals.length];
        for(int i=0; i<subgoals.length; i++)
        {
            System.out.println("Creating Goal: "+subgoals[i]);
            goals[i]    = createGoal(subgoals[i]);
            dispatchSubgoal(goals[i]);
        }
        for(int i=0; i<goals.length; i++)
        {
            waitForGoal(goals[i]);
        }
    }
    else
    {
        for(int i=0; i<subgoals.length; i++)
        {
            System.out.println("Creating Goal: "+subgoals[i]);
            if(subgoals[i]!=null)
            {
                IGoal   subgoal = createGoal(subgoals[i]);
                dispatchSubgoalAndWait(subgoal);
            }
        }
    }
}
```

Figure 6.5: Plan body of the standard GPMN activation plan used by Jadex BDI agents to emulate GPMN goal activation semantics

The writer receives the GPMN business-side model as shown in Figure 5.11, then using this model to write an equivalent model of a Jadex BDI agent in XML using the conversion mapping shown in the previous section and Figure 6.2. While an implementation of a BDI agent model reader could be useful for roundtrip engineering purposes, the user of the system is not expected to change the converted BDI model; therefore no roundtrip engineering is necessary. Furthermore, the BDI agent model can include elements like meta goals which have no equivalent representation in GPMN, making a reverse conversion only applicable in some circumstances. As a result, the implementation of an equivalent BDI agent model reader was omitted, allowing only the export but not the import of BDI agent models from the GPMN workflow editor.

As mentioned in section 6.4, the default BDI agent model does not offer an equivalent for the GPMN activation edge or the parallel and sequential goal semantics, both of which requires the injection of a technical element called activation plan to perform the necessary goal activations.

The activation plan was implemented and included in a support library and can be referenced by the model writer during the conversion process. Figure 6.5 shows the body of this activation plan. The plan is configurable both regarding the goals it needs to activate as well as the parallel or sequential semantics of its top goal.

This difference in semantics is addressed by offering two cases within the plan, the parallel and the sequential case. In the parallel case, all subgoals are dispatched in a for-loop. A second for-loop ensures that all goals have finished before ending the execution of the plan. In contrast, in the sequential case, each subgoal is dispatched only after the previous one has finished, so that only one goal is dispatched at any given time.

Finally, the BDI kernel in its original state could only support plans implemented in Java the same way that the activation plan was implemented while GPMN processes also call for support of plans that are implemented using BPMN workflow fragments. While Jadex includes a BPMN kernel for executing BPMN workflows, this kernel is distinct from the BDI kernel used to execute BDI agents.

Therefore, it was necessary to extend the BDI kernel in order to support the use of BPMN workflow fragments as an alternative to the standard Java implementation of plans. This was accomplished by using the BPMN interpreter available as part of the BPMN kernel and calling it if the referenced plan in a BDI agent is a BPMN workflow model.

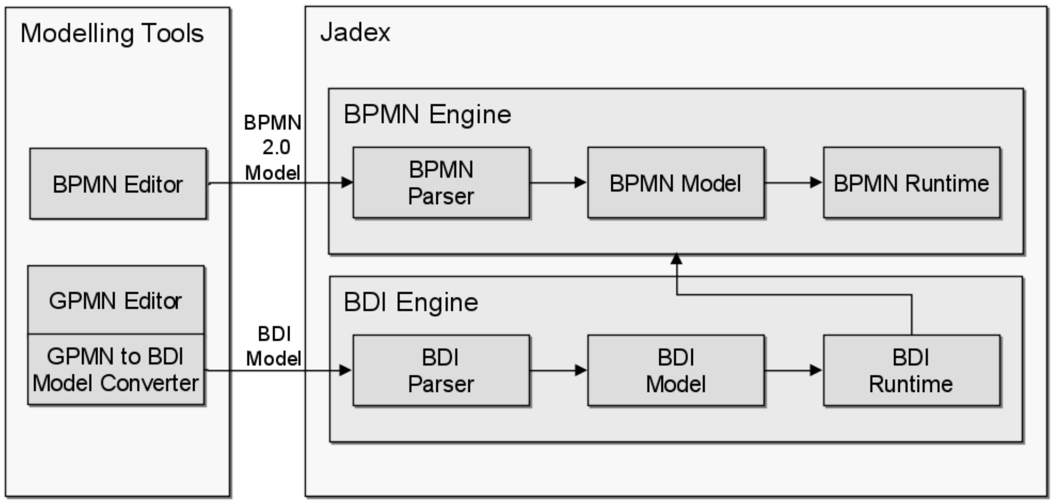


Figure 6.6: Approach for executing GPMN processes in Jadex based on model conversion and the available BPMN and BDI engines

The resulting system (see Figure 6.6) allows the creation of GPMN business process and BPMN workflow fragments as plans using the editors as shown in section 5.5. The BPMN workflow fragments used as plans are saved as a normal BPMN 2.0

workflow model using the BPMN editor. The GPMN goal hierarchy model should first be saved as an intermediate model in order to allow change at a later point, then exported as a Jadex BDI agent model using the export functionality provided by the BDI model writer for execution. The resulting BDI agent together with the BPMN fragments then represent the GPMN workflow on the Jadex execution platform.

# Chapter 7

# Requirements for a Distributed Workflow Management System

The previous chapter described how goal-oriented business processes and workflow modeling can improve the design and implementation phase of the business process management lifecycle for long-running autonomous processes and a flexible goal-oriented business process and workflow modeling language called GPMN was proposed. Furthermore, it was demonstrated how GPMN workflows can be executed by performing a model conversion to a Jadex BDI agent model and using the resulting BDI agent as a representative for the workflow, thus effectively creating a workflow engine for such processes.

However, the mere execution of workflows is insufficient for non-trivial business processes since additional infrastructure is necessary to properly include and automate the business environment of the processes including forwarding tasks for human users to the right workflow participants as part of the execution phase of the BPM lifecycle. Additionally, the monitoring phase also requires support for various tasks such as event processing and logging. This infrastructure, of which the workflow engine executing the workflow instance is considered to be a vital part, is called the workflow management system (see also section 2.6).

This chapter will introduce the necessary infrastructure that forms a workflow management system for long-running autonomous business processes. The system participates in both the execution and monitoring phase of the BPM lifecycle and therefore has to address the research goals for those phases set out in section 1.2.

## 7.1 Execution Platform and Service-oriented Middleware

As defined in section 2.7, workflows represent the automated part of a business process. However, business processes are also required to interact with the outside world, for example, with departments or competency centers providing services to the workflow and with customers as shown in section 2.1.

In the previous chapter, the workflow engine for GPMN workflows was introduced by combining the modeling tools with the Jadex BDI and BPMN engines but since real world interaction is necessary, executing workflows on a suitable workflow engine is insufficient since the workflows must have a way to communicate with the business environment.

Since the departments or competency centers of organizations are spatially distributed, this interaction necessarily requires the use of some sort of distributed system which allows the workflow engine to communicate with client nodes and machinery over a wide area. Generally, two possible types of infrastructure are possible with one having two common options of implementation:

- Centralized workflow management using a central server executing workflows with workflow participants accessing the system through e.g. a web interface. This approach is fairly limited, for example, the workflows cannot access machinery or other systems directly without human mediators.
- A service-oriented architecture as introduced in section 2.8. While the workflow is executing, it has the opportunity to call services as part of the service-oriented architecture. Interactions with external systems are done through those service calls which is also used to issue tasks to human participants of the workflow.

Due to its limitations, the first approach fell out of use, since it is often highly desirable to include legacy systems and machinery in workflows which makes such an approach non-viable. As a result, for most modern implementations a SOA-based approach is chosen, often based on either WSDL- or REST-based web services.

## 7.2 Workflow Management Systems

As defined in section 2.6, workflow management systems encompass the tools necessary to enact automated business processes or fractions of business processes called workflows and aid their execution. While this includes the means for executing workflows using workflow engines, other components are necessary to adequately support workflows in a business environment.

Many tasks performed by workflows such as notifying workflow participants of labor that needs to be performed as well as documenting results and data from that labor tend to be required in a variety of business environments. As a result, most workflow management systems offer a similar set of features. In order to raise awareness of these common features, the Workflow Management Coalition (WfMC) has standardized a workflow management system reference model shown in Figure 7.1 (see also [68]).

The reference model itself does not prescribe any particular technology or architecture to implement it. Instead it defines a set of features and tools that are necessary to implement a workflow management system in any suitable technology stack. The core of a workflow management system based on the reference model are the workflow engines, which have the job of executing workflow instances based on

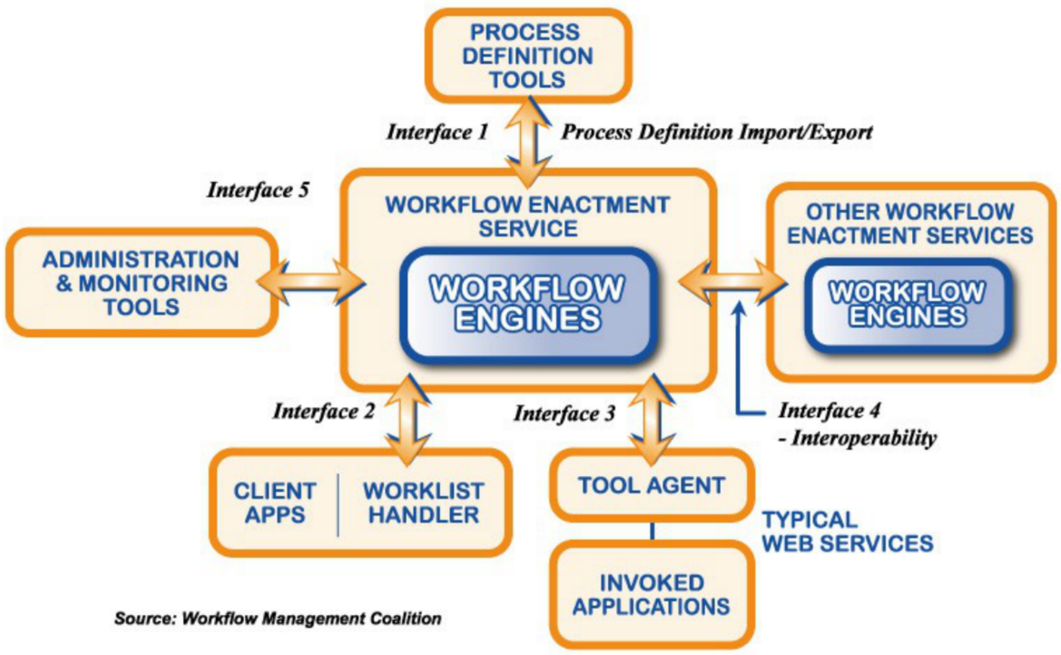

PROCESS DEFINITION TOOLS
Interface 1
Process Definition Import/Export
Interface 5
WORKFLOW ENACTMENT SERVICE
WORKFLOW ENGINES
ADMINISTRATION & MONITORING TOOLS
OTHER WORKFLOW ENACTMENT SERVICES
WORKFLOW ENGINES
Interface 2
Interface 3
Interface 4 - Interoperability
CLIENT APPS
WORKLIST HANDLER
TOOL AGENT
TYPICAL WEB SERVICES
INVOKED APPLICATIONS
Source: Workflow Management Coalition

Figure 7.1: Workflow management system reference model as proposed by the workflow management coalition (see [68])

previously designed workflow models. Multiple engines may be available to support different kinds of workflow models. Surrounding the workflow engines is the workflow enactment service, which helps to enact the workflow instances by maintaining a set of available workflow models, creating the workflow instances and assisting the rest of the workflow management system to interact with the workflows.

This interaction of the workflow management system is based on five different interfaces:

- Interface 1 is the process definition interface, which allows the import and export of workflow models. When a new workflow model is created using the process definition tools (e.g. the GPMN editor), this interface allows the integration of that model into the system and enables the enactment of instances based on this model. It also allows the removal of old or deprecated models in the system.

- Interface 2 represents the workflow client interface. When human workflow participants have to perform a particular task, for example if such a task is defined as a BPMN activity (see section 3.7), this task has to be brought to the attention to the person who is supposed to perform the task. When a task that requires a human participant is executed in a workflow instance, this creates a work item which includes a description of the task and needs to be distributed to the person capable of performing it. Typically, workflow participants use an application called a workflow client application which accesses this interface to retrieve such work items from a worklist offered by the workflow management system.

- Occasionally, workflow tasks are performed by machines which do not use a workflow client application or the human participant such as machines in an assembly line or the workflow uses a legacy applications that do not integrate into the workflow management system as part of interface 2. For these cases, interface 3 is used to allow the workflow instances to communicate either with such machines or a wrapper around the legacy application. In many workflow management system, this interface are the typical web services that are called by the workflow.

- Depending on their requirements, organizations may deploy multiple workflow management systems, each with their own workflow enactment service. Alternatively, there may be a desire for multiple organizations to integrate their systems to further cooperation. Furthermore, multiple instance of the enactment system may be desirable in order to distribute the load when a large number of workflows are enacted. Interface 4 facilitates the interoperability between multiple workflow enactment systems, which can also include different workflow engines to enable additional workflow model types. In many system, this interaction is also performed using web services.

- Handling the workflow participants requires a user management that defines user accounts and roles. Furthermore, some workflows may require manual enactment or detailed management such as the retractions of tasks from an temporarily unavailable participants may be required. Finally, the monitoring phase of the business process management lifecycle stands in need of a system for monitoring workflow instances which then needs to be presented, analyzed and logged by administrative employees. Interface 5, the administrative interface, allows these types of administrative interaction with the system using administration and monitoring tools.

The workflow management system reference model sets the baseline of what is required by a workflow management system. However, long-running autonomous business processes have their own special requirements regarding their execution which also affect the workflow management system (see section 1.2). Therefore, the next section will outline the requirements for a workflow management system which attempts to meet these requirements in addition to the baseline set by the workflow management system reference model.

## 7.3 Requirements of a Workflow Management System for Long-running Autonomous Processes

The workflow management system facilitates and therefore substantially influences both the execution and monitoring phases of the business process management lifecycle shown in section 2.7. In order to improve support for long-running autonomous business processes, six research goals have been defined. Since workflow instance agility is only partially addressed by the work for reasons explained in section 1.2 and strategic-operational cohesion has been addressed by the GPMN workflow language, this leaves four research goals concerning the execution cycle:

- The system must support *workflow model agility* by allowing the workflow designer to include contingencies for predictable changes that influence the process. For the execution phase, this is supported by the GPMN workflow engine presented in chapter 6.
- Organizational agility is the most demanding requirement on the workflow management system. The objective of this research goal is to address the needs of organizations which do not necessarily follow a full process-oriented organizational layout but instead retain some functional organization and the associated departmental structure due to reason described in section 2.1. This means the workflow management system has to deal with organizations with fairly autonomous departments which desire to maintain considerable autonomy regarding their internal structure, approach and infrastructure. This requirement not only requires a distributed approach for implementing the workflow management system but also requires that the system allows the independent departments a certain degree of participation and self-management of the system as a whole.
- Balancing global control and local autonomy means that a trade-off has to be reached: On the one hand, workflow management system with a centralized administration excessively favors control and does not allow sufficient autonomy by organizational subunits. On the other hand, a loose system of unrelated web services without any organized workflow management system which includes handling and distributing work items, user and role management and process model management allows for a lot of autonomy of participating actors but does not include enough structure to provide adequate global control. As a result, a middle ground has to be struck, in which a structured workflow management system is employed which is also capable of distributed and devolved administration.
- Finally, the long runtime of the targeted processes requires increased care in terms of system robustness. The demand for this requirement is emphasized further by the distributed nature of the system resulting from the need for organizational agility. When using such a system, the organization may restructure and therefore require a change in the structure of the workflow management system at runtime. Additionally, the high number of involved nodes increases the chance that any particular node might fail. The system therefore must be able to cope with sudden faults such as node disappearances and allow for the distributed workflow management system to be restructured dynamically without impacting the overall system. This requirement can be met through two means: First, the system must allow functional replication to avoid loss

of function in case of faults or shutdowns. Second, if a fault overwhelms the safety margin given by the replication, the system must degrade gracefully to avoid damage to the long-running processes.

The monitoring phase of the business process management lifecycle also falls into the responsibility of the workflow management system. The monitoring parts of the workflow management system also have to address a set of requirements in order to meet the research goals for the monitoring phase:

- The monitoring phase is primarily concerned with gathering events of running workflow instances within the system and making them available to the administrative and analysis tools of the system. The first research goal for this phase is the organizational agility: Since each participating department or competency center is notionally autnomous and maintains its own independent IT system, the workflow management system is required to allow for the integration of the monitoring system with those independent systems by allowing each organizational subunit to maintain its own event gathering part of the system which will still receive all the events occuring within the system as a whole.

- Due to the long execution time, monitoring events must remain available over long stretches of time. Meanwhile, reorganizations and system failures may lead to the loss of nodes and therefore parts of the monitoring subsystem of the workflow management system. As a result, the monitoring part must ensure that gathered events stay available by replicating the event data on multiple nodes.

Many workflow management systems are available and some examples like Activiti (see [4]) and JBoss jBPM (see [90]) are presented in section 6.1. However, while they are very suitable systems for production and administrative workflows, they are difficult to adapt for long-running autonomous business processes for the following reasons:

- The workflow engine or engines together with the modeling tools form an integrated part of the system and are not easily replaced with a GPMN workflow engine and modeling tools.

- While remote services such as WSDL- or REST-based web services can be used to interact with remote systems, the workflow management system funcionality such as user management and monitoring is a monolithic and centralized entity. For example, in case of jBPM, a central Java application server is used as the platform to for the workflow management system. This approach impedes an adequate devolution of workflow management system functionality. While it would be possible to replicate the functionality remotely through the use of proxy services, these services would still rely on the central node which represents a single point of failure and does not allow replication.

As a result, a decentralized workflow management system must be provided to in order to reach the requirements for the support of long-running autonomous workflows. Therefore the options for the core component of such a workflow management system aside from the previously discussed workflow engine needs to be discussed: The workflow enactment service provides the surrounding infrastructure to connect the workflow engine with the business environment as well as enabling the necessary communication and interfaces for the other parts of the workflow management system and is the first component of the workflow management system discussed in the next section.

## 7.4 Workflow Enactment Service

A key component of a workflow management system is the workflow enactment service. This service includes one or more workflow engines for executing workflows and, in many cases, forms an integrated unit with its engines. The enactment service provides the necessary environment and middleware components to enable workflow instances to communicate with the business environment, for example through interfaces in the workflow management system reference model.

For a workflow management system targeting long-running autonomous business processes, it is ideal if the workflow enactment service also helps to support the additional requirements described in section 7.3. Most workflow management systems include a form of enactment service, however, it is not always labeled as such since it sometimes, e.g. in the case of jBPM (see [90]), consists of a BPEL workflow engine on an application server with the use of web services but can be more complex in more powerful workflow management systems.

A relatively straightforward approach for implementing the workflow enactment service would be to follow a similar approach by implementing it as well as the distributed workflow management system based on such a service-oriented architecture (SOA, see section 2.8), for example by using REST web service.

However, for the distributed workflow management system, the Jadex Active Components platform was chosen as the enactment service due to the following advantages it confers over a simple web-service SOA framework:

- Suitable engines for executing GPMN workflow models like the BDI and BPMN engines are already included in Jadex (see sections 6.2.3 and 6.2).
- The Jadex platform offers a comprehensive service model as well as a concept for service searches.
- If required, services can easily be made available as both WSDL and REST web services (see [19]).
- Jadex includes concepts from PaaS cloud computing that includes the use of automatic provisioning, non-functional properties and deployment features (see [25] and [27]).

- Since the target system will be distributed, it's execution is inherently parallel in nature. The active components concept mitigates the difficulties of concurrent programming such as data loss or deadlocks by shielding individual components from each other and ensuring their internal integrity.
- Supporting multiple departments requires communication using a network and the implementation of suitable communication protocols and formats. Remote communication and service calls are already included in Jadex and can be used transparently, showing no difference between the use of local and remote service calls in terms of implementation details.

As a result, the Jadex Active Components platform is used both as the workflow engine as well as the workflow enactment service for the distributed workflow management system. The next section introduces the Jadex Active Components platform and approach, followed by a section of platform enhancements that were implemented as part of this work in preparation to the implementation of the workflow management system.

## 7.5 Jadex Active Components

The Jadex platform was originally developed as a platform for executing software agents with a particular focus on BDI agent architectures (see [130] and [127]). However, while software agents are useful for implementing certain type of distributed applications, other approaches which are more suitable for particular types of applications are available. The usefulness of approaches for the development of distributed systems is dependent on the type of distributed application because different kind of distributed applications are forced to tackle different challenges which emerge from developing for a distributed system.

Jadex active components is an approach that attempts to unify concepts from agents, components, services and object-oriented programming to address three software engineering challenges (see Figure 7.2): Many software applications and systems are likely to become distributed system. This already occurs once you employ a client/server model but can also include distributed data as they can be found in peer-to-peer or distributed database systems. Another challenging area is concurrency, which allows applications to perform tasks in parallel, potentially allowing for significant performance gains but comes at the cost of synchronizing multiple parallel execution threads.

The final aspect involves non-functional criteria which are aspects of a software systems that, while important, are not directly tied to the functionality of the software. The definition of non-functional criteria are somewhat soft and can depend on the application or even the severity: For example, software performance is often regarded as being part of the non-functional criteria since the execution time generally does not impede the functional results. However, if the performance is excessively poor and delivering results at a point in time which is too late for them to be useful, performance becomes part of the functional criteria.

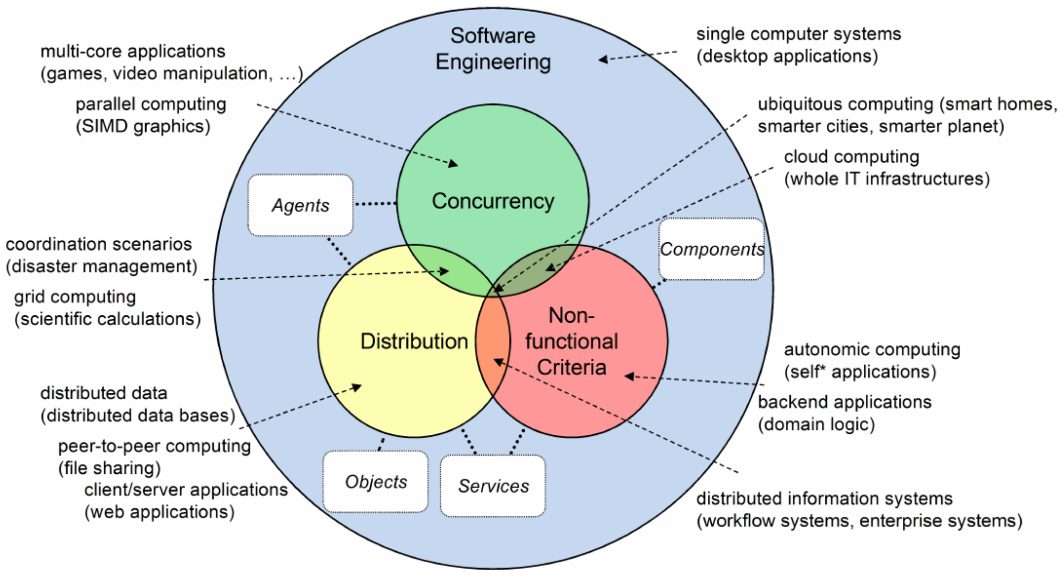

Software Engineering
Concurrency
Distribution
Non-functional Criteria
Agents
Components
Objects
Services
multi-core applications (games, video manipulation, ...)
parallel computing (SIMD graphics)
coordination scenarios (disaster management)
grid computing (scientific calculations)
distributed data (distributed data bases)
peer-to-peer computing (file sharing)
client/server applications (web applications)
single computer systems (desktop applications)
ubiquitous computing (smart homes, smarter cities, smarter planet)
cloud computing (whole IT infrastructures)
autonomic computing (self* applications)
backend applications (domain logic)
distributed information systems (workflow systems, enterprise systems)

Figure 7.2: Challenges, paradigms and applications for the development of complex systems (from [21])

Jadex active components attempts to address all three of these challenges by combining four different concepts that have been used to address one or two of the three aspects, while also maintaining a programming model which facilitates software engineering. First, the concept of agents enables both concurrency and distribution by assembling an application out of multiple distributed agents using asynchronous messages to communicate. Similarly, objects have been used for distribution as well, e.g. through remote procedure calls (RPC) and remote method invocation (RMI) approaches.

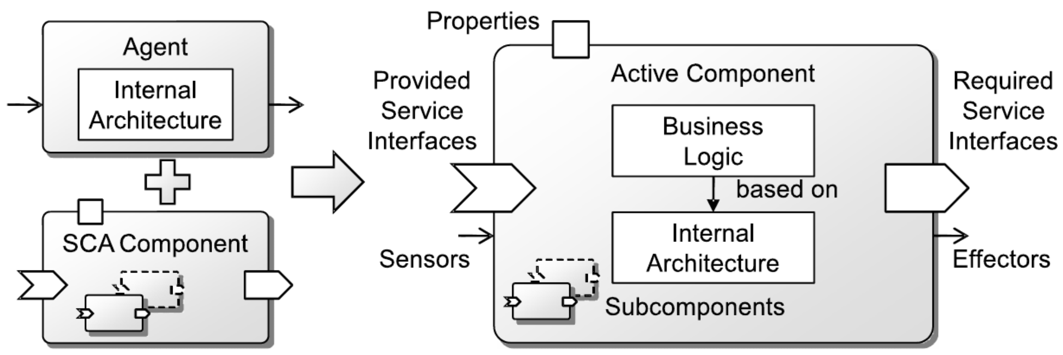


Figure 7.3: Jadex active components combine concepts from the service component architecture and agent technology (from [21])

Services have been used as part of SOA frameworks not only to allow distribution through remote invocation but also to include non-functional criteria such as load-balancing and the use of service-level agreements. In a similar fashion, large-scale application structures called components have been used in order to quickly develop applications. This is accomplished by clearly defining both the interfaces, usage and the functionally provided by each component, thus emphasizing the reusability of components as a non-functional feature.

The active components concepts attempts to unify these approaches (see [131]) in order to provide a solution that addresses these challenges within a software engineering framework. The concept takes the Jadex approach for agent architectures and combines them with a SOA approach called Service Component Architecture (SCA, see e.g. [108]). SCA components consist of four primary features: The properties, which provide configuration information for the component, provided services which are services the component declares to offer and are denoted by incoming arrows and required services which declare the services the component uses in order to function, denoted by outgoing arrows. The component can also include subcomponents which can also offer and use services. The explicit declaration of both provided and required services allows

Jadex adds the agent concept to this approach in order to create active components (see Figure 7.3). Agents contribute the concept of an internal architecture, for example a BDI architecture or a microagent architecture (see [127]) but can also include other architectures such as execution of BPMN workflows. Based on this structure, the application business logic can be added by the developer, which can then offer provided services and use required services as declared.

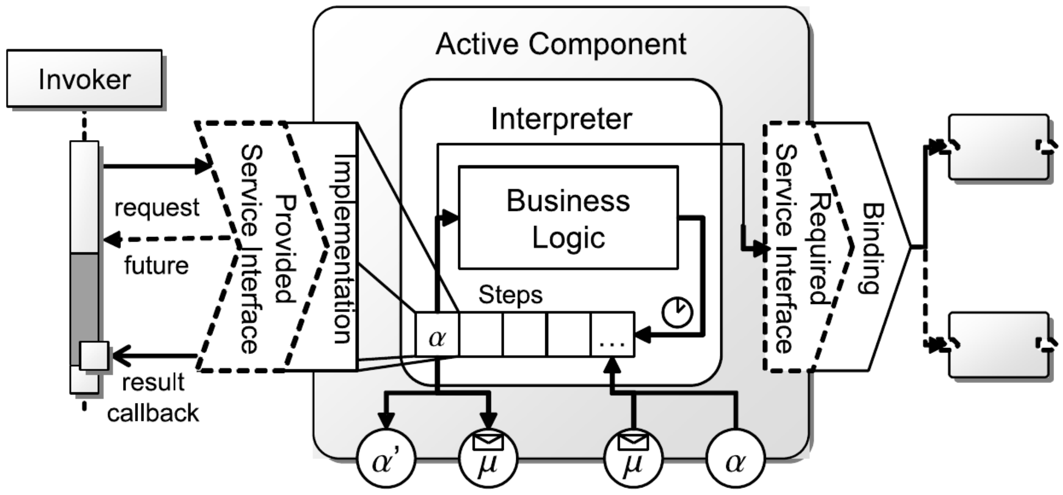


Figure 7.4: Service call behavior and internal execution of active components (from [132])

In order to provide internal consistency and ensure deadlock safety, active components follow the actor model (see [3]): Active components only allow a single thread to be active within the component at any given time (see Figure 7.4), which execute small steps inserted into an internal execution queue. If another component calls a service on the component, a step for handling the invocation is added using a call interceptor chain. The call invocation is then executed on the internal execution thread of the component, thus ensuring that only a single thread is active.

Since the execution of a service call is not instantaneous, the caller has to manage the delay. Simply suspending the thread of the calling component is insufficient since this will freeze the component for the duration of the service invocation and prevents the calling component itself from handling any more service calls and therefore risking a deadlock situation.

Therefore, the calling component uses a future-based mechanism as an asynchronous service call approach. The caller uses a local proxy representation of the service to invoke the Java method on the service. The method then returns a future object which represents a promise of eventually providing the return value (or exception in cases of errors). The caller can then inject a callback object into the future object which will receive the return value of the service call. The call interceptor chain ensures that the callback is executed on the thread of the calling components, again satisfying the requirement of a single active thread per component.

In addition to these concepts, the Jadex platform also provides awareness mechanisms for locating active remote platforms, a service search for finding required services with multiple search scopes (e.g. local to the platform, remote/global) and automatic marshalling and transport mechanisms for communication. The next section will give an overview of enhancement added to the Jadex platform in order to benefit the workflow management system in addition to the concepts already offered by the platform.

## 7.6 Jadex Platform Enhancements

Before the workflow management system can be implemented, additions have been made to the Jadex active components platform to enhance its capabilities. Two factors become important for a distributed workflow management system with the kinds of redundancy requirements as described in section 7.3:

- The distributed nature of the proposed workflow management system means that increased communication data between the various distributed components of the system is to be expected. This is heightened by the redundancy requirements, which in some cases necessitate data synchronization. As a result, the communication between Jadex platform nodes need to be as efficient as possible both in terms of compactness of the messages as well as the performance of transmitting the to a remote platform.

- Some parts of the system need to be able to store data in a persistent manner for later review and analysis and to maintain the data between system-wide shutdowns. For example, the monitoring subsystem must be able to store the observed events that have occurred within the system to allow a review of the data at a later point or to enable long-term statistical analysis. On the other hand, the system is expected to be dynamic, with nodes appearing and disappearing based on reorganization efforts, provisioning of additional nodes to mitigate heavy loads and unexpected outages and errors in some nodes over the long running time of the processes. This means that a global storage system must be available that is able to cope with this dynamic nature by using replication and permanent monitoring to redistribute data based on the appearances and disappearances of nodes.

The next two section will describe two enhancement to the Jadex platform which have been implemented to address these two problems and facilitate the development of the distributed workflow management system. The first section will introduce a new message format for Jadex which aims to improve both the performance of marshalling the data that is being send during service calls and reduce the size of the message that are exchanged. This is followed by the introduction of a distributed general-purpose dynamic key-value storage system which allows the workflow management system to store data in a globally available and dynamically adaptable form.

### 7.6.1 Compact and Efficient Messaging Format

As shown in section 7.3, long-running autonomous business processes require a dynamic and distributed workflow management system. This means that the system is devolved into multiple parts which needs to communicate efficiently with other parts of the system. The demand for efficiency is magnified by the necessary redundancy due to the dynamic nature of the system. As a result, the Jadex platform was enhanced with an efficient message format capable of assisting such a workflow management system (see [83] and [89]).

In a distributed system such as a multi-agent system, the information that is being conveyed between the components of the system has to be encoded in a predefined format to ensure that the information can be understood by the receiver of the message. The structure and form of such message formats can vary to a considerable degree between systems and often involves tradeoffs between beneficial features a particular message format may have. The benefits of such features also depends on the context in which the format is used.

In order to differentiate different design goals and emphasize the ones that are the focus for the compact message format, the following features of message formats were used (from [83]):

- *Human Readability* allows humans to read messages with standard tools like text viewers without the help of decoders or other special tools.
- *Standard Conformance* requires messages to conform to a published message format standard or language standard, allowing interaction between systems conforming to those standards.
- *A Well-formed Structure* defines a valid form for messages, allowing the system to distinguish between valid and invalid messages.
- *Editability* goes beyond human readability by allowing users to edit and restructure messages using standard tools such as text editors.

- *Performance* describes the computational requirements to encode and decode messages.
- *Compactness* evaluates the size of encoded messages.

Message formats can emphasize these features to varying degrees, however, with some features partially contradicting others, no message formats can exceed in all of them at the same time. For example, a human-readable format necessitates a certain verbosity to ease understanding which stands in opposition to the compactness of messages encoded in such a format.

| | **Development** | **Production** |
|---|---|---|
| **Human Readability** | high | low |
| **Standard Conformance** | medium | medium |
| **Well-formed Structure** | high | low |
| **Editable** | high | low |
| **Performance** | low | high |
| **Compactness** | low | high |

Table 7.1: Features for message formats and their importance during application development and when used in production after deployment (from [83])

However, the importance of the features also vary depending on the application or the context in which they are used. For example, Table 7.1 evaluates the importance of the features based on different development stages of software: During the development phase, the correct communication approach of the software has to be determined and the number of errors in the software is still high. Therefore, it greatly assists software development if the message format is human-readable and editable because it allows the developer to debug the application using relatively low-level tools like packet sniffers. A well-formed structure also helps to automatically identify problems in message content by employing automatic evaluation and test tools which validate messages based on the structure. Performance and compactness is less important in this stage since the goal is the development of the functional aspects of the software and the system load consists of the low number of test operations by the developer, so a lower performance or increased bandwidth demand are less of an issue.

This stands in contrast to the situation when the software has left development and is deployed in a production environment. At this stage, the previous development cycle should have been able to identify and eliminate most of the bugs in the software, so aspects like human readability are less of a concern. However, since the load tends to be heavier during production and slow responses are less acceptable, the performance of the format becomes a more important feature of the language. Also, in order to keep infrastructure costs low, the load on the network has to be considered which puts more emphasize on the compactness of the message format.

The usefulness of standard conformance of a message formats depends on the need to interact within a heterogeneous system, where an agreement on a standardized message format becomes a necessity. Therefore, the importance depends on the application context. In case of the workflow management system, the system itself only uses the Jadex platform as its basis. Communication with systems outside the Jadex environment can be done using REST service calls. The communication between components of the system being the primary concern for large loads, the communication efficiency and compactness between Jadex platforms are important in this case, with a lower performance and more standardized communication path is available to interact with other systems.

| | FIPA SL | Java Serialization | Jadex XML | Jadex Binary |
|---|---|---|---|---|
| Human Readability | = | - | + | - |
| Standard Conformance | + | = | = | - |
| Well-formed Structure | + | = | = | - |
| Editable | = | - | + | - |
| Performance | - | + | - | + |
| Compactness | - | = | - | + |

Table 7.2: Message formats in relation to the six specified features with Jadex binary focusing on performance and compactness (from [83])

The goal of the development of the compact and efficient message format called *Jadex Binary* was to emphasize the performance and compactness of the message format first and foremost. Before the development of Jadex Binary, the default message format used by the platform was a format called *Jadex XML*, which uses an XML-based structure to format messages. This meant that the Jadex XML format emphasized the other aspects such as human readability and being editable over performance (see Table 7.2).

As additional reference points for formats with similar emphasizes, two representative formats were used to compare with both Jadex XML and Jadex Binary. First, FIPA SL is a standardized format used to allow different multi-agent platform to communicate with each others (see [53]). It is a highly-formalized language with a clear definition of its structure. Since it is text-based, it is reasonably well editable and human-readable. It therefore is a representative of languages with an emphasize on the first four features shown in Table 7.2.

In contrast, the Java language itself also offers an option for encoding objects into a format through its serialization API. It is the standard approach in Java for serializing objects and both its compactness and especially its performance are very good. However, it has three main disadvantage that make it difficult to use in the context of Jadex:

- In order to serialize objects, the Java virtual machine requires the developer of the class of the object to declare it as serializable by implementing the java.io.Serializable marker interface. This not only applies to the class of the object itself but also to further subobjects contained within the object. However, in many cases a developer may wish to transfer objects based on classes which are not under his control such as classes defined in external libraries. An example of this is the java.awt.image.BufferedImage class included in Java itself. Since it is included, it cannot be changed by the developer and the serialization cannot be extended to include this class

- The serialization does not tolerate any sort of divergences between classes. If the class definitions between two nodes differ even the slightest, the serializer will decline to decode the encoded object. This includes changes such as merely adding a field to a class but can also simply include the use of different bytecode compilers on each node. This requires all classes on each node not only to be in perfect synchronization, even if an omitted field could be tolerated, but it also requires all of it to be compiled using the same toolchain.

- Some classes include redundant data which is cached for quick retrieval but can easily be recalculated if necessary. For example, an implementation of the Java class java.util.Date may include not only the epoch-based field which defines the point in time but also cache value for the day, month, year, etc. in case it is requested to speed up later requests. In this case transmitting this cached information is not beneficial since serializing and transmitting the information often takes longer than recalculating at the destination. However, since the Java serialization is not customizable, the transmission of such information cannot be inhibited.

As a result, the serialization of messages and contained objects require a more flexible approach. Nevertheless, due to its simplicity and integration with the Java virtual machine, it can act as a useful comparison for evaluating Jadex Binary as a format focused on performance and compactness.

Since the focus of Jadex Binary is compactness and performance, it does not use a string-based encoding like Jadex XML. Instead, it uses a binary form in order to encode the input objects that represent the message or the data of a particular service call. The format itself is byte-oriented and uses type and statistical information about the data in order to use shorter encodings for common cases over rarer ones, thus including some form of initial data compression.

The encoding and decoding is based on a bottom-up approach: First, encodings for primitive types are provided which can then be used to represent more complex data object that are based on those primitives.

The first primitive that is considered are integer values. Most byte-oriented approaches to storing integer values often use the same fixed framing as used on the underlying hardware. For example, they may be stored as 32 bit (4-byte) or 64-bit (8 byte) values. In order to allow a wide enough headroom for storing values, usually a fairly large frame is chosen that covers even the most extreme values that may occur.

However, in many cases such as identifier values, this value range is not actually fully exploited and the overwhelming number of values are on the low end of the value range. In order to exploit this, Jadex Binary uses a special encoding for integer values with a variable size. These variable-sized integer values use an encoding that encodes low values in a compact size at the expense of larger values that, in some cases, tend to use slightly more space than in ideal fixed-size formats.

The encoding is based on a similar encoding approach used by the Extensible Binary Meta-Language (EBML, see [162]), which in turn was inspired by the encoding

| Bytes | Format | Value Range |
|---|---|---|
| 1 | 1####### | 0 to 127 |
| 2 | 01###### ######## | 128 to 16511 |
| 3 | 001##### ######## ######## | 16512 to 2113663 |
| 4 | 0001#### ######## ######## ######## | 2113664 to 270549120 |

Table 7.3: Encoding of a complex object in Jadex Binary (from [83])

scheme used in the Unicode UTF-8 encoding (see [148] and [169]). A variable-sized integer consists of at least a single byte, but can contain an arbitrary number of bytes with examples of up to four bytes shown in Table 7.3. The number of bits set to zero, starting from the highest-order bit, specifies the number of additional bytes in the value beyond the first byte. These additional bytes are refered to as extensions and allow the encoding to specify arbitrarily-sized values.

The highest order bit set to one terminates the zeros that specify the number of extensions. The remaining bits of the byte and the following extensions can be used to encode the value itself. Each additional extension adds an additional value range, which starts at the first value following the previous range, thus each value can only be encoded with a specific number of extensions as shown in the figure. The advantage of this approach is that small values use less space while still allowing larger values to be encoded.

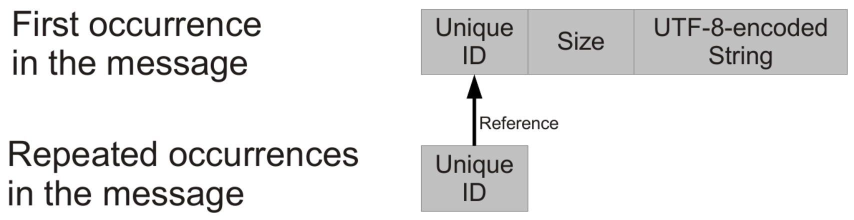

First occurrence in the message
Unique ID
Size
UTF-8-encoded String
Reference
Repeated occurrences in the message
Unique ID

Figure 7.5: Structure of encoded string values in Jadex Binary (from [83])

The first use for variable-sized integers is the encoding of string values as primitive types. On the first occurence of a particular string in the object graph, its encoding consists of three parts: First, the string is assigned a unique identifier using a counter. This unique identifier is encoded as variable-sized integer since most objects contain only a small number of unique strings and the identifier values therefore tend to be low. Nevertheless, the encoding allows for large numbers of unique strings due to the variable encoding.

The next part of the string encoding is the size of the string. This allows the decoder to determine when the string is finished and the next part of the message begins. Since most strings are short, the size of the string is also encoded as variable-sized integer. Finally, the string itself is encoded as UTF-8 and appended to the encoded message. Most text strings tend to consists primarily of ASCII characters which as relatively small to encode in UTF-8, but other character sets are still supported.

If the same string occurs a second time within the same message, it is encoded by simple appending the variable-sized integer of its identifier to the message as a reference to the actual string. Messages are decoded in the order they are encoded with the both the encoder and decoder storing previously found unique strings in a *string pool*. As a result, the decoder can implicitely infer that a particular identifier is a reference because it has already stored the string from its previous decoding.

In addition to these primitive values, the basic encoding library adds support for some additional simple encodings for primitive values. Aside from variable-sized integers, fixed-size integers are also supported in sizes of 8, 16, 32 and 64 bits. Furthermore, 32 bit and 64 bit IEEE floating point values are supported. Both fixed-size integer values and floating point values are put in network byte order (big endian) and added to the message.

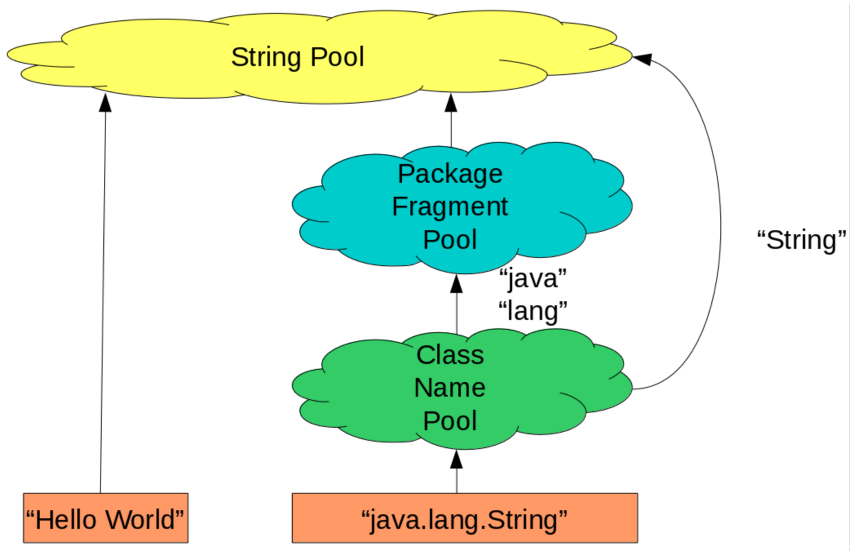


Figure 7.6: Handling of fully-qualified class names in Jadex Binary

The next step after encoding primitives is the encoding of more complex objects. However, when transmitting complex objects, the object type, represented as a Java class, has to be identified. Identifiers for classes in Java consist of two parts, the class name and the package in which the class can be found. The package itself can be any string (with some mild restrictions), but are often structured hierarchically using a dot notation. The package name together with the class name connected with a dot is called a fully-qualified class name. For example, the class in Java representing strings is has the fully-qualified name java.lang.String.

Since most packages contain multiple classes and multiple package contain the same substrings due to their hierarchical structure (for example, many packages included in the Java standard library start with "java."), it can be exploited to reduce the size of multiple occurences of both package substrings and class names.

Figure 7.6 shows how this is accomplished in Jadex Binary. The class name encoding is based on the same string pool as normal strings and can therefore share identifiers. From the perspective of the decoder, the identifier of the full class name is first looked up in a class name pool. If the name is not contained in the pool, it is the first occurence of that fully-qualified class name in the object and a new identifier is assigned. The next part is the identifier of the class name separated from the package name and is looked up in the string pool. This is follows by a list of package fragments that identify the package and are stored in the package fragment pool.

This means that not only are reoccurring fully-qualified class names reused through reference but each class name and package fragment is also assigned individual identifiers. This approach means that substrings of the classes can be shared through those identifiers. For example, the "java" and "lang" part of the package can be shared between the classes java.lang.String and java.lang.Number.

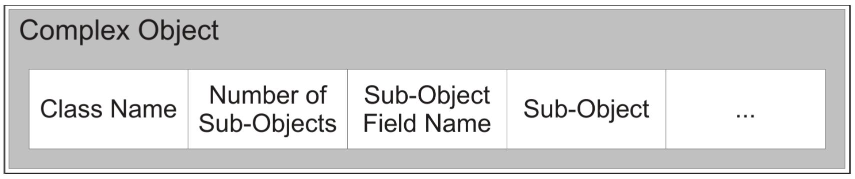


Figure 7.7: Encoding of a complex object in Jadex Binary (from [83])

Both the encodings of primitive values and encodings of class names are available as methods as part of an encoding context which can then be used to encode more complex objects. This is accomplished using a object traverser, which includes a number of encoders or decoders which are used to handle specific objects and employ the methods available in the encoding context for encoding primitives and class names to encode those input objects.

As a catch-all, a encoder and decoder for objects following the Java Beans standard is included which extracts the bean property fields and uses the traverser to encode its subobjects. This results in a structure as shown in Figure 7.7. These objects start with the fully-qualified class name of the encoded class, using the class name encoding as decribed above. This is followed by a variable-sized integer that specifies the number of subobjects included in the object as fields. After the specification of the number of subobjects, the encoded subobjects themselves are appended to the encoding, starting with the field name to which they are assigned.

The subobject itself is recursively encoded using the traverser. This means that each subobject can itself be a complex object and contain subobjects. However, object structures are not necessarily trees but can include cyclic references as shown in Figure 7.8. As a result, object must be interpreted as graphs and the encoding must be able to resolve the occurence of cyclic references.

This is accomplished by maintaining a pool of previously decoded object and assigning them identifier as they are discovered, similar to the approach taken for the string pool. The objects are added to an object pool. If the object is again found in another place in the object graph, the object is references by encoding the previously assigned identifier as a variable-sized integers. This way the cycles can be broken during encoding and the correct object reference inserted during the decoding of the objects.

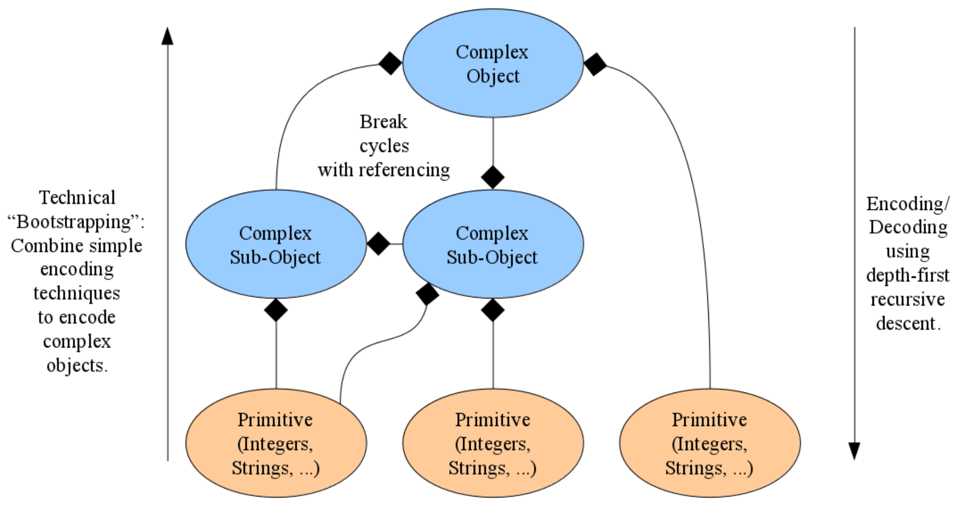

Technical "Bootstrapping": Combine simple encoding techniques to encode complex objects.
Complex Object
Break cycles with referencing
Complex Sub-Object
Complex Sub-Object
Primitive (Integers, Strings, ...)
Primitive (Integers, Strings, ...)
Primitive (Integers, Strings, ...)
Encoding/ Decoding using depth-first recursive descent.

Figure 7.8: In Jadex Binary, complex objects are encoded using a recursively-descending algorithm which uses the primitive encodings are its building block and is capable of resolving reference cycles

In addition to the encoding of complex objects, Jadex Binary also introduces special support and optimization for Java array types. The algorithm differentiates between primitive arrays and object arrays. In primitive arrays, the type of the elements can be inferred by the array type itself since primitive types cannot be subclassed and are always assigned a value. As a result, the encoding of primitive arrays contain the array type, the number of elements in the array and the concatenated values of the elements without any futher type information.

Object types on the other hand require more complex handling: While the array type of object arrays also specify the type of its elements, the actual element can be a subclass of the type specified by the array. Furthermore, the value of the element can be a typeless null value. However, in many cases the element type does match the type defined by the array.

The encoding algorithm therefore attempts to exploit this heuristic by preceding each element with a boolean value that specifies whether the element's type is defined by the array type, in which case the element is appended.If the element does not exactly match the type defined by the array, the boolean value specifies that additional type information follows before the element itself is encoded. As a result, the type information for each element is only encoded if it differs from the array specification.

The resulting encoding of messages was compared to the previously introduced formats with FIPA SL and Jadex XML having goals that differ from the compactness and performance goals of both Java serialization and Jadex Binary. For the compactness test, the test object contained a 514 byte string literal, a randomized long value encoded as string, a single integer value an array of 20 integer literals, an array of boolean values and an array with 100 additional instance of the object itself.

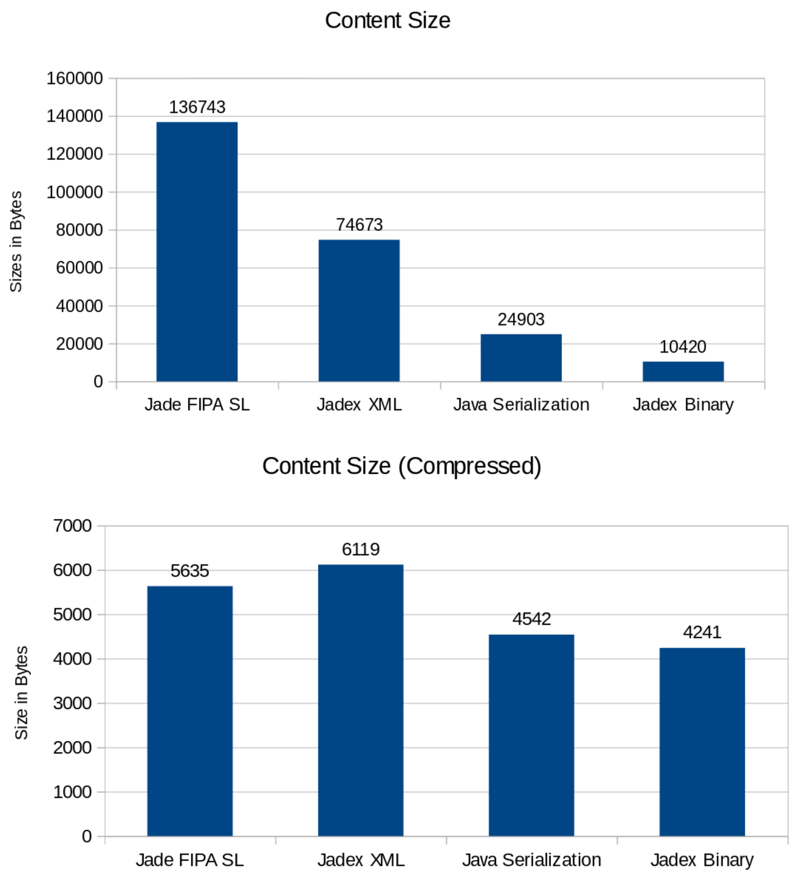


Figure 7.9: Size of a large message encoded using different message formats with (right) and without (left) additional application of the DEFLATE compression algorithm (based on [83])

The first test involved the perofmance of the formats itself without any additional compression algorithm being applied (see Figure 7.9, left half). In terms of compactness, the uncompressed output of Jadex Binary is considerably smaller than the other formats including Java serialization. This can in part be attributed to the redundancy-reducing properties of Jadex Binary.

However, by default, Jadex applies the DEFLATE compression algorithm (see [39]) to the message before they are transmitted to further reduce their size. When the compression algorithm is applied, the differences between formats is reduced (see Figure 7.9, right side), nevertheless, Jadex Binary maintains a lead over the other formats.

For the performance test, the test object was changed slightly to increase the load on the algorithm and reduce variability of the outcome. This was accomplished by increase the number of subobjects from 100 to 100000 subobjects. The tests were conducted using a system containing an Intel i5 750 processor with four cores clocked at 2.67 GHz. The system included 8 GiB of memory, however, the Java memory heap was limited to 2 GiB. The software used to execute the test was the Oracle Java SE 6 Update 31 Java virtual machine running on a Gentoo Linux system running an

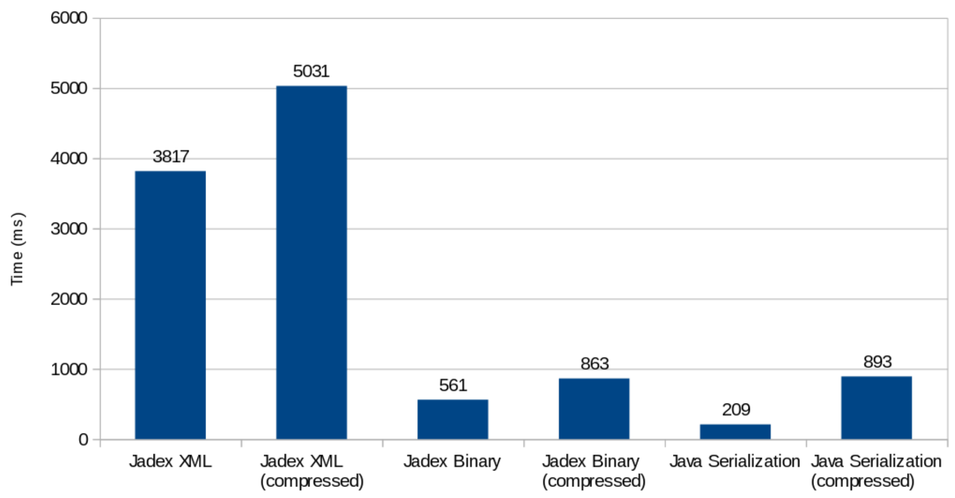


Figure 7.10: Perfomance comparison of message formats with and without application of an additional compression algorithm

unmodified Linux 3.2.2 kernel compiled for the x86-64 instruction set. The tests were repeated twice with the first results being discarded to reduce the impact of just-in-time compilation and lazy object initialization.

The results shown in Figure 7.10 omit the results FIPA SL for clarity, since its uncompressed execution time exceeded 20 seconds ([83] contains the full results). The results show that while the compression is able to reduce the compactness differences between the formats, the larger input size of bulkier formats increase the time for the message to be ready for transmission. In uncompressed form, Java serialization is remains ahead of Jadex Binary, however, when compression is applied, its larger input message size negates its advantage.

As a result, the new Jadex Binary format allows high performance and compactness in message sizes while maintaining the flexibility of the Jadex XML format at the cost of other features listed in Table 7.1. The format thus enables the distributed workflow management system to exchange messages at a high degree of performance and low networking impact. Additionally, when combined with the Jadex streaming extension (see [28]), it allows workflows to quickly exchange even large amounts of data with a low overhead.

### 7.6.2 Distributed Storage Service

The environment in which the distributed workflow management system is used is highly dynamic: Nodes in the system can appear and disappear at any moment either due to organizational restructuring or failures occuring over the long execution time of the business processes. However, the workflow management system is not only responsible for executing workflows but is also tasked with the monitoring aspect of the BPM lifecycle. While monitoring a production system, certain system events occur which need to be stored for use in the analysis phase of the BPM lifecycle. As a result, a reliable long-term storage is necessary to store these events but also in order to offer the workflows themselves a storage service which they can employ to permanently store information.

While nodes can store data locally, the disappearance of a node means that the data stored on that node is lost to the system. As a result, a certain level of replicated data must be created and maintained across the system to ensure that all data can be recovered even when nodes are removed. A number of approaches have been considered in [37] in order to solve this issue and offer a distributed and resilient storage service for Jadex.

One possible approach to create a level of replicated data is the use of a relational database system such as MySQL in a master-slave configuration (see [77]), where the data is distributed among multiple database nodes. In this configuration, two modes of operation are available: Eager replication, which guarantees consistency of the data but introduces additional overhead by requiring synchronized commits between nodes. In contrast, in lazy replication, this overhead is avoided by asynchronously committing the data to the slave nodes but the assurance of consistency is lost since the slave nodes may not always contain the most recent data set.

However, this approach is inadequate for the dynamic environment of the workflow management system: The system has to be manually set up with dedicated master and slave nodes and preconfigured. This means that in case of disappearing nodes, the system cannot fully automatically address the problem by recruiting additional nodes, nor is it capable of automatically integrating new nodes. The approach also replicates all the data on all slave nodes and therefore introduces an unnecessary replication level. Finally, the master node is a critical failure point and slave nodes cannot be automatically promoted to serve as a new master node.

In part, these issues can be attributed in part to the fairly rigid requirements of relational databases. As a result, a number of alternative approaches have been developed under the NoSQL name, which depart from the stringent consistency requirement of traditional SQL databases. Candidates following this approach and which offer some dynamic storage concept include examples such as Amazon Dynamo (see [36]) which is a distributed key-value store and Google BigTable (see [33]) which follows a structured data model concept called Column Family.

However, these concepts tend to have concepts when used in the environment envisioned for the distributed workflow management system. One major issue is the configuration of the systems, which tend to be relatively complex and require the explicit initial definition of parameters before the system can be started including aspects like the number of nodes participating in the system or the definition of master- or super-nodes.

These configuration issues impede the usefulness of such as system in highly dynamic environments. If any node in the system may appear or disappear at any time, the system must be self-configuring and self-organizing in order to represent a viable solution for such an environment. Furthermore, while the systems can cope with the loss of nodes while maintaining data availability, node replacement and integration of new nodes is usually non-automatic and require administrative intervention. Since the workflow management system envisions a decentralized administrative control, it represents an issue for those approaches in this situation.

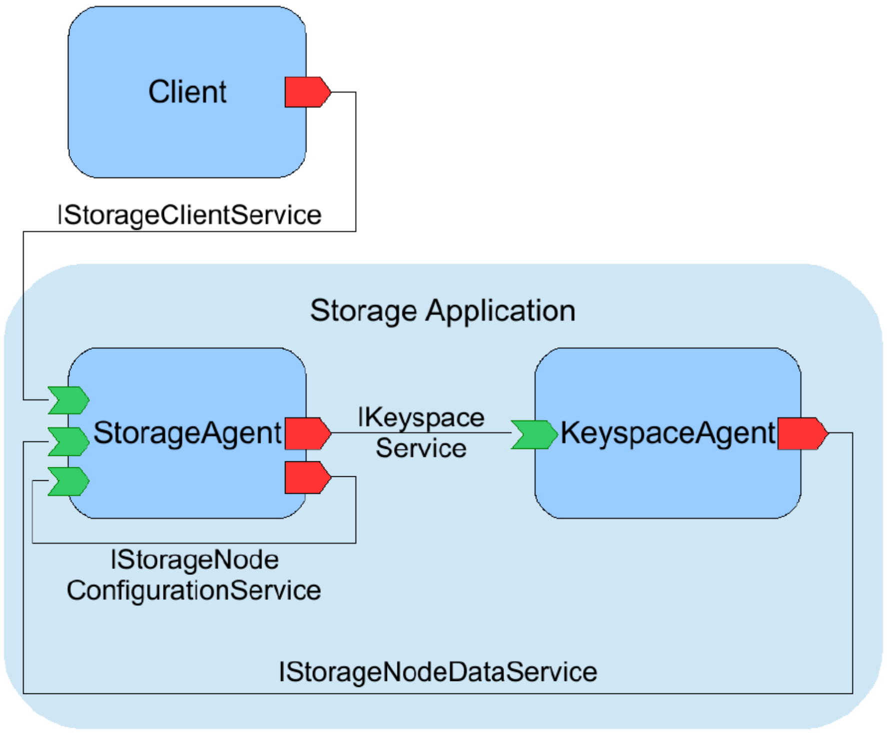


Figure 7.11: Architecture of the distributed storage system for the workflow management system represented as SCA component diagram (from [37])

As a result, a dynamic distributed storage system which can easily integrated in a highly dynamic Jadex environment has been developed (see [37] and [88]). Since the expected data is relatively simple, a key-value approach has been chosen, but an extension to support more structured forms of data can be implemented at a later point. The approach differs from a distributed hash table approach because it assumes that nodes can disappear at any moment.

The architecture of the storage system is shown in Figure 7.11. For clients wishing to use the system it offers a service interface IStorageClientService which the clients can use to access the storage service using Jadex service calls on that service interface.

The service allows clients to store data with the storage service being responsible for both distribution and replication of the data. A specified replication level of the data is maintained even when nodes are added or leave the system. For example, if a node leaves the system, the data of that node is automatically replicated on a spare nodes using the copies of the data still available in the system. The storage is segmented into keyspaces to allow multiple applications and components of applications access to separate data sets and avoid key collision between them.

The storage system application consists of two components which are replicated on every node participating in the storage system, the StorageAgent and the KeyspaceAgent. The storage agent provides two functions for the system: First, it interacts with the storage system clients through the IStorageClientService interface. Second, it coordinates with other storage system nodes using the IStorageNodeConfigurationService. This coordination includes the initial distribution and replication of the data as well as the restoration of the targeted replication level in case nodes are lost.

The StorageAgent component also retrieves data that is stored on remote nodes if necessary, allowing the client transparent access to the data stored throughout the system. For example, if the StorageAgent receives a read request through the IStorageClientService for data that is not stored locally, it will search for the data in other StorageAgent by querying them for the availability of the data. Once the correct node for the data is found, the location is cached in case read or write requests follow the initial one.

The local storage for each storage system node is represented by the KeyspaceAgent. This agent encapsulates the local storage which currently employs an Apache Derby (see [50]) relational database for storage. However, other storage options can be implemented as alternative such as flat file storage system.

As a result, the system is based on a peer-to-peer approach of equal nodes in order to accomodate the dynamic requirement of appearing and disappearing nodes. Howver, the system also supports the occurrence of system partitions where a part of the storage system separates from the rest of the system which can be the case if network lines are lost.

However, due to the CAP (consistency, availability, partition tolerance) theorem (see [57]), a system can only maintain two of the three goals of availability, resistance to partitioning and consistency. Therefore, the storage system has to sacrifice the consistency goal. However, the system uses a vector-clock for each data entry in order to document where consistency errors occurred and allows the user of the system to implement a domain-specific resolver which includes the necessary steps to resolve conflicts of data entries that occurred during a partition event.

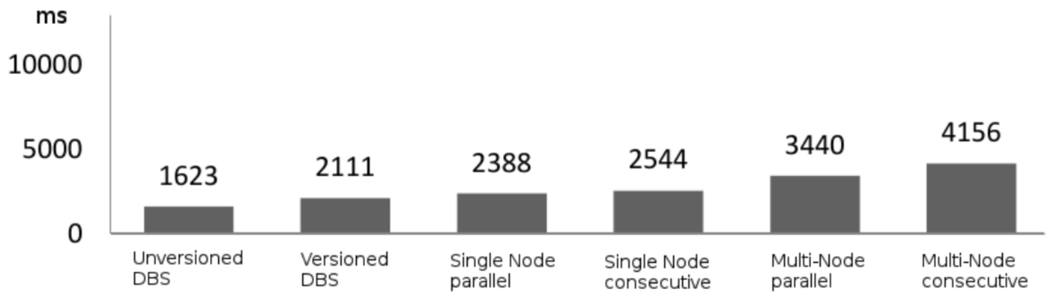


Figure 7.12: Performance of 1000 write requests using both unversioned and versioned local databases as well as different configurations of the distributed storage system (from [37])

Compared to the local storage of data within the nodes, the added coordination and synchronization effort necessary for the storage system is expected to introduce a certain performance overhead. Ideally, this overhead should be kept relatively low without sacrificing any of the advantages the distributed and replicated storage introduces.

As a result, the performance of the storage system has been tested both regarding its functionality and performance. Functional tests included both storage and retrieval in both single- and multi-node environments. Additional tests included the addition and automatic integration of new storage nodes, the deliberate but controlled shutdown of nodes, partitioning of the system and the deliberate termination of nodes through a hard process removal from the local operating system. In each case the system maintained the availability of the data and was able to detect data conflicts using the vector clock in case of the partition event.

In order to ensure an adequate level of performance, a series of tests were performed. Figure 7.12 shows the performance of 1000 write requests when performed on both local databases as representatives of centralized or local storage options as well as the storage system. For the local storage options, two different database setups were used. First, an unversioned database, in which the data entries are not versioned as they are in the storage system. In order to separate the overhead necessary for the versioning, a versioned local database is used as a second reference point.

The storage system was tested in four different setup. First, a local single node storage system is used to show the overhead introduced by the storage system itself. Here, two different access modes are used: A parallel mode, where the writes are issued to the storage system in parallel and a consecutive mode, in which the write requests are issued sequentially after the previous write request has been confirmed. Both access mode tests are then repeated on a full multi node storage system which then documents the performance overhead based on the distribution and replication of the data.

As is shown in the figure, the storage system, as expected, introduces performance overhead due to the vector clock versioning, the self-organization and the distribution and replication. Since the system is capable of handling parallel access request, the parallel access mode tests are more representative of the real world use of the system since it does not include the added latency of the confirmations. While the performance is degraded compared to local storage, the length of a write request is either slightly lower or slightly higher than twice the access time of an unversioned or versioned local database system respectively. Due to the added functionality of resilient behavior in a dynamic environment, this performance loss can be considered an acceptable tradeoff.

This concludes the extension of the Jadex platform for the workflow management system. The next chapter will introduce the overall architecture of the distributed workflow management system, followed by a section about the implementation details of important components.

# Chapter 8

# Distributed Workflow Management System Architecture and Implementation

This chapter introduces the distributed workflow management system aimed at meeting the research goals for long-running autonomous business processes. The system is based on the requirements, approaches and software components described in the previous chapter.

The distributed workflow management system called *Jadex WfMS* has two goals that need to be addressed as part of its architecture and implementation: First, it should provide the functionality of a traditional centralized or semi-centralized workflow management system as defined in the reference model by the workflow management coalition (see [68] as well as section 7.2). However, in order to adequately support long-running autonomous business processes and workflows, the system also has to address the requirements in section 7.3 based on the research goals of the execution and monitoring phases of the BPM lifecycle introduced in section 2.7.

## 8.1 Architecture of the Jadex Workflow Management System

In order to fulfill the requirements, the functionality of the workflow management system has to be modularized and distributed. The parts of the system also have to be able to be replicated not only to increase the resilience over the extended execution time of the workflows using it but also to allow different organizational subunits to perform their own independent administrative management on each part of the system.

As a result, the necessary modularization of the functionality the workflow management system has to address had to be determined. The following parts of the system were identified as necessary in addition to the functionality provided by the workflow engines and features provided by the Jadex platform:

- Workflow enactment and execution: The workflow management system must be able to enact new workflow instances. The creation of instances and execution is already available using the workflow engines and platforms; however, the system must also provide a way for users to add workflow models to the system, maintain them in a repository and provide an interface for enacting them.
- Work item management: If a workflow contains tasks that need to be performed by human workflow participants, work items are created and distributed to the appropriate users as part the workflow management system. The system must provide this functionality in a distributed form that lets workflow instances create new work items that need to be performed, offers a management capability that distributes the items to the right groups of users and ensures that the results of the tasks are returned to the workflow.
- User management: Human users can play multiple roles within the workflow management system. For example they can be workflow participants who perform tasks for a workflow instance. Another group of users are in charge of performing administrative actions in the system such as manually enacting workflow instances or redistributing work items when the original workflow participant becomes unavailable e.g. due to illness or vacation. This means that the system must be able to manage user accounts that are used to authenticate users of the system, assign roles used to distribute work items and define privileges for administrative and other actions regarding the system.
- Monitoring: While the workflow management system is active, information can be gathered about the events occurring in the system. This can include events about workflow instances being enacted or instances entering a state where they are terminating, goals being activated, plans being executed, tasks being performed; also workitems created, assigned and removed. The gathering of such information is critical for the monitoring phase of the BPM lifecycle. As a result, the system requires the means to gather and store the events that occur.
- User access: Users of the workflow management system need to be able to access the system. However, the system must be able to enforce the roles and privileges set by the user management so that each user is only able to use system and workflow functionality in the framework their privileges allow. The system must therefore offer a clearly defined interface which user client software can use while protecting the rest of the system from unauthorized access.

Based on this set of functionalities, the architecture of the workflow management system has been designed as shown in Figure 8.1 and as described in [82]. Each

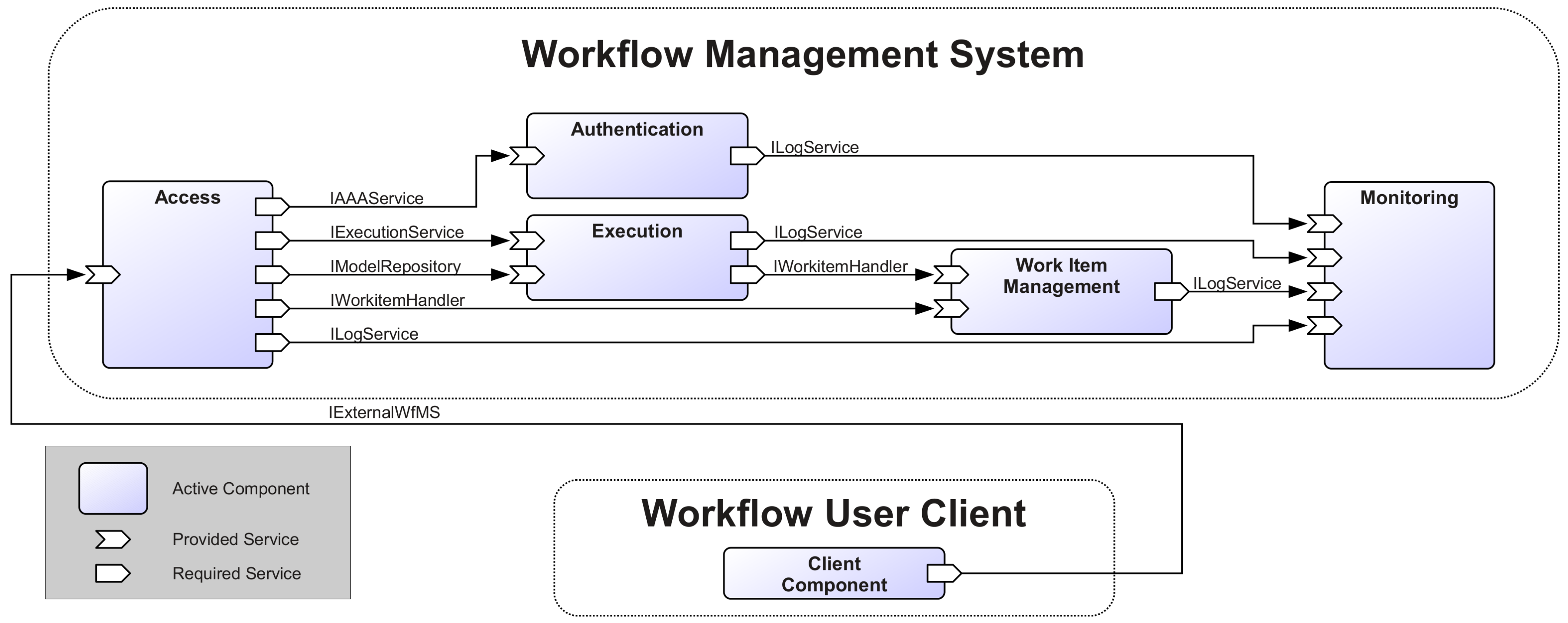


Figure 8.1: Architecture of Jadex WfMS, the distributed workflow management system, each component shown can be distributed, replicated and administered independently (from [82])

functionality has been designed as a Jadex active component using and offering a variety of service interface to interact with the other parts of the system. As a result, the Jadex WfMS workflow management system itself consists of the following five components:

- The *execution component*, which provides the functionality of workflow enactment and execution. The functionality is split into two parts which are represented by two different service interfaces. First, the component manages an internal repository of deployed workflow models that can be used to enact workflow instances. Additional models can be added to or existing models can be removed from the model repository using the *IModelRepository* interface which offers the necessary service methods to perform these functions. Second, while the platform itself performs the actual enactment and execution of the workflow instances, an interface for enacting and terminating workflow instances is provided called the *IExecutionService* interface. This interfaces not only wraps the platform functionality to enact workflow instances as active components but also performs the association between the named workflow models in the repository and the actual models stored on disk, allowing the user to only specify the name used in the repository.

- The *work item management component* concerns itself with the availability of tasks that are performed by workflow participants. The component only provides a single service interface called IWorkitemHandler, which offers the functionality to create, assign, suspend, submit and finish work items. While the work items specify the role or user that is supposed to perform the task, the work item management component is agnostic regarding user management and simply manages all available work items.

- The *authentication component* is tasked with providing the tools for user management. It offers a single service interface called IAAAService. The name is based on the functionalities authentication, authorization and accounting which this components aims to facilitate. The component maintains the user accounts that have been created which includes critical information such as passwords and authentication tokens, roles the user fulfills with regard to workflows and work items, security roles which define the privileges for functionalities of the workflow management system for the users. This component only provides information about users and allows the creation, deletion and modification of user accounts but it does not enforce the defined restrictions on users.

- The *monitoring component* is responsible for gathering events occurring within the workflow management system, storing them permanently and making them available for analysis on request. This component only provides a single service interface, *ILogService*, which allows other components to announce events that have occurred. It also allows the retrieval of stored events for analysis.

Since events about nearly every aspect of the workflow management system are monitored, all other components use the service interface to store events.

- Finally, the *access component* represents the gatekeeper for the workflow management system with regard to human users accessing the system. Users can either be workflow participants, system administrators or may fill both roles at the same time. It offers the *IExternalWfMS* interface which is used to access all the functions of the workflow management system from a user client perspective. The component itself does not hold any state, instead it uses the other components to perform the functionality the IExternalWfMS interface offers. For example, in order to authenticate users, the components uses the authentication component with the IAAAService interface to access the necessary information.

In order to meet the resilience and local administrative requirements for the workflow management system, each component of the system can be replicated and multiple instance of them can be active within the system at the same time. The components self-organize and distribute functionality without additional configuration and each component instance can be maintained by a different part of the organization.

System component instances can be added and removed while the system is active, with missing functions being delegated to alternative instances in case of component shutdown. As long as at least one instance of each component is active, the whole range of system functionalities will remain available. If one component type is missing, the system will have reduced functionality such as being unable to enact further workflow instances if no execution component is available.

The next section will introduce the implementation details of each component as well as more detailed system concepts such as the security model used to shield the system from unauthorized access.

## 8.2 Jadex Workflow Management System Implementation

The requirements for the distributed workflow management Jadex WfMS necessitate that the system can support local administration of parts of the functionality of the system by organizational subunits, addition and removal of nodes by subunits. These requirements are derived from the goal of allowing increased local autonomy within the organization while maintaining many global control aspects of workflow management (see section 1.2). Additionally, in order to support the extensive execution time of long-running autonomous processes, the system should be able to replicate functionality for distribution across multiple networked nodes.

In the previous section, five main components of the system were defined which each represents a certain aspects of the functionality of the workflow management system. This chapter will provide further implementation details regarding each of the components.

### 8.2.1 Execution Component

The execution component encapsulates two related functions based on the WfMC reference model (see section 7.2): First, while the Jadex platform itself already provides the means to enact and execute both GPMN and BPMN workflow instances as active components (see section 6.5), the interface for enacting them includes a degree of complexity which is unnecessary for workflow enactment. The execution component abstracts from the lower level layer for executing active components on the Jadex platform and offers a layer with an interface for enacting workflow instances which then passes the execution details on to the platform. This part of the functionality refines the platform features into the workflow enactment service as defined by the WfMC workflow reference model.

The second function consists of the management of deployed workflow models which are used to enact instances. This functionality is the connection to the process definition tools and represents the functionality of interface 1 in the WfMC workflow reference model. The execution component combines these aspects since they tie into each other in a way that becomes relevant in a distributed system: The local node or platform is only able to enact workflow instances with models that are available locally. Therefore, the workflow model must be deployed locally through the definition service. It is generally not useful to deploy the workflow model in one node in the distributed system and then attempt to enact a workflow instance of it on another node since the model would have to be transferred before it can be executed which represents a form of ad-hoc deployment on enactment.

This issue does not arise in a centrally-organized workflow management system since the workflow models can only be deployed in the same location as they are executed. In a distributed workflow management system, it is therefore advantageous to combine the two functions and allow deployment on the nodes where they are used.

However, the execution component follows the reference model in that it splits the two function between two service interfaces, the IModelRepository for deploying and undeploying workflow models and the IExecutionService for workflow enactment. The IModelRepository offers three methods in order to manage the models deployed on an execution component. The addProcessModel() method allows the addition of new process model to the component's process repository. Users can specify an identifier for the model which is later used to refer to the model. The model itself can either consist of a plain process definition file or it can consists of a packaged .jar file which can also contain additional subprocess models and classes for further functionality that is used by the process. The deployed process model is then stored in a configurable location on the local persistent storage medium and an entry is created that references the model identifier used during deployment with the deployed files.

Workflow models deployed in such a way can then be enacted as workflow instances using the IExecutionService interface. Here, three methods are used: First, the startProcess() method receives the model identifier used during deployment as input. The service then locates the associated model files for the process and enacts the main process file as an active component representing the workflow instance on the Jadex platform. The method returns an instance identifier that represents the running workflow instance.

Most business processes have a clearly defined point at which they are considered to be finished. Workflows based on such processes will terminate themselves by shutting down the active component instance used to represent them. However, if a workflow requires manual termination or becomes unable to terminate itself due to an implementation or modeling error, the IExecutionService interface offers a terminateProcess() method which will initiate the termination of the workflow instance if provided with the instance identifier. Finally, the IExecutionService also offers the capability of requesting a list of active workflow instances by calling the getProcessInstances() method.

If a workflow model is no longer required, it can be undeployed by calling the removeProcessModel() method of the IModelRepository interface. This method will remove the deployed model from the repository and further enactments of workflow instances based on that model will not longer be possible. However, workflow instances based on the model which are still active will remain so until they are terminated. Consequently, the files of the model will only be removed once all workflow instances based on the model have terminated. The final method of the IModelRepository service is the getProcessModels() service call method which is used to retrieve a list of all locally deployed workflow models.

As a result, the execution component offers two interfaces that are used to both deploy workflow models and enact workflow instances on nodes where it is being executed. Since the repositories are decentralized, each organizational subunit can maintain its own repository of workflow models as required, increasing their autonomy with regard to workflow model management and enactment.

### 8.2.2 Work Item Management Component

Many workflows do not represent fully-automated business processes since not all tasks of a business processes can be automated. For example, while most data processing tasks can be performed automatically by a machine, customer counseling or customer service requests are usually processed by an employee. As a result, business processes involving such tasks are partially automated and rely on human participants to carry out certain work orders. In a traditional workflow management system, this is accomplished with the creation of work items: For example, if a BPMN process reaches a human task within the workflow, a work item containing a description of the necessary task is created for the task and the process thread is suspended at that point until the work item has been handled and is returned to the process.

Since organizations contain employees with different sets of skills, it is not useful for a work item to be processed by random employees. While directly specifying a specific employee by name within the task is possible, it is often avoided since the specified employee may be unavailable due to vacation or sickness. Furthermore, in many cases tasks can be performed by groups of employees and it is desirable to distribute the workload evenly between them. As a result, workflow management systems often use the concept of a role for assigning work items to people: Workflow participants are assigned one or more roles such as ”customer representative” or ”sales managemer” depending on the position in the organization and their skill set. Human tasks in workflows are then assigned roles depending on the skills or authority required. In BPMN, this is usually accomplished by defining a lane with the role name and placing the BPMN activity in that lane.

When a such a task is executed in a workflow instance, the workflow management system creates a work item and marks it with the specified role. The workflow management system is then able to assign it to any available person from the pool of participants that are in the specified role. This is usually performed using two distinct mechanisms: The work item can either be assigned automatically to an available person with the role (”push”) or the workflow management system offers it as an option to eligible people active within the system who can then elect to assign the work item to themselves (”pull”). From a workflow management system perspective, once a work item has been assigned to a particular person, it becomes an *activity*. As a side note, this activity is distinct from the BPMN activity element (see [68]).

Therefore, the workflow management system must be able to support the management of work items and their assignment to specific users, which in the current implementation solely consists of the ”pull” type of arrangement though an alternative implementation would be possible. However, the requirements of the distributed Jadex WfMS also mandate that the work item management component can be replicated on multiple nodes which can be activated or deactivated at any time. This means the work item component has two options regarding the availability of work items in the system:

- The work item is only created and available on a single work item management component, often on the same node where the workflow instance has been enacted though a separation of the functionality is possible (*unitary mode*).
- The work item is created on single work item management component, then distributed among all available work item management components in the system (*replicated mode*).

Both modes have different advantages and disadvantages over the other whose importance depends on the specifics of the workflow as well as the business environment: In unitary mode, only a single copy of the work item exists. This means it can only be requested by a single user, after which it becomes an activity until the user finishes the task, relinquishes the work item or until the administrator removes the assignment manually. Meanwhile, the work item will be filtered out with regards to other users who will subsequently not be able to request the work item. Therefore, in the unitary mode, processing of work items by participants is transactional and redundant duplicated work is not performed. Additionally, if the work item management component is active on the same node as the execution component, the workflow can continue as soon as the work item results are submitted by the workflow participant.

However, a major disadvantage of this mode becomes apparent when the disappearance, even temporarily, of nodes is considered. If the node with the work item management component containing the work item disappears, the work item becomes unavailable. If the work item is already assigned to a participant, the participant will be unable to submit the results of the work item until the node returns. Nevertheless, work items on other work item components remain available. This means this failure mode could be regarded as a functional degradation in the face of a fault which does not influence other parts of the system.

The replicated mode on the other hand avoids this problem: Once the work item is completed, it is distributed among all available work item management components. This means that even if the original node where the work item was created disappears, workflow participants can still request the work item, perform the work and submit the results.

On the other hand, this enhanced availability of work items comes with a price: If a node or multiple nodes with work item management components get separated from the system, workflow participants in both fragments can request the work item without the other fragment being aware of the assignment. This means that the work specified by the work item may end up being performed more than once.

However, the workflow instance itself will only accept a single result for a work item before proceeding, with the result arriving at a later point being discarded. As a result, the functional degradation in the replicated mode can result in redundant work being performed instead of work items becoming unavailable.

Both modes can be advantageous in different situations and depend on the context of the workflow. The implemented work item management component therefore supports both modes, with a default being set in each execution component which can be overidden by a property set in the workflow model for each human task. As a result, the workflow designer may choose between the modes for each defined human task and can select the one most appropriate for the situation.

Creation and distribution of work items as well as submission of work item result is managed by the IWorkitemHandler service, which has the following methods available to perform those functions:

### 8.2.3 Authentication Component

Users of the workflow management system can have two main interactions with the workflow management system: First, they can be part of workflows as workflow participants who request, fulfill and submit work items as they are generated by the workflow instances. Second, users can perform administrative actions with regard to the system such as the manual enactment of workflow instances and the manual removal or a work item assignment. However, not all users should be able to perform all the functions and request all work items available in the workflow management system. Additionally, unauthorized persons should not be able to interact with the system at all. As a result, the workflow management system must have a user management that maintains such information about the user as well as providing an authentication mechanism which can be used by the user to authenticate themselves to the system.

This function is provided be the authentication component: The system maintains accounts of users of the workflow management system including all the information associated with such accounts. The first important information are secrets that are used to authenticate users. The current implementation of this functionality is based on passwords, however, alternative approaches based on identifying information like secret keys could be implemented.

The second set of information regarding users pertains to their roles within the system. In this regard, one has to differentiate two kinds of interactions with the system which result in two different sets of roles that can be assigned to users:

- *Workflow roles* or simply *roles* refer to the potential participation of a users with regard to workflow instances that are enacted on the workflow management system. For example, a user may hold the workflow role ”technical support” which means that the user can participate in workflow by requesting work items that the workflow specifies as being suitable for this role. As implemented, workflow roles can contain other workflow roles as subroles and each user account can include multiple roles. When the role of a user is resolved, a set of roles is created that contains all the roles and their subroles as defined by the user account. If the role specified in the work item is contained within that set, the work item becomes visible to the user and can be requested from the work item management component.

- *Security roles* define which actions the user can take with regard to the workflow management system. In order to specify these roles, every function the workflow clients can invoke on the workflow management system such as enactment of workflow instances or removal of a work item assignment has an associated *privilege*. Security roles can contain one or more privileges which means that a user account which includes that security role is able to use the functionality defined by the privilege. Similar to workflow roles, security roles can include subroles and user accounts can be assigned multiple security roles. Privileges of a user are determined by first creating a set which includes all security roles and subroles, then forming a union of all privileges assigned to those security roles. The use of a particular functionality is permitted if the associated privilege is contained in this union set.

The authentication component itself does not enforce the authentication of users nor their respective authorization regarding privileges. It merely contains the functionality for authentication and the information regarding both workflow and security roles and exposes them using the IAAAService interface. This interface offers a number of methods for retrieving information about users and manipulating the user accounts managed by the component. First, the createUser() and deleteUser() method allows the creation and deletion of user accounts, both methods expecting an account name as input.

Accounts created in such a way are assigned a random password to prevent abuse, so a second call setting the password of the new account is required. This is accomplished through the modifyUser() method which includes three arguments: A set of workflow roles, a set of security roles and finally the secret used to identify the user. Any of the arguments can be set to a null value which causes the respective information to be retained rather than overwritten.

Four methods are available to create and delete both workflow and security roles called createRole(), createSecurityRole(), deleteRole() and deleteSecurityRole(). The methods for creating and deleting workflow roles each have a parameter specifying the role name with the createRole() method also including a set of subroles which may be empty.

The createSecurityRole() method also includes an argument for the name of the security role but also has two additional arguments, one for defining the set of subroles similar to the argument used in the createRole() method and a set of privileges that are granted to the security role. Both of the latter arguments can be left empty, however, defining a role with neither subroles nor privileges confers no privileges to associated user accounts and is therefore of limited usefulness. The deleteSecurityRole() method removes a security role from the system.

In order to query the workflow roles of a specific user account, two methods are available: First, the getRoles() method which returns the roles specified in the user account if the user account is supplied as an argument. Second, the getAllRoles() method returns not only the workflow roles of the specified user account but also resolves all the subroles and returns a union set of rules that includes all roles explicitely or implicitely granted to the user. The former method is primarily used for managing used accounts while the latter one is used to identify work items that the user is qualified to process.

A similar approach is used with regard to the security roles. The method getSecurityRoles() returns the security roles of a user account which can then be used with the getSecurityRolePrivileges() method to query the particular privileges of the security role as well as the getSecurityRoleSubroles() method in order to query the subroles of a particular security role. In addition, the getUserPrivileges() method is available which returns all the privileges granted to a particular user based on the assigned security roles and their subroles.

In order to support the requirements of the distributed workflow management system approach, multiple instances of the authentication component can be active within the system, each being potentially maintained by different subunits of the organization. Each of the authentication components contains its own set of user accounts, workflow roles and security roles and can be administered separately. This allows each of the organizational subunits to autonomously maintain their own set of user data based on their workflow participants and other members of the organizational subunit.

Since multiple instance of the authentication component can be inserted into the system, there is the possibility that two or more of them may contain conflicting information regarding particular roles or users. As a result, an approach has to be chosen to resolve such conflicts. While one of the requirements of the system is to grant additional autonomy to organizational subunits, the current implementation of the authentication component presumes that the system is still used in a generally cooperative environment.

As a result, the current implementation of the authentication component is based on an inclusive approach for dealing with conflicting information: If a single authentication component is available that is willing to authenticate a user, declares the user to be able to perform a certain workflow role or grants a user a certain privilege, it is assumed this information is correct even if other authentication components are unable to do the same. However, if necessary the component could be extended to implement a more restrictive approach.

As mentioned, the authentication component only provides the necessary information and mechanisms for authenticating users and authorizing their actions. The enforcement of both the authentication and authorization is performed by the access component which is described in the next section, which will also describe the security approach used to shield the internal workflow management system components from external user clients accessing the system.

## 8.3 Access Control and Security

In general, users of workflow management systems either use workflow client software or a web user interface to access the functionality of the system. Only authorized users are supposed to access the system while also being restricted depending on the workflow roles and system privileges granted to their user accounts. As a result, workflow management systems tend to include approaches limiting user access to the system in order to enforce these requirements.

Using the default configuration, Jadex allows local service calls between services provided by components on the same system while disallowing remote invocation of services unless the service is specifically flagged as a public service that allows remote calls, unless a secret password is used which allows access to the remote platform.

However, Jadex also allows tailoring access security based on a virtual network concept (see [26]), where a platform can participate in trust networks which are defined by a network name and an accompanying secret that allows access to the network.

This mechanism is used to restrict access to the workflow management system (see Figure 8.2). First, two virtual networks are defined which separate the security domains: A client network which contains the workflow clients which allow users to access the system. This can either consist of local workstations or web servers containing a web-based workflow client. The second network is the system network which contains all components of the workflow management system functionalities.

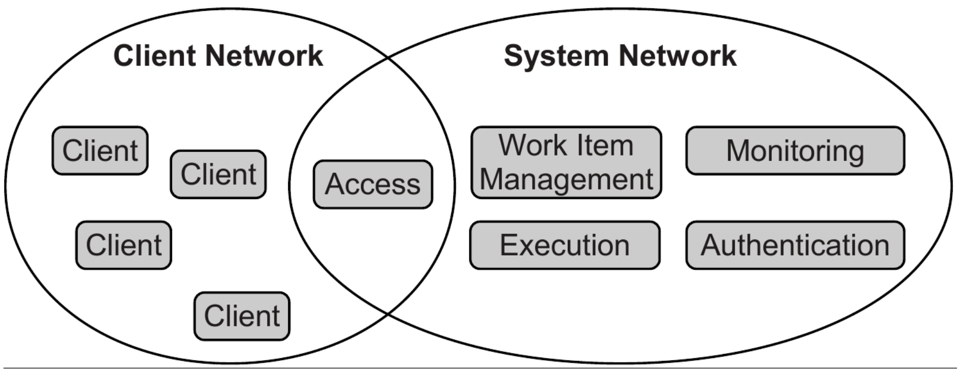


Figure 8.2: Enforcement of access is provided by the access component in Jadex WfMS, which is used in combination with the security network concept of the Jadex platform to shield the internal components of the system from the external workflow client network (from [82])

Access to this network and knowledge of the network secret is restricted to persons in charge of maintaining the workflow management system for their organizational units.

As a result of this separation, the user workflow clients cannot directly access the service interfaces of the workflow management system components, nor can the system components access any client workstations or servers. The next step is to provide a controlled path between the two networks which allow restricted access based on the user account definitions provided by the authentication components. This controlled access path is provided by the access component. The access component is executed on a special node in the system which participates in both the system and the client networks.

The access component provides an interface which the workflow clients can use to access the system. Since the access component participates in both the system network and the client network, it can be reached by the workflow clients and can delegate requests to components of the workflow management system. As a result, the access component is the gateway between the two networks. Due to this position, it can restrict access based on user account information provided by authentication components which it can access through the system network.

Figure 8.3 demonstrates how the authentication component assists the access component by providing user account information. When a user uses the workflow client to connect to the system, the client software uses the service interface of the access component to authenticate the user to the system. The access component uses the provided authentication information and calls either a local authentication component or, if no local one is available, any of the remotely available ones and requests to authenticate the user. The called authentication component can then either authenticate the user directly if the user is known to the component locally, or can successively call other authentication components in the system and request authentication.

If none of the authentication components can authenticate the user, the authen-

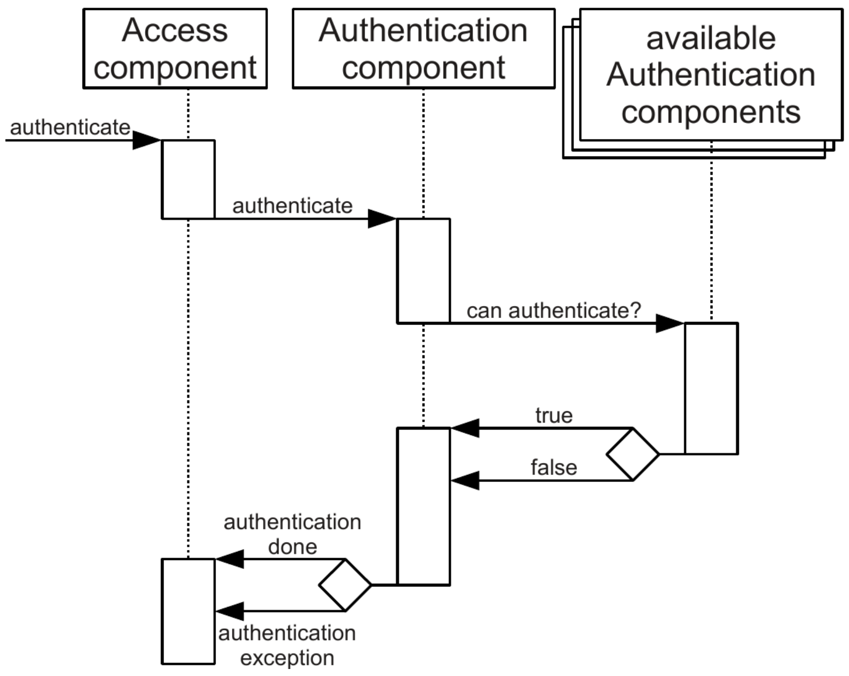


Figure 8.3: Authentication interaction using the access component which employs the authentication components to authenticate users (from [82])

tication request is denied, otherwise the workflow client is authenticated to be in use by the authenticated user. In either case, the information is communicated through the access component to the connecting workflow client, which can then report to the user whether the authentication was successful and the user is logged into the system or, in case of authentication failure, provide an error message which offers the user to an opportunity to correct the provided authentication information.

Once a user workflow client has been successfully authenticated, all further interactions with the system are still passed through the access component, which allows the access component to restrict access to the system based on user privileges. For example, Figure 8.4 shows the interaction of a workflow client with the system when requesting a work item created in unitary mode. The request is first received by the access component. The access component then has the opportunity to verify the user privileges and workflow roles with the authentication component to assure proper authorization of the action requested by the user.

If the user is authorized to request a work item, that request is delegated to the local work item management. Since the work item is being processed in unitary mode, it only exists once in the system. This means that the work item management may either have it available locally, in which case it can be assigned immediately, or, in case the work item is available on a different work item management component, the local one calls the component storing the work item and requests the transfer of the work item. The work item is then transmitted to the local work item management, after which it is stored by work item management component that is used by the requesting user. The work item can then be locally assigned to that user and the

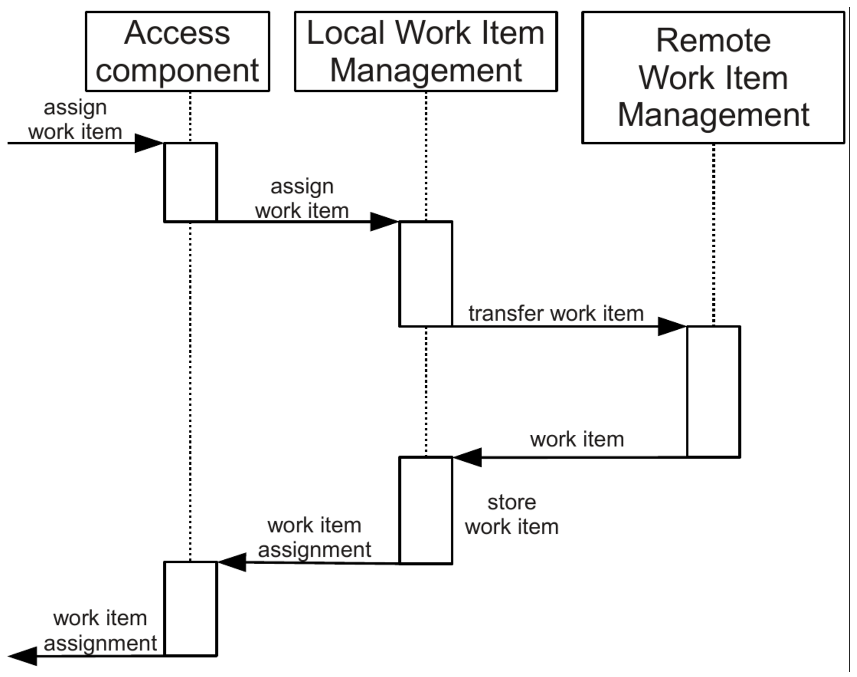


Figure 8.4: Accessing a work item through the access component in unitary mode (from [82])

request is finalized.

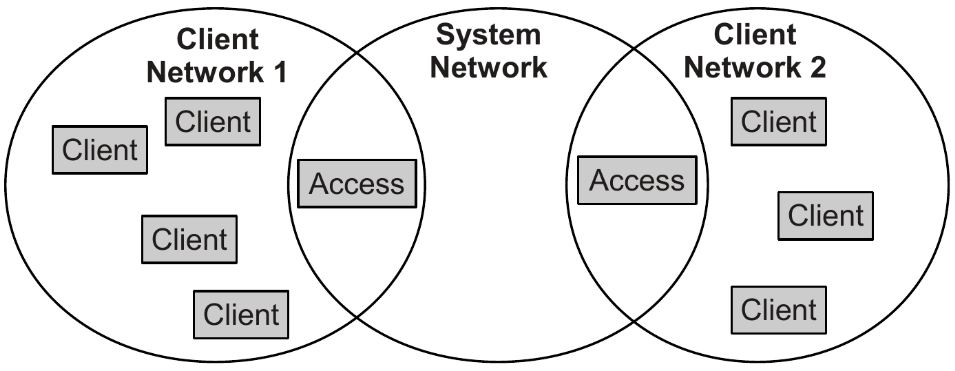


Figure 8.5: Support of multiple client networks for different organizational units facilitated by separate access components (from [82])

In addition to controlling the access to the workflow management system and enforcing roles and privileges, the access component also allows the separation and isolation of workflow clients and autonomous management of workflow client workstations and systems by organizational units. Figure 8.5 demonstrates such a scenario. Two organizational units may need to be able to set up workstations and access points for their employees in order to avoid unnecessary bureaucratic actions on transfers or change of personnel.

This can be accomplished by separating the single client network into multiple independent ones. For each separate client network, a separate node is defined with an access component that acts as a gateway for the client network. Both client networks in this scenario have independent network secrets which allows local management of that client network using its secret without influencing other networks. Despite this separation, all workflow clients still interact with the same workflow management system and can share workflow instance, roles and other resources.

In conclusion, the access component of the workflow management system in combination with the secure virtual networks allow the restriction of workflow client access to the system based on the user information stored in the authentication components. Furthermore, it enhances local autonomy of the overall system by allowing the creation of separately managed and isolated workflow client networks while still maintaining the global control of the shared workflow management system.

The next section will proceed to the analysis phase of the BPM lifecycle. For this phase, detailed monitoring of current and former workflow instances is required, which must be recorded and stored in a persistent way for later analysis. Therefore, the monitoring approach of Jadex WfMS will be introduced, which includes not just the monitoring component of the workflow management system but also the structure of the events used to record occurrences within the system.

## 8.4 Workflow Monitoring

The monitoring component of Jadex WfMS is primarily responsible for supporting the monitoring phase of the BPM lifecycle described in section 2.7. In section 1.2 it was argued that the monitoring phase touches two of the research goals for long-running autonomous business processes: On the one hand organizational agility must be supported by allowing organizational subunits to set up and perform their own monitoring independently. On the other hand the system must be sufficiently robust to reliably monitor the long-running workflow instances executing on the system.

However, the purpose of the information gathered during the monitoring phase is not only to provide real time feedback concerning active workflow instances but also to facilitate the analysis phase of the BPM lifecycle in which the workflow is inspected and analyzed. As a result, while the monitoring component itself should meet the goals necessary for the monitoring phase, the information gathered must support the goals necessary for the analysis phase: Here, a key aspect regarding long-running autonomous business processes is to link action that the workflow performed during execution with the strategic goals of the business process. This has to be done despite the flexible execution paths of the workflows involved.

In order to reach the goals of robustness and organizational agility, once again the monitoring component is implemented to support distribution and replication of multiple instances (see [84] and [88]). The component provides a service implementing the ILogService interface which offers the logEvent() method allowing all parts of the workflow management system to add new events to the system. In the first implementation of the monitoring component, replication and distribution was achieved by including a distributeEvent() method in the service interface. This method was used by the receiving monitoring component to distribute the received event to the remaining monitoring components in the system.

Since events can either be transmitted to a monitoring component or lost, there can be no disagreements about the content of events, restricting inconsistencies to missing events instead of conflicts even in the case of partitioning. This means that the system represents a BASE (Basically Available, Soft State) approach that can avoid conflicts due to the type of data being recorded.

However, the monitoring component being responsible not just for recording but also distributing events had two disadvantages: First, the components used local storage which meant that the data was replicated in each of the monitoring component nodes when a lower number of replications would be sufficient to ensure robustness. Second, it caused a high amount of traffic since the events were transmitted to every monitoring node in the system.

As a result, the monitoring component was revised to be based on the distributed storage system described in section 7.6.2. The enhanced monitoring component receives the event through the service implementing the ILogService interface, then proceeds to store it in a shared keyspace within the distributed storage system. The storage system performs the necessary replications and maintains them in case of disappearing nodes and allowing other monitoring components to access the events through the system. This approach means that the number of replications can be set in such a way that it is high enough to maintain a reasonable level of robustness but low enough to avoid unnecessary overheads.

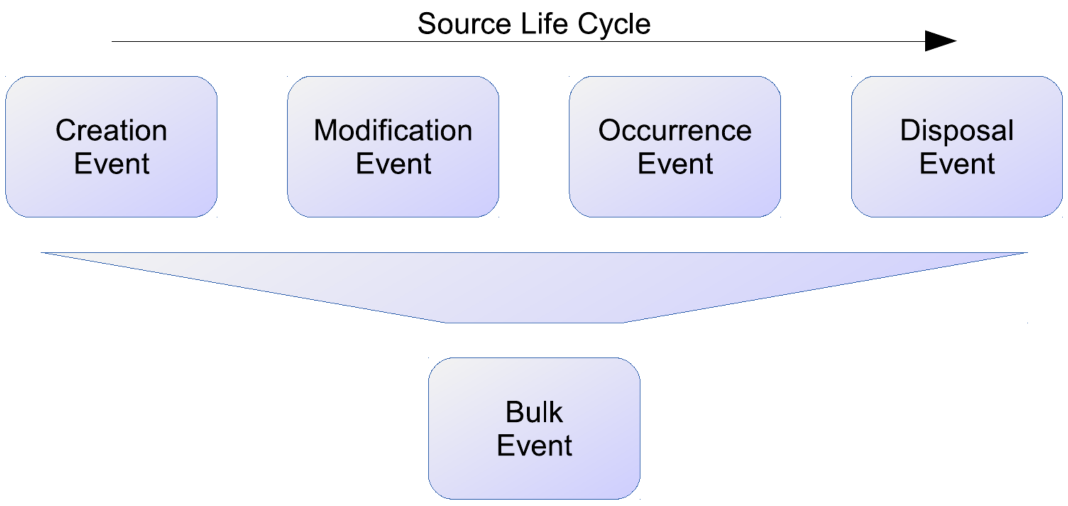


Figure 8.6: Event types used in the monitoring system with the bulk event type used to agglomerate a large number of events for bulk transfers (from [84])

As mentioned, the monitoring system must not only ensure the availability of the gathered information but also include the right information to allow a connection with the strategic planning using the workflow model as the basis. The monitoring of the workflow management system should cover a wide aspects of the workflow execution. Therefore, most aspects that are involved in the execution of GPMN workflows are capable of recording events that have occured. Such aspects include any part of the system that can be instantiated or executed like component instances, goal and plan instances, task, work items and so on. Each of these aspects can therefore be a *source* of an event, which is part of the information stored within the event. The event also includes a timestamp of the point in time when the event was created.

In order to further refine the events that can be issued by the event sources in the system, five event types were defined which cover the range of recorded events in the system (see Figure 8.6). As a result, all events that are recorded by the system are one of these types (see [84]):

- *Creation events* denote the creation of the event source. For example, if a workflow is enacted, the first action of the resulting component instance is to record a creation event with the monitoring component that describes the creation of this component instance.
- *Modification events* can be issued by an event source in order to record a state change of the source. A typical example of this are changes of the business context of GPMN workflow instances.
- *Occurrence events* simple denote when an action took place. For example, if a message is received by a component instance or if an internal user event is triggered within a component an occurrence event is recorded to document it.
- *Disposal events* denote the end of the life cycle of a source. When recorded, it denotes the termination of the source with the implication that no further events can be expected.
- Finally, *bulk events* represents a special type of event that is only used to agglomerate multiple events into a single event for bulk transfers. This reduces overhead if a large number of events need to be transfered. A common case are bulk transfers of events for analysis.

Unlike task-based modeling languages, GPMN does not provide a fixed execution path except as part of the BPMN fragments in plans. As a result, during execution it is hard to associate the action being performed with the goals that are part of the GPMN model. The events should therefore include sufficient information to provide the user of the system with a view that connects the actions performed as part of the workflow with the business goals defined in the workflow model.

However, since the monitoring part of the workflow management system is distributed, the events need to be replicated and distributed to ensure availability. This means multiple copies of events are transfered which combined with the fine granularity of the events being monitored means that the events should be as compact as possible to keep the data being transmitted at a minimum. While the compact message extension described in section 7.6.1 helps to reduce message overhead, the information stored within the events should also be as small as possible.

As a solution for this problem, events also include a cause reference in addition to the source reference. While the source reference denotes the origin of the event,

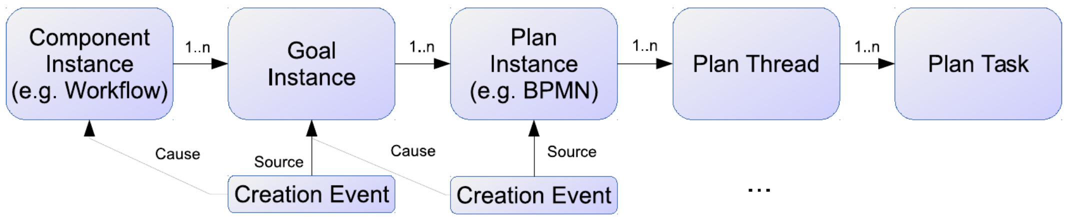


Figure 8.7: Example of the source-cause hierarchy used to establish causalities between different events (from [84])

the cause reference signifies the reason why a particular event has occurred. This can then be used to create a source-cause hierarchy which allows the association of event sources with their reasons as shown in Figure 8.7 while only storing one additional reference in each event. This means that the events themselves are small in size while still allowing the association of low-level actions with strategic high-level goals.

For example, if a GPMN workflow is enacted, a component instance representing the workflow instance is created. This workflow instance then activates the primary workflow goals which creates a workflow goal for each of them. This goal instance creation is recorded by storing a creation event with a reference to the goal instance as the source. However, the creation event also contains a reference to the workflow instance as the cause for the event. This denotes that the cause for the creation of the goal instance was the workflow instance.

This pattern repeats down the cause hierarchy: Once plan instances are created and executed, creation events for those instances are also created with the cause of the plan instance creation being the goal that is connected by a plan edge. However, the monitoring of events extends beyond plan execution. While a Java-based plan appears monolithic, BPMN plans allow additional granularity.

The next level in the hierarchy within BPMN plans are the plan threads which denote execution threads that are being followed as part of the plan. Based on the BPMN execution model, plans start with a single plan thread but additional threads can appear in case of forking behavior such as the one triggered by parallel gateways causing additional plan threads to be created. Plan threads can then execute BPMN activities as part of the plan, which again records creation events when the execution starts and a disposal event when the execution finishes.

Event monitoring can also extend into the work item management. If a BPMN activity consists of a human task which causes the creation of a work item, it also causes creation events and corresponding disposal events to appear in the monitoring system. The final recorded event source in the system is the workflow participant activity for which (1) a creation event is issued once a work item is assigned to a workflow participant and (2) a disposal event is stored once the assignment becomes invalid. However, assignments can become invalid not just due to the user finishing the assignment but also if the work item becomes unassigned due to administrative action or because the workflow participant decided to return the work item to the system.

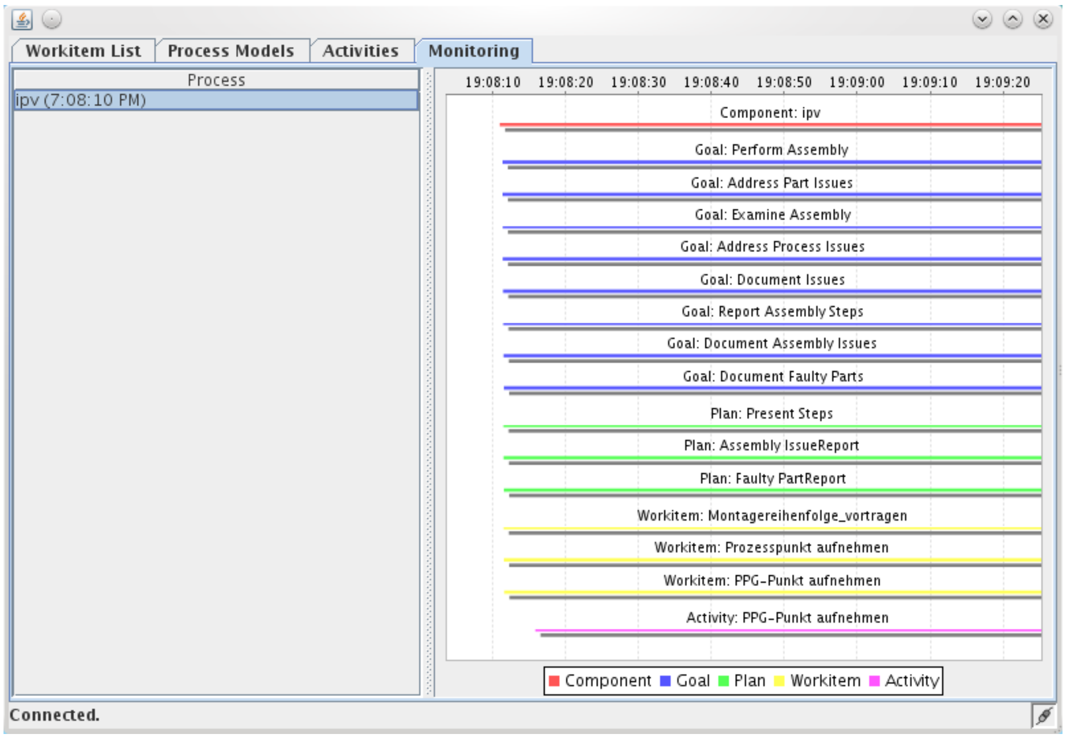

Workitem List
Process Models
Activities
Monitoring
Process
ipv (7:08:10 PM)
19:08:10 19:08:20 19:08:30 19:08:40 19:08:50 19:09:00 19:09:10 19:09:20
Component: ipv
Goal: Perform Assembly
Goal: Address Part Issues
Goal: Examine Assembly
Goal: Address Process Issues
Goal: Document Issues
Goal: Report Assembly Steps
Goal: Document Assembly Issues
Goal: Document Faulty Parts
Plan: Present Steps
Plan: Assembly IssueReport
Plan: Faulty PartReport
Workitem: Montagereihenfolge_vortragen
Workitem: Prozesspunkt aufnehmen
Workitem: PPG-Punkt aufnehmen
Activity: PPG-Punkt aufnehmen
🟥 Component 🟦 Goal 🟩 Plan 🟨 Workitem 🟪 Activity
Connected.

Figure 8.8: The monitoring support being used to inspect a workflow instance displayed as a Gantt chart (from [84])

Events recorded by the monitoring components can then be used to perform real time as well as post-execution analysis of workflow instances using a monitoring and analysis tool shown in Figure 8.8. Currently the tool displays the executing or executed workflow instance in a Gantt chart. The tool already sorts the available event sources based on their typical position in the hierarchy with the component or workflow instance itself being listed first.

However, the monitoring system which the analysis tool uses to retrieve the events associated with a workflow instance consists of the set of distributed monitoring components of Jadex WfMS which in turn uses the distributed storage system enhancement of the Jadex platform. This means that in some cases such as network partitions, there may be missing events. This is due to the fact that the storage system used by the monitoring system is based on an eventually consistent model (see section 7.6.2) which means that while a consistent state is eventually reached, there are transitory states in which some information may be missing.

However, the analysis tool is able to compensate to a certain extend for missing events by employing some heuristics. For example, if a disposal event of a plan instance is recorded, it contains the timestamp when plan execution was finished as well as the goal that was the cause of the plan execution in the first place. In such a case, if the monitoring component is unable to supply the corresponding creation event for the plan instance, the plan instance can still be displayed with some minor inaccuracy in the chart.

This is because the timestamp of the plan instance creation has a lower bound, namely the creation of the event cause which in this case is the goal instance. Therefore, the monitoring and analysis tool can as a preliminary estimate set the beginning of the plan instance to the same timestamp as the creation event of the goal instance. When, at a later point, the creation event for the plan instance becomes available, the tool will automatically correct the starting point of the plan instance to match this creation event.

Since even a moderately complex workflow instance has a large number of event sources, the tool allows the user to perform "drill-downs": By selecting a particular event source, the view changes and the tool narrows the event sources down to those that can be transitively linked to the selected source.

For example, the user can select a goal instance as the drill-down target. The tool will then show the goal instance as the first item on the Gantt chart and only display the subgoal and plan instances as well as plan threads, tasks, work items and user activities whose cause can be linked to the selected goal instances. This allows a user performing a workflow instance analysis to not only deduce the associated business goals of the more low-level aspects up to and including the performance of activities by workflow participants but also the time intervals that were necessary for low-level aspects like tasks but also higher-level ones like goals.

In addition to monitoring the execution of real world workflow instances, it is sometimes also useful to analyse a workflow model before it is executed under real circumstances for the first time. As a result, the next part of this work will present an approach for accomplishing such a pre-execution analysis for GPMN workflows which allows a workflow designer to validate a propsective workflow model before it is employed in practice.

# Chapter 9

# Workflow Model Analysis and Validation using Simulation

One of the key differences of long-running autonomous business processes and workflows compared to production and administrative workflows is that their models are often used for a single execution cycle after which the process or workflow model is changed to apply the lessons learned from experience and integrate new approaches (see section 1.2). Compared to production workflows, this means that the traditional post-execution analysis phase has a reduced importance since the gained experiences from an execution cycle generally relate to the subject matter of the workflow rather than the workflow itself.

However, the long-running nature of the processes introduces a different demand on the analysis phase of the BPM lifecycle: Due to the long execution time, it is an explicit aim to include as many contingencies for predictable changes in the business environment in the workflow model before the workflow instance is enacted. Additionally, the long execution time increases the chance of design and implementation errors to be exposed during execution. Finally, the lack of explicit execution paths in goal-oriented workflow models increases the need for workflow designers to receive feedback on the actual execution behavior of their workflows. These issues result in a strong need for an approach that provides runtime information about goal-oriented workflows before the workflow is actually deployed and enacted in the intended real world situation.

This means that the analysis phase has to be performed before a real world deployment of the workflow, potentially even before the design or implementation phase has finished. As a result, such an approach has been developed for GPMN-based workflows to assist workflow designers in developing the workflow models (see [86]), which will be presented here.

While goal-oriented workflow emphasize the need, the demand for a pre-deployment analysis is not unique to goal-oriented workflows in that workflow designers generally have a need to produce workflow models in which the number of flaws are minimized. Generally, two main approaches are available to ensure a level of correctness of implemented workflow model, both of which have certain advantages and disadvantages. The first approach is a formal verification of the workflow model. This approach takes the implemented workflow model and performs a static analysis to prove that the model either complies or fails to comply with certain constraints. A number of formal verification approaches have been applied to workflow models with certain success including graph reduction [139], model verification [48] and propositional logic [13].

Formal verification approaches have the advantage of providing a degree of provable correctness: If implemented and applied correctly, the facts that are derived by the verification process are guaranteed to be correct under the given circumstances, providing a high degree of confidence. The disadvantage of such approaches are the limitations of what can actually be proven about the workflow models. For example, a number of techniques are available to prove model correctness such as the termination of parallel branches in the model with corresponding join elements. However, in case of non-trivial semantics of the workflow such as tasks that are written in a sufficiently powerful (Turing-complete) programming language, an automatic proof is not possible.

This limitation of formal verification approaches is compounded if the workflow is subjected to a model conversion processes before execution. In this case, not only the correctness of the original workflow model but also the conversion process itself has to be proven correct for the strong guarantees to be applicable to the results. Finally, the executing workflow engine also requires a proof of correctness to ensure that the behavior of the enacted workflow matches the semantic definitions of the model.

Due to these limitations, an alternative approach which provides weaker guarantees but a broader applicable scope can be chosen to increase confidence in the behavior of enacted workflows. This approach is based on enacting the workflow in a simulated environment and systematically performing carefully chosen tests to assess the behavior of the workflow instance. This approach is therefore based on the actual behavior of the workflow during runtime instead of basing it on the static analysis of the workflow model.

After validating the correct behavior of the workflow with regard to the performed tests, a reasonable empirical assumption can be made that the workflow will behave in the expected manner if subjected to the same circumstances. However, unlike the formal verification approach, this approach of validating workflows does not provide absolute certainty of correctness. Nevertheless, it is capable of providing a reasonable level of assurances regarding workflow behavior and, unlike formal verification, can validate workflow behavior with non-trivial semantics.

A number of implementations of this approach exist in the market, examples including tools like L-SIM [159], iGrafx Process [111], Casewise Corporate Modeler Suite [5] and the ARIS Toolset [140]. Such simulation-based tools usually target a specific workflow language such as BPMN in case of L-SIM or are part of a larger workflow or business process software suite and are tailored towards the workflow models that are part of that particular suite. In addition, many tools tend be made to assist in optimizing business processes instead of validating correctness and therefore primarily target performance figures as is the case in the ARIS Toolset which offers a number of features assisting in the measurement of operating performance of workflows.

The approach presented here instead primarily attempts to address the validation requirements for goal-oriented workflow models through the use of test cases provided by the workflow designer in order to assess the expected behavior of the designed workflow model.

Workflow behavior depends on language features that define different execution paths or branches. In BPMN, these features primarily consist of the gateway model elements, which split the control flow either exclusively or in a concurrent manner, thus redirecting the excution along different control flow edges depending on runtime parameters.

The same mechanism is more indirect in GPMN workflows: The workflow context defines the business situation which then causes different goals to activate and plans to be executed. This subtlety in the execution path is the reason for process designers experiencing issue when attempting to understand the execution flow of a goal-oriented workflow. In both cases the primary source which influences these points are the result of actions by a workflow participant.

For example, when used to design workflow models at Daimler AG, the non-obvious behavior of goal-oriented workflows was noticed by workflow designers. In order to ensure a predictable behavior during execution, many of them resorted to the use of manual tests: The workflow designer would enact the current version of the workflow model in a test environment and then manually perform the actions of the workflow participants using the workflow client while evaluating the resulting workflow behavior using the monitoring system. The performed tests were usually deliberately targeting common situations and corner cases that could occur during a real world enactment.

Since the manual tests required a considerable amount of time and had the risk that a workflow designer would, at any point in time, omit a test due to negligence, the goal of the simulation-based test system was to assist workflow designers by allowing them to create a set of important standardized tests for each workflow model which could be performed and validated automatically. This also means that the process execution can be executed in a mode that allows it to finish "as fast as possible" or, in other words, the system is designed to simulate the environment within an accelerated time frame and allow the simulation to finish as quickly as the hardware allows. This approach partially resembles the use of test system in software development, where test cases are defined and executed regularly and automatically to avoid the introduction of errors into software code.

However, the input provided by workflow participants in non-trivial workflow models can be quite broad, resulting in an extremely large input space for possible values to simulate. This means that the simulation complexity quickly increases beyond what can be simulated within a reasonable time frame. As a result, the simulation model and coverage will necessarily be limited. This means that unlike a full coverage test or a formal verification, a level of uncertainty will remain as the system relies on the workflow designer to provide adequate tests for the workflow model.

## 9.1 Validation Approach and Client-side Model

While Jadex provides certain facilities for simulation including environments (see [85]), the facilities to control time or events are most useful in context of simulating workflow environments. This means a simulation environment needs to be implemented to support validation of workflows with a simulation-based approach. Since the goal of any simulation model is to closely resemble a real operating situation, the existing workflow management system can act as the starting point of the simulation environment with certain adaptions in order to simulate the business environment.

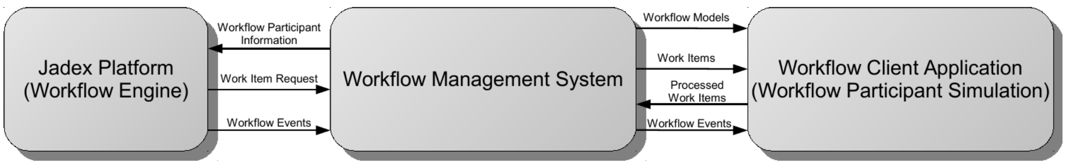


Figure 9.1: Structure of the interaction between subsystems in the simulation-based testing approach (from [86])

However, in order to complete a fully automated environment, a simulation model for the workflow participants needs to be implemented. For this, as shown in Figure 9.1, the standard workflow client application used by workflow participants can be replaced by an implementation which simulates the interactions of the participants with the workflow management system since the participants are unavailable during simulation. This simulated workflow client application has to be supplied with a behavior model simulating workflow participants. After automated enactment, the simulating workflow client application will then provide the information to the workflow management system based on this behavior model.

In particular, the application will request the work items made available by the workflow management system, process them using the behavior model, then submit the results back to the workflow management system in order to advance the workflow instance. Meanwhile, the application will also use the monitoring system to gather information about the workflow instance for analysis by the workflow designer.

In theory, the behavior model for workflow participants can be as complex as necessary in order to provide a realistic simulation of a real world workflow enactment and processing. However, in order to achieve the goal of a behavior validation of workflow models, the workflow participant model has been somewhat simplified. This simplification also allows the system to assist the workflow designer to provide an adequate behavior model for the workflow participants.

The simulation model of the workflow participants is based on the client-side model which uses the meta-model shown in Figure 9.2. This meta-model is based

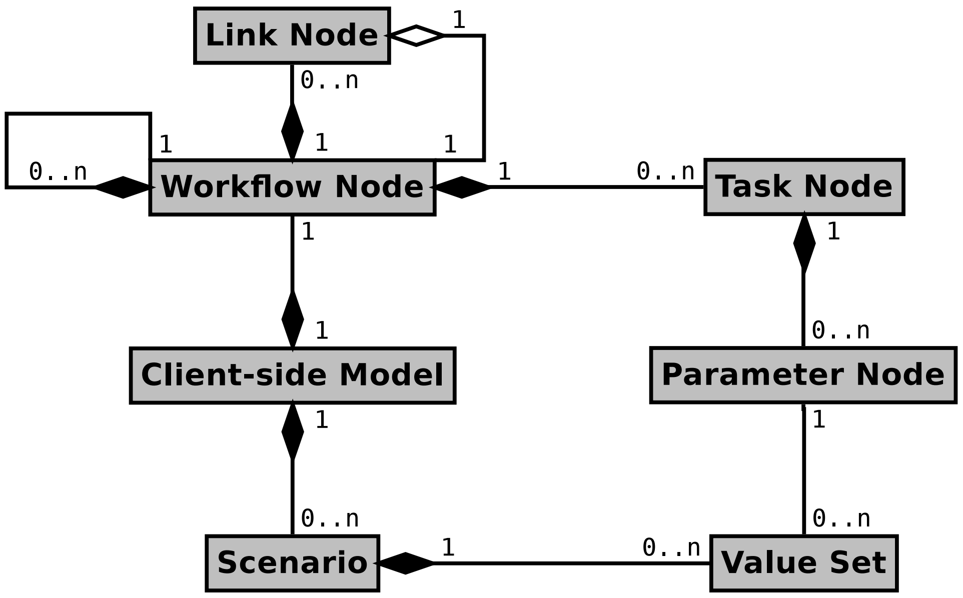


Figure 9.2: Meta-model of the client workflow tree model used by the simulation workflow client (from [86])

on the fact that the interaction of the workflow participants with the workflow management system is limited by the workflow model itself. The meta-model therefore allows the simulation workflow client to derive a simulation model for the workflow participants based on the possible inputs of the workflow model, creating the initial structure for the *client-side model.*

This initial model consists of three primary elements, which are used to generate the tree-based client model from the selected workflow model. The first element used in this tree structure are workflow nodes. Workflow nodes represent workflows or sub-workflows as they appear in the initial model. The selected workflow is represented by such a workflow node but any sub-workflows contained within the model are also represented in the model with such a node. Workflow nodes of workflows that contain sub-workflows represent this relationship by maintaining references to the workflow nodes of the sub-workflows.

In case of GPMN workflows, referenced workflow nodes can also include BPMN plans since the client simulation is primarily concerned with the potential enactment of a workflow model rather than the circumstances of the enactment, which from the perspective of both the workflow participants and the simulated workflow client is irrelevant.

In theory, workflow models allow the workflow designer to define cycles of sub-workflows. For example, if a sub-workflow enacts a workflow instance based on the workflow model of its parent, such a sub-workflow cycle is created. These cycles result in the sub-workflow model becoming a graph rather than a tree. However, such cycle structures are relatively rare in most real world workflows and therefore workflow designers expect a tree model to be used to represent the structure.

As a result, a tree structure is the form desired by the workflow designer but it may, in rare cases, not match the actual structure of the workflow model. Therefore, as a compromise solution, the client-side model still follows the tree structure but uses *link nodes* to represent the rare case in which a cycle is present. If such a cycle is detected during the generation of the client-side model from the workflow model, the second and later occurrences of the same sub-workflow node is replaced with a link node containing a reference to the original workflow node. The link node itself contains no further reference to other workflow nodes and therefore breaks the cycle in the graph. If the workflow designer selects a link node, the referenced workflow node is selected.

This approach presents the workflow designer with the expected tree model while also allowing for the rare cases in which the workflow graph contains cycles.

GPMN and BPMN workflow nodes are presented to the user as visually distinct but are not distinguished as such within the simulation model. However, BPMN workflows can include references to *task nodes*. Task nodes are representations of tasks to be performed by workflow participants that result in the generation of a work item within the workflow management system and therefore need to be dealt with by the simulated workflow client.

The outputs and inputs of such tasks are defined in the workflow model as typed parameter, including whether the parameter is considered to be information delivered to the workflow participant performing the task (*input parameter*) or if it is considered to be information that the participant needs to provide as a result of their work (*output parameter*). In some cases, the parameter can be specified to be both an input and an output parameter, which is done when the workflow participant is expected to modify the information.

Parameters that are specified as input parameters in the tasks of a BPMN workflow are represented in the client model as *parameter nodes*. These parameter nodes are referenced by the workflow node representing their workflow. The parameter nodes are the meta-representation of the information a workflow participant needs to provide during the execution of the workflow. This means that in order to simulate the behavior of the workflow participants, the simulation workflow client has to supply the information defined by the parameter nodes.

This behavior is defined by *scenarios* and *value sets* which are supplied by the workflow designer. The next section will introduct both concepts and show how they can be used to complete the simulation model used to validate goal-oriented workflow models.

## 9.2 Scenarios

After the initial client-side model is generated from the workflow model, it is presented as a tree to the workflow designer. The parameter space of possible inputs for a single parameters tends to be very large, such as in case of e.g. 32-bit integer values with billions of possible states, or even not properly bounded as in the case of string values. Compounding this issue is the fact that the complexity of an exhaustive test sweep of the full parameter space of all parameters increases exponentially, preclude such a test even in relatively trivial cases.

As a result, the test coverage necessarily has to be partial, which means that there is a chance that an error may not be found because it falls within the part of the test range that has not been tried. Therefore the validation method presented here relies on the process knowledge of the workflow designer in order to define a set of tests that cover critical issues and corner cases regarding the workflow.

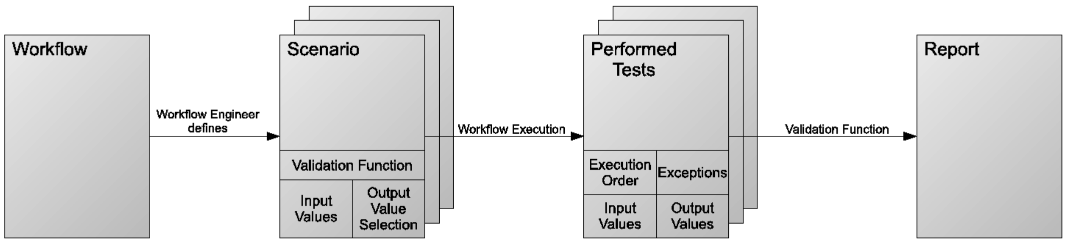


Figure 9.3: Testing procedure used to validate workflow models (from [86])

The full test procedure used to validate workflow models using simulation in this approach is shown in Figure 9.3. As shown in section 9.1, the simulation client application generates a client-side model as a base for the designer to add the behavior regarding the parameters that must be supplied during workflow execution. Since the complexity of a full parameter sweep exceeds reasonable testing times, the workflow designer is tasked with supplying a reasonable set of value that need to be tested.

When creating a workflow model, the workflow designer has a general idea of the situations in which the workflow is going to be used in a business environment. In addition, the workflow designer may be aware of some corner cases that depend on the workflow model. The presented validation system therefore uses these notions and allows the workflow designer to create *scenarios*, which represent well-defined test cases for certain situations against which the workflow model needs to be validated.

Scenarios consists of three parts: First, the input values used in the scenario to simulate the inputs of the workflow participants. This can consist either of single values, discrete sets of values or, in case of integer values, a range of values as specified by the workflow designer. As a result, each scenario contains a number of of tests based on every possible combination of parameter values available.

The second part of scenarios are the output values selection. Workflow instance may generate a large number of output values depending on the tasks executed and the supplied input values. However, not every output value is necessary to validate the correct behavior of the executed test. Therefore, the workflow designer can reduce the validation data by selecting only the output values necessary for validation.

The final part of a scenario is the *validation function.* This function is used to validate the correct behavior by comparing the results of a simulated workflow execution with the expected results as defined by the workflow designer as part of the function. Since a large variety of validations are possible which are heavily dependent on the workflow and the business environment, the validation function is supplied in the form of a Java class in order to allow the workflow designer as much definition range as possible. The goal of the validation function is to generate a report for the workflow designer whether the validation of the model has succeeded or, if it has failed, what condition led to the validation failure.

Once the scenarios are added to the base client-side model, the simulation model is considered complete. It consists of the workflow model, the workflow management system and the simulation client application with client-side model including the scenarios defined by the workflow designer.

This simulation model can then be used to validate whether the workflow model conforms with the specified validation function. The simulation client application will automatically enact a workflow instance for every parameter combination defined by the parameter sets of the scenarios. During execution, the selected output values are gathered as long with information from the monitoring system which includes order of subprocess and task execution and exceptions that may have occurred. This information about the performed test is then passed to the validation function of the scenario which generates a report for the workflow designer.

This report can then be used to correct errors found in the workflow model. Since the system can be invoked and executed automatically, it can be regularly applied during the modeling of the workflow. Provided the workflow designer defines a good set of scenarios that reflect both critical cases as well as the intended real world deployment situation, a reasonable assurance can be reached that the developed workflow model will perform as expected once deployed.

## 9.3 Example Use Case

As mentioned at the beginning of this chapter, a motivation for the development of a test system was the labor-intensive testing cycle that was used by workflow designers using GPMN workflows at Daimler AG. Therefore, the use of the simulation-based test system was demonstrated by applying it to a workflow developed by Daimler Group Research and using it to identify an error in the workflow model.

The chosen workflow is considered to be a change management process (see [30]). Change management processes are used to coordinate the organization and labor necessary to introduce a change to an existing product which is already in production. They ensure that adaption to aspects like changes in the physical geometry of the product or changes to the production line proceed smoothly and prevent early mistakes that can result in production slow downs.

The chosen workflow is very large and complex and the details often contain business secrets of Daimler AG. As a result, this section will focus on a smaller subprocess as shown in Figure 9.4, which concerns itself with the gathering of information about the planned changes to the product, locating key personnel and allocating necessary resources. The output produced by this part of the process is a detailed specification of the changes and the requirements to execute them, both of which will be used in a later part of the process.

Like most GPMN processes, this subprocess starts with a primary goal which

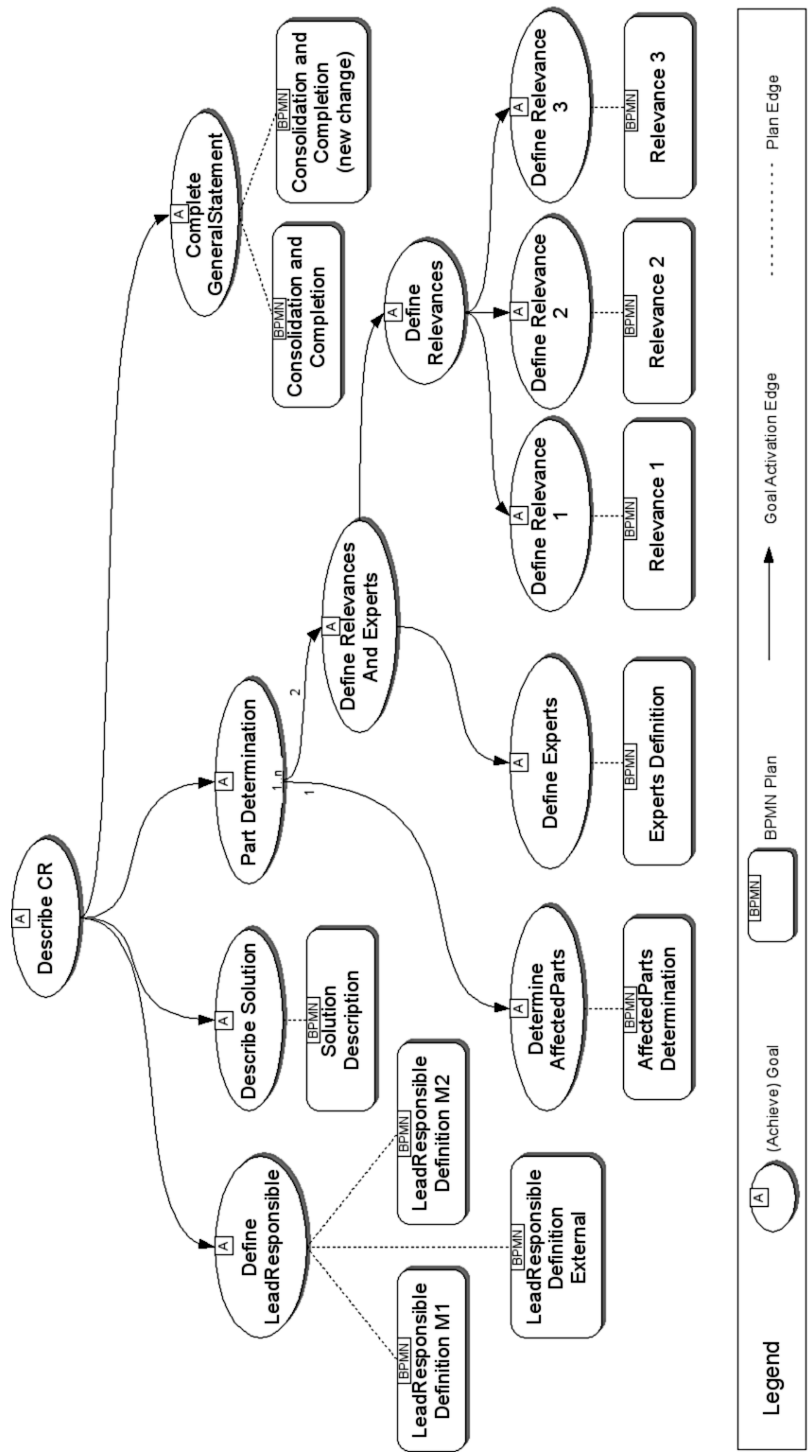


Figure 9.4: Goal hierarchy of the active change management workflow used to evaluate the simulation-based validation approach, created with an early version of the GPMN editor (from [86])

aims to generate a description of the planned change. The goal is then subdivided into multiple subgoals such as defining the lead person responsible for the change, generating a decription of the planned solution, determining the parts involved and finally the completion of the change statement.

These goals then get further refined until plans can be designed and attached. The subprocess uses multiple conditions on both plans and goals to ensure that prerequisites are met by the workflow context before a particular goal is pursued or plan executed.

When the workflow model was first implemented, it contained an error in one of the BPMN fragments which are used as plans in the workflow model that the workflow uses to reach the specified business goals.

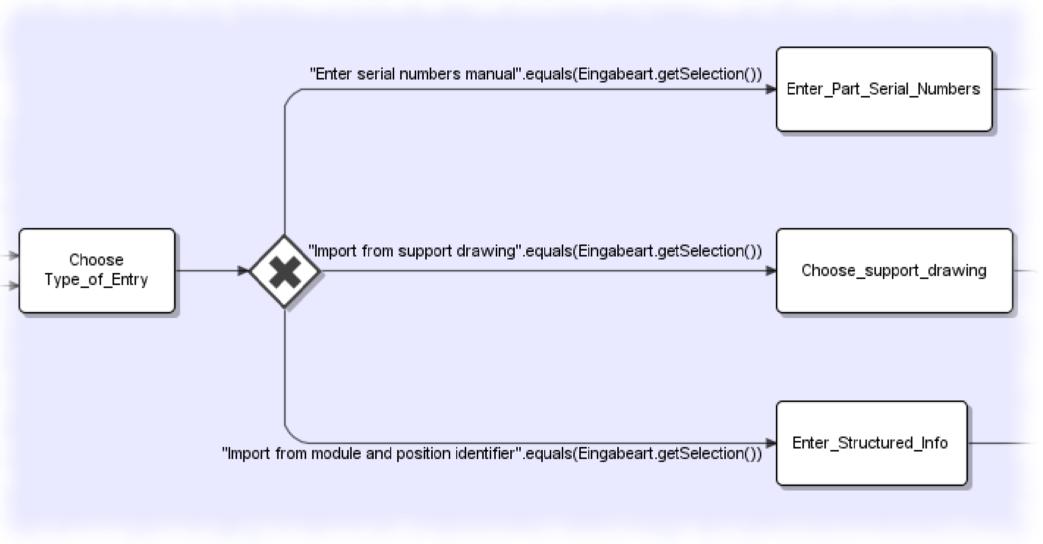

"Enter serial numbers manual".equals(Eingabeart.getSelection())
Enter_Part_Serial_Numbers
Choose Type_of_Entry
"Import from support drawing".equals(Eingabeart.getSelection())
Choose_support_drawing
"Import from module and position identifier".equals(Eingabeart.getSelection())
Enter_Structured_Info

Figure 9.5: Excerpt of the Affected Part Determination plan from the change management workflow (from [86])

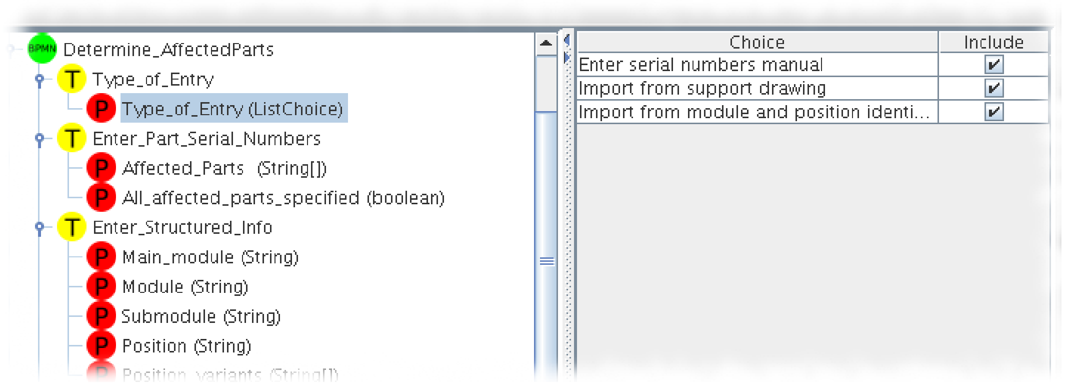
The plan containing the error is called ”Affected Part Determination” which is used to assess what physical parts of the product that are affected by the planned product change. As part of the plan, the lead developer in charge of the change process is required to enter these parts so they can be added to the resulting set. In order to do so, the lead developer has a choice between three different methods of specifying the affected part.

As the first choice, the lead developer enters a list of serial numbers of the affected parts. The second available option is to provide a drawing which is used for part identification. As a third way, the lead developer can enter a structured description of the affected parts.

These three options were implemented as part of the BPMN plan fragment as shown in the cutout of the BPMN fragment in Figure 9.5. The first task in this part of the plan generates a work item in order to let the lead developer chose the desired entry type. This work item contains a list of three things detailing the available entry types which are presented to the lead developer as three options. The lead developer can chose one of the options which causes the associated string value to be committed as the result of the work item.

Once the work item is returned, the chosen string is passed to the exclusive-or gateway in the BPMN fragment. Here, the string is compared to the strings available on the outgoing edges behind the gateway branch. The process then proceeds to execute the branch of the fragment that matches the string chosen by the lead developer, which then provides a new work item that supports the chosen type of entry.

However, in the first implemented version of the process, one of the strings on the branches did not match a corresponding string as generated by the type selection task. Since none of the sequence edges following the gateway is marked as a default edge, the workflow would terminate with an exception if the option containing the error was chosen by the developer.



```
BPMN Determine_AffectedParts
  T Type_of_Entry
    P Type_of_Entry (ListChoice)
  T Enter_Part_Serial_Numbers
    P Affected_Parts  (String[])
    P All_affected_parts_specified (boolean)
  T Enter_Structured_Info
    P Main_module (String)
    P Module (String)
    P Submodule (String)
    P Position (String)
    P Position_variants (String[])
```

| Choice | Include |
|---|---|
| Enter serial numbers manual | ✓ |
| Import from support drawing | ✓ |
| Import from module and position identi... | ✓ |

Figure 9.6: Simulation client showing the client-side model for the plan excerpt and demonstrating the parameter selection for simulation (from [86])

This error was found when the simulation-based testing system was used to validate the workflow model. In one of the scenarios created by the workflow designer, the designer would test each branch of the three-choice option. In order to limit the test complexity for this particular scenario, every other part of this workflow except the section shown in Figure 9.5 was supplied with a single value parameter set. However, for the task making the choice of entry, each of the three strings were added to the parameter set by selecting them in the simulation client displaying the client-side model (see Figure 9.6).

In addition, the test scenario was also used to validate the input methods by adding multiple mock entries for the parts that were added to the parameter sets of each of the data entry method tasks, which are used by the workflow as part identifications for the parts affected by the change request during the simulated workflow execution.

Based on this client-side model, the simulation workflow client computed that the scenario had a total complexity of 972 test runs before the specified parameter space was sweeped. The scenario was then executed on a workstation featuring an Intel i5 CPU clocked at 2.67GHz together with 4GiB of working memory. The simulation of all 972 simulated executions finished in less than a minute on this machine, a considerable improvement over the manual testing previously employed.

These tests also included the selection of the data entry option that contained the branching error. This led to an exception being raised during simulation and the simulated workflow was terminated. The monitoring system recorded these events which were then requested by the simulation workflow client and recorded in the simulation log along with information about the exact situation being simulated such as the set of parameters being used to simulate the workflow participants.

This simulation log which contained all performed tests and their associated information as shown in Figure 9.3 was then passed to the validation function which used the information in the log to generate a simulation result report for the workflow designer. In this report, the occurrence of the exception was highlighted, leading the workflow designer to examine the part in which the error had been generated. Using the parameter specifications used, the workflow designer was quickly able to identify the issue of mismatched strings in the workflow branch and could correct the error by ensuring a pair of matching strings.

## 9.4 Summary of the Simulation-based Testing Approach

As a result, the simulated testing approach was convincing in terms of improving upon the previous manual testing procedure for goal-oriented workflows, reducing the impact resulting from the issue of the non-obvious execution paths of GPMN-based goal-oriented workflows. While the client-side modeling is still somewhat limited, it could be improved by adding additional considerations such as timing and logistic concerns into account in order to produce a simulation model closer to a real world deployment of the workflow model. Nevertheless, the testing options available already grant the workflow designer a considerable amount of assurance that the workflow model will perform as intended. In addition, as a side effect of including the BPMN fragment plans in the simulation, the simulation-based testing system is also capable of simulating and validating plain BPMN workflow models using the same approach.

Furthermore, the simulation-based testing approach allows the workflow designer to perform a workflow model analysis ahead of time, representing the analysis phase of the BPM lifecycle as used for GPMN-based goal-oriented workflows. For this phase, two research goals were set in section 1.2 that are specific to the necessities of long-running autonomous business processes. First, the conflict between the the necessary workflow model agility which results in a complex workflow behavior and the requirements of a BPM analysis phase to associate the workflow behavior with the workflow model.

The testing system accomplishes this by combining the direct simulated execution of the workflow model along with the detailed monitoring available by the workflow management system. This allows the workflow designer to quickly associate post-enactment events with elements in the workflow model.

The second research goal for the analysis phase was to allow the workflow designer to connect the strategic and operational aspects of the business process during the analysis phase of workflow models. The combination of the goal-oriented workflow model and the client-side simulation model in combination with the monitoring system also allows the workflow designer not only to record all the action being performed by the workflow instances during execution but associate them with the business goals that are defined in the workflow model, which were modeled in the workflow design and implementation phase to match the real business goals of the organization for the business process being modeled (see chapter 4). This immediately provides a strong link between the strategic layer represented by the goals and operational layer represented by the actions and work items generated during the execution of the workflow instance. The simulation-based validation approach therefore represents a viable option for the analysis phase of the BPM lifecycle which takes the demands of long-running autonomous business processes into account.

Being representative of the analysis and therefore final phase of the BPM lifecycle, this validation approach also concludes the coverage of the current system used to support long-running autonomous business processes. The next chapter will summarize the various aspects of the system and will demonstrate how they contribute towards the originally set research goals.

Chapter 10

# Application Scenario and Conclusion

This chapter will summarize the presented agile business process management system and its parts, demonstrate an application scenario which was carried out in cooperation with Daimler AG and establish how each of the research goals for supporting long-running autonomous business processes as presented in chapter 1 is addressed by the system. Finally, the current limitations and potential improvements are presented.

An overview of the system is shown in Figure 10.1. The system currently supports two languages for modeling workflows: First it includes an implementation of the standardized task-based language BPMN. Based on this language, the system supports the newly-developed goal-oriented language GPMN, which uses BPMN workflow model fragments as plan elements.

For designing and implementing GPMN business processes and workflows, two editor tools are available, one for modeling BPMN plan fragments and another for modeling goal-oriented GPMN models that use those fragments. As an option, the BPMN editor tool can also be used to develop standalone BPMN workflows.

Both editors are capable of generating executable workflow models provided the workflow designer supplies the necessary execution detail. In the case of the BPMN editor, the model consists of the BPMN 2.0 model format as defined in the specification. However, it includes some extensions to support features that are specific to the Jadex Active Components Platform which is used to execute the workflow instances resulting from the model. These extensions include information such as platform-specific configurations and services that are required or provided as Jadex Active Component services.

The BPMN engine is available as part of the Jadex Active Components Platform. Its parser can directly read the BPMN 2.0 models generated by the BPMN editor in order to construct an executable BPMN engine model. Using this engine model, BPMN instances can be generated which are executed using an internal BPMN interpreter.

GPMN workflow models however are converted to a BDI agent modeler before

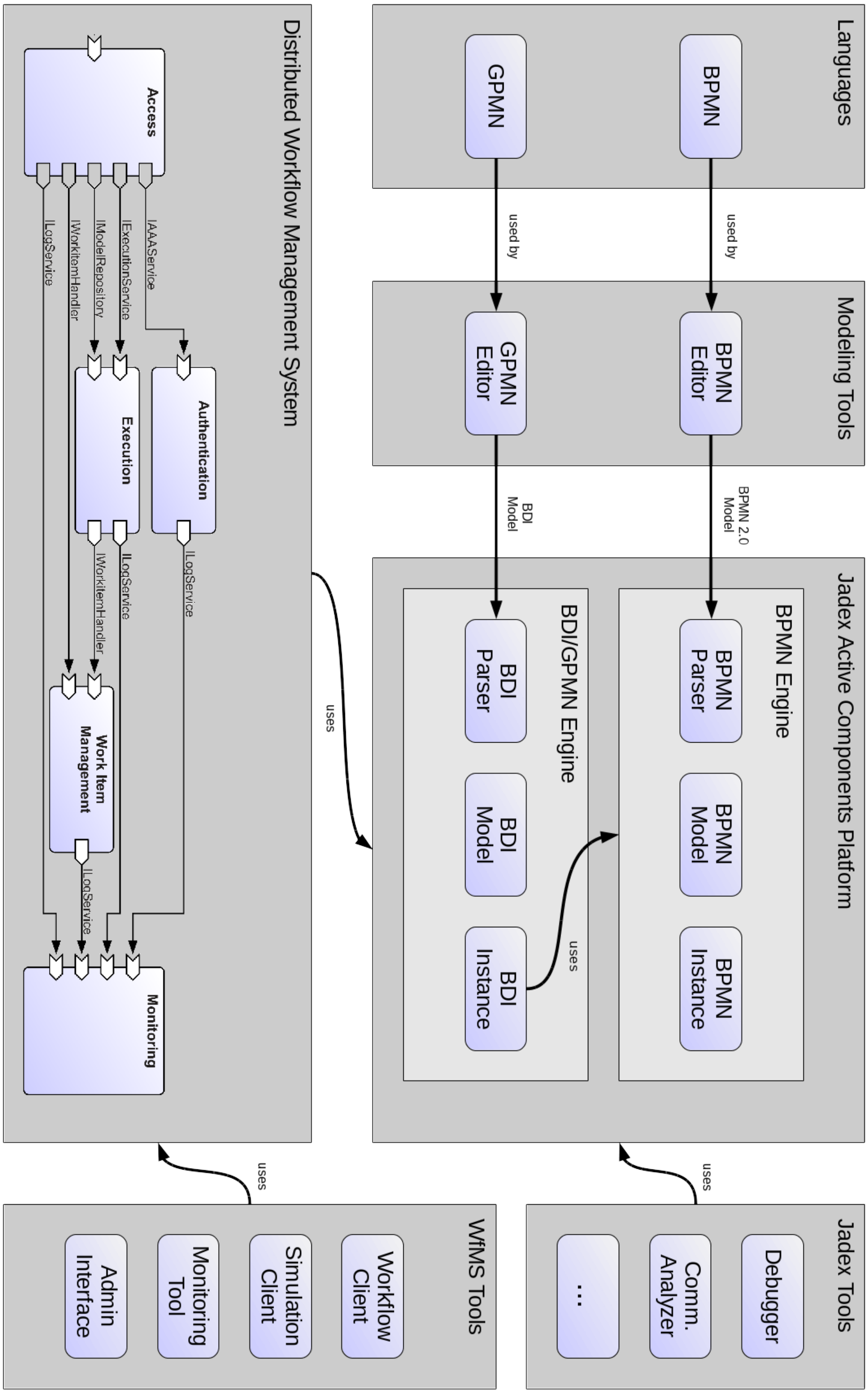


Figure 10.1: Overview of the system used to support long-running autonomous workflows

they are processed and executed by the BDI engine in the platform. The original GPMN modeler can be saved by the editor in the GPMN intermediate format and converted to a BDI model at will. A conversion from BDI models into GPMN intermediate models is unsupported since BDI includes elements that are not currently available in the GPMN language.

Since both BPMN and GPMN workflows are executed by the platform as active components, in both the case of BPMN directly and in the case of GPMN as BDI agents, the complete tools support available for Jadex can be applied on those workflows. This includes tools such as the debugger for stepping through the workflow as well as the communication analyzer for inspecting the communication behavior of workflow instances.

The Jadex platform was also used to implement the distributed workflow management system. This system provides the typical function of a workflow management system including user management, access control and work item distribution. However, the system is divided into five separate active components which can be individually replicated and distributed among a network of nodes running the active component platform.

The five components are the access component, which acts as a gateway for external clients connecting to the system, an authentication component which provides user and rights management which is used by the access component to regulate access. An execution component handles the repository of workflow models and assists in the enactment of workflow instances. The work item management receives and assigns work items that are generated for human participants of the workflow and finally the monitoring component which receives and agglomerates events that occur during workflow execution.

The replication of these five components increases reliability by allowing for fallbacks in case of sudden outages. The destributed nature of the system allows subunits of departments to administer aspects of the system such as user management while still participating in an overarching workflow management.

Finally, tools are available to access and interact with the system for a variety of tasks. A standard workflow client is available for typical interactions of workflow participants such as requesting and completing work items. The client also includes an administration interface and a monitoring tool which can be used for such tasks provided the user possess the correct privileges.

Validating and testing workflow models can be accomplished using the simulation client, which automatically enacts workflow instances and then simulates the interaction of workflow participants according to pre-defined test cases. This can be used by workflow designers to allow a reasonable level of assurance that the implemented workflow model will perform as anticipated given particular scenarios.

## 10.1 Application and Deployment Scenario

The full system was tested on two separate occasions in a real world scenario based on production preparation at Daimler AG. Production preparation is part of the set of processes used to move a newly developed car series from prototype to full-scale production ramp-up. This means that inital product development produced a prototype and the production process of the new series has to be developed and initiated.

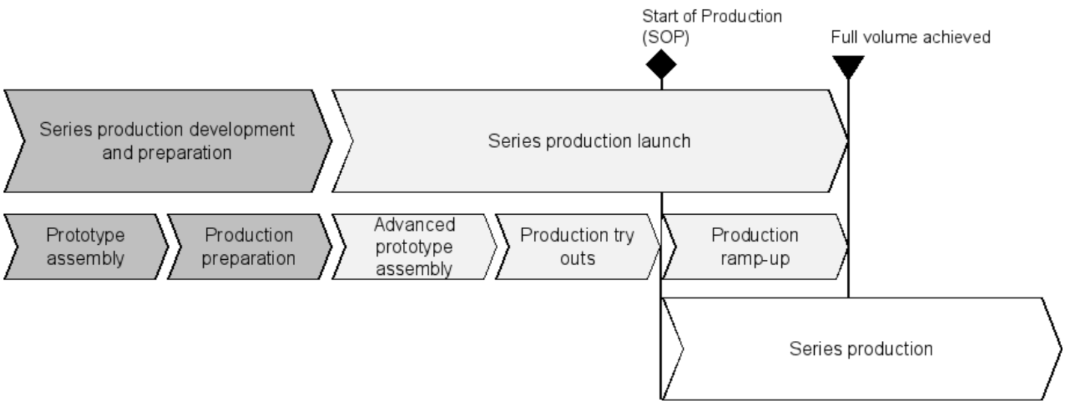


Figure 10.2: Process map and phases in preparation for a series launch at Daimler AG

Figure 10.2 shows the map of processes which are used to start the production of a new car series. It is separated into two major steps: The first step is the series production development and preparation in which the product as well as the production process are developed and validated. This is followed by the series production launch when the developed product and production process is slowly introduced until full series production can commence.

The series production development and preparation involves two major groups of processes, prototype assembly and production preparation. For testing the proposed system, the production preparation process was chosen as an example of a typical step in the preparation for a new series launch. Before the first production process is enacted, the new product has been developed with all targetted features. In addition, a first production process model is available.

### 10.1.1 Introduction to Production Preparation

The production preparation process is essentially a slowed-down test enactment and performance of the production process. This means the whole prototype is assembled using the current model of the proposed production process. While this test assembly is being performed by a production mechanic, domain experts from different departments who are concerned with production aspects are present in order to evaluate the production process itself as well as issues related to the production process (see Figure 10.3).

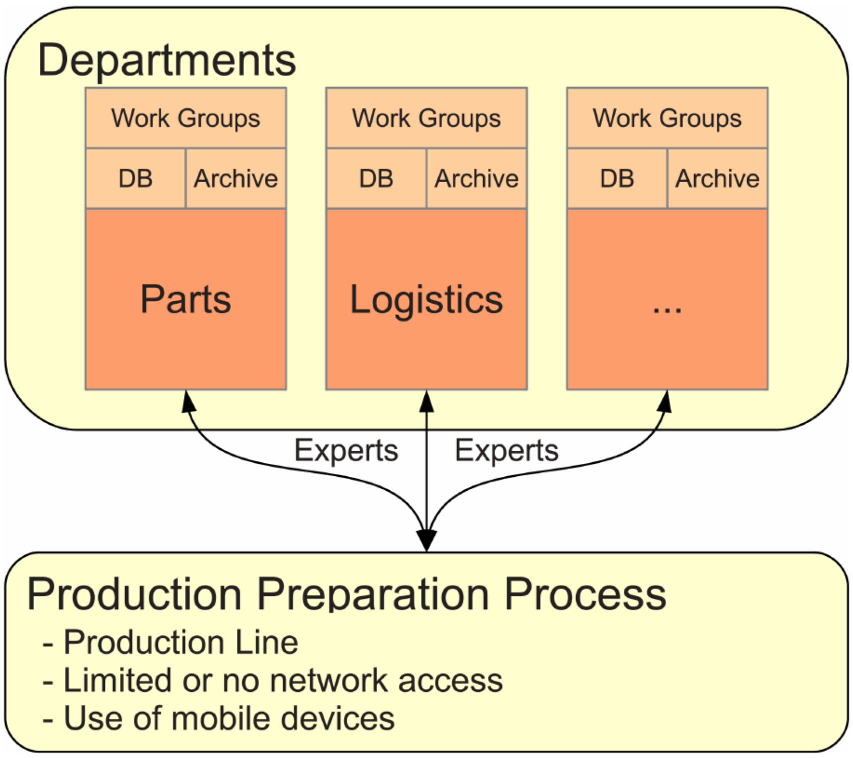


Figure 10.3: During production preparation, domain experts from various departments gather to evaluate production issues

The domain experts in the process are concerned with a number of aspects regarding the eventual production of the vehicle, which includes the following examples:

- Production process issues: This area involves the domain experts involved in the design of the production process. During production preparation, these experts evaluate whether the production process itself is correct and efficient. One important question involves the order of the process steps and whether they allow the assembly of the vehicle at all. For example, it is an error in the production process if the mechanic is instructed to attach the wheels before the axles are mounted. The production process experts are also interested in opportunities to improve the process. For example, this can sometimes be accomplished by reordering assembly steps: During assembly, the vehicle is often raised, lowered and rotated on the assembly line. If two parts of the vehicle need to be assembled while the vehicle is lowered, it is more efficient to attach both parts to the vehicle while the vehicle is in a lowered state instead of raising in between the two-part installation.

- Parts issues: This area involves the parts engineers who evaluate the parts that were developed during the prototype stage and that are now used to assemble the vehicle. Here, two main issues need to be evaluated: First, whether assembly is possible at all and not blocked or otherwise impeded and second whether the assembly is efficient. For example, parts can be attached to the vehicle faster if they are snapped-on using plastic hooks or glued in place rather than using bolts and screws. However, a bolted connection with the vehicle is usually more durable, requiring the parts engineers to balance construction efficiency vis-à-vis solid construction.

- Logistics issues: Logistics issues involve the question of how parts are made available to the assembly line. Before assembly, parts are stored on special shelves and carts which are wheeled to an assembly line station when and where they are needed. The logistical issues addressed during production preparation involve the shelves and carts required at particular stations and the exact position of parts on both carts and shelves to minimize the time required by mechanics to retrieve relevant parts.

- Ergonomic issues: A considerable amount of assembly steps are performed by a mechanic who is working on a particular station on the assembly line. However, the repetitive nature of the labor raises important health aspects that must be taken into consideration. While mechanics are often rotated through different stations to mitigate issues of repetitiveness and many stations involve multiple steps, the mechanic still has to perform the same set of steps for many hours. This means that the strain on the human body needs to be minimized, both in order to meet legal requirements as well as avoiding mechanics developing medical conditions resulting in an increased number of days absent from work. For example, if mechanics are repetitiously forced to bend over to attach a particular part because the vehicle is lowered at their station, ergonomic rules require the assembly of the particular part to be moved to a different station where this particular strain is not required.

The experts who deal with these issues originate from different departments within the business. Nevertheless, for purposes of production preparation, they come together at a simulated assembly line in order to observe and evaluate the assembly using the production process. This approach also allows for discussion between experts which allows them to resolve conflicts over specific aspects.

The production preparation process itself mainly concerns itself with documenting issues that need to be addressed. The issues themselves are recorded and addressed later within the individual departments. Each department has its own approach for resolving the issues found during production preparation assembly and employ their own IT applications and database systems to record and process the issues found.

Furthermore, due to the sensitive nature of new products and issues of industrial espionage, network access at the assembly line is limited. This means that during production preparation, identified issues are often recorded manually on white boards, hand-written notes or office productivity software such as spreadsheet applications, which are then later used to transfer them into each department's IT systems.

Once the experts have been assembled for production preparation, the vehicle is assembled according to the current version of the planned production process. In addition to the domain experts, an experienced production mechanic is present to perform the assembly steps. For each assembly step in the production process the following basic actions are performed:

1. The assembly step instructions are presented visually using a projector and read out loud to the group of expert and the mechanic. This includes detailed instructions or visualizations if they are part of the assembly step in the production process (missing but needed assembly instructions are considered to be a production process issue that needs to be addressed).

2. The domain experts report already known issues that have not yet been addressed between production preparation iterations to everyone present.

3. The mechanic performs the assembly according to the step instructions while the domain experts observe.

4. A production preparation report for the current assembly step is created based on the discussion and recorded issues of the domain experts.

During each step everyone participating in the production preparation process is allowed to interrupt and discuss issues they have noticed. This is not necessarily restricted to people concerned with the particular area such as parts design: A logistics expert can also bring a parts issue they have noticed to the attention of the group.

Issues that have been identified and confirmed by the domain experts are recorded manually by the same experts so they can be resolved internally at a later point. Most of the time during production preparation those involved are standing or moving around the prototype; however, if an issue needs to be recorded, the expert may have to move away from the group to appropriately notate the issue. In addition, parts issues are also often documented by taking a picture with a camera which documents both the part and the vehicle in such a way that it demonstrates the issue with that particular part.

### 10.1.2 Goal-oriented Implementation of Production Preparation in GPMN

While the production preparation process has been in use for many decades, research was initiated to improve the fairly ad-hoc approach of this particular process. Due to the autonomy of the domain experts, the divergent IT systems in use and the relatively free-wheeling nature employing the experience and discussion of the domain experts to mitigate any potential production issues in advance, a high work load and corresponding responsibility is placed on the organizer of the production process called the production preparation manager.

By using the workflow system described in this work, an attempt was made to improve the following areas of the process:

- Provide a structure for the production preparation process and reduce the production preparation manager's workload without impeding the free-wheeling and creative nature of the process.
- Allow some integration of the IT systems so that issues get transferred to each appropriate department and helping that department address the issues without requiring the use of paper or other means of record that are foreign to the process
- Employ mobile devices to allow domain experts to record issues while they remain standing next to the vehicle in a digital format. Furthermore, allow the use of the integrated camera to take pictures as required to document issues.

In order to reach these improvement goals, the plan for the test deployment for the system included the following development steps:

1. Design and implement a workflow model with high enough flexibility so it can be used as the workflow for performing a production preparation process.
2. Create a deployment plan for the workflow management system that takes the reduced connectivity at the assembly line as well as the use of mobile devices into account.
3. Implement a workflow client for mobile devices that can be used during the production preparation process.

Steps one and two were achieved by using the system described in this work while the special workflow client was implemented by an employee at Daimler Group Research.

The first step involves the design and implementation of the workflow model. Since the production preparation process requires a high degree of security due to the involvement of business secrets and also in order to reduce the complexity, the process model shown in Figure 10.4 is a simplified GPMN model of the model used for the test deployment, which, among other things, only addresses parts and production process issues but is sufficient to explain the basic functionality of the workflow.

The main goal and root of the goal hierarchy is simply called "Production Preparation" and represents the goal of achieving a successful production preparation after performing a necessary number of iterations of the preparation assembly. The center subgoal is named "PP Assembly" and concerns itself with finishing the production preparation assembly at the prototype assembly line the necessary number of iterations and will only conclude as successful if the previous assembly iteration was marked as the final iteration of production preparation. This goal is again divided into two subgoals: One of those two goals is the "Assembly Review" goal. This goal includes an achieve condition that can only be met when all assembly steps of the production process have been performed at the assembly line.

The goal is sequential and is connected with four subgoals which represent the four steps which are done by the production preparation group for each part of the production process: Report the current assembly step, report the known issues

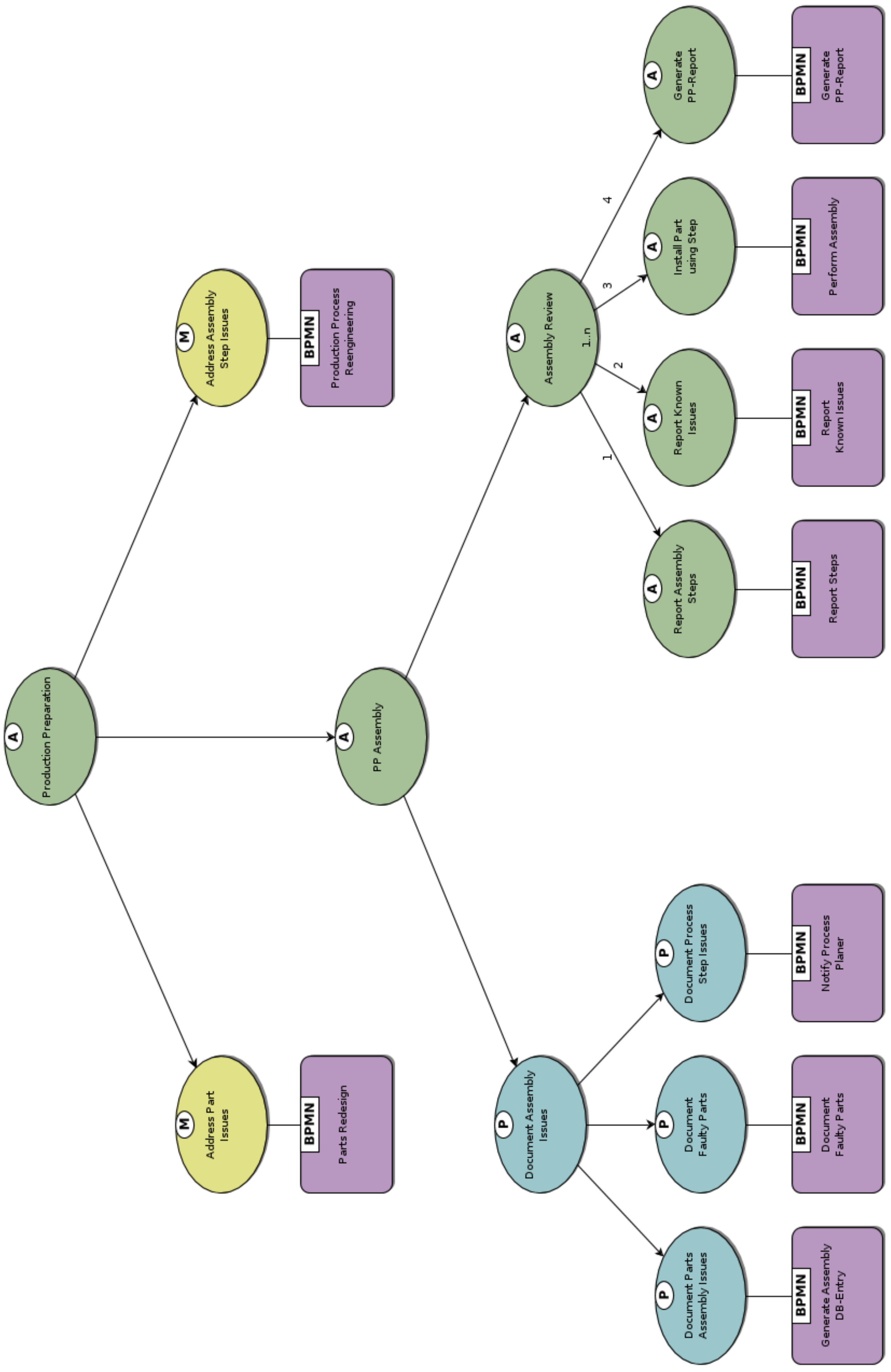


Figure 10.4: Jadex virtual networks and devices in the production preparation deployment scenario (from [82])

with the step, perform the assembly on the vehicle prototype and finally generate a production preparation report for this step. These goals each have a plan that include the generation of a work item and other necessary actions.

Since the "Assembly Review" goal is marked as sequential, each of the four steps are performed in the defined order. However, once the final step has been performed the goal does not terminate successfully due to its achieve condition and the retry flag being set. Instead, the goal continuously activates the four subgoals in the specified order until the all steps of the production process have been reviewed, which causes the achieve condition to evaluate true.

While the "Assembly Review" goal is active, the other subgoal of the "PP Assembly" goal is active as well. This goal is called "Document Assembly Issues" and is responsible for ensuring that issues found during the assembly are properly recorded. Since there is the unlikely but possible scenario that no issue is found during assembly, the goal is modeled as a perform goal instead of an achieve goal. It carries a drop condition that also becomes true once the last production process step has been performed.

The "Document Assembly Issues" goals has another three subgoals for each of the possible types of issues that can occur. The "Document Parts Assembly Issues" is used to record difficulties when attaching a part without assembly being completely impossible. For example, this includes excessive use of time or awkward work positions that could be fixed by designing the part differently.

The subgoal "Document Faulty Parts" is concerned with parts that are not only difficult to attach to the vehicle but are flawed in such a way that assembly becomes impossible. Finally, the "Document Process Step Issues" subgoal concerns itself with recording issues of the production process step itself such as step ordering.

Each of the three subgoals are perform goals with the retry flags set and each has a single plan attached which generates a work item for recording the particular type of issue. Since the subgoals are perform goals with a retry flag, they continuously stay active and generate another work item whenever the previous one was used to record an issue.

This approach allows the workflow participants to go through the assembly of the vehicle step-by-step in a similar way as it was done previously in the manual approach. The workflow deliberately does not moderate the free-wheeling interaction between the participants and only provides a rough framework for the assembly. While the assembly is being performed, work items for recording the various issues are available due to the perform goal subtree. This allows workflow participants to enter new issues at any time during the process and at different stages of a part assembly.

Once all assembly steps of the production process have been performed, the achieve condition of the "Assembly Review" subgoal evaluates true. This causes the whole subtree to terminate successfully. In addition, the drop condition of the "Document Assembly Issues" subgoal causes this subtree to terminate as well which causes the active work items for recording issues to be withdrawn. The termination then concludes an iteration of the production preparation with the "PP Assembly" achieve goal capable of triggering additional iterations as deemed necessary by the participants.

In addition to the "PP Assembly" subgoal branch, the workflow model also includes two further subgoals. These two subgoals are used to address the issues found during a production preparation assembly within the responsible departments. The first subgoal is called "Address Part Issues" which has a plan attached to redesigning parts that have been flagged during assembly. The "Address Assembly Step Issues" subgoal however has a plan for redesigning the planned production workflow to address process step issues.

Both goals are modeled as maintain goals with the maintain condition which aims to maintain a state of zero known issues in their respective categories. This means that whenever an issue is recorded during assembly, one of these two goals triggers their plans to address the issue. The plan will be repeatedly activated until all the issues have been cleared by the department.

The next section provides an overview of the workflow management system deployment in which the workflow model was used and demostrates how the distributed nature of the workflow management system supports the special circumstances of production preparation such as involvement of multiple deplartments as well as the isolated assembly line.

### 10.1.3 Workflow Management System Deployment

The production preparation process has a number of unusual circumstances related to how the process is conducted as was described in section 10.1.1. Regarding the deployment of a workflow management system on which the previously described workflow model can be enacted, three requirements need to be satisfied in order to address the circumstance of the process:

- The issues found during assembly are addressed by work groups within each specialized department which tends to use their own IT systems.
- The assembly takes place at a mock assembly line with the workflow participants standing and moving around the vehicle using mobile devices.
- Due to business security and secrecy, network connectivity is limited, for example, direct wireless access to the intranet is not possible.

In order to allow a smooth usage of the system during and in between assemblies, these requirements had to be taken into account during system design and deployment for the workflow management system.

The requirements were met by leveraging a combination of two distinct features of the workflow management system presented in this work. The first feature is the distributed nature of the system as described in section 8.1, the second feature is the separation of client and system networks in order to limit client access to the system as shown in section 8.3.

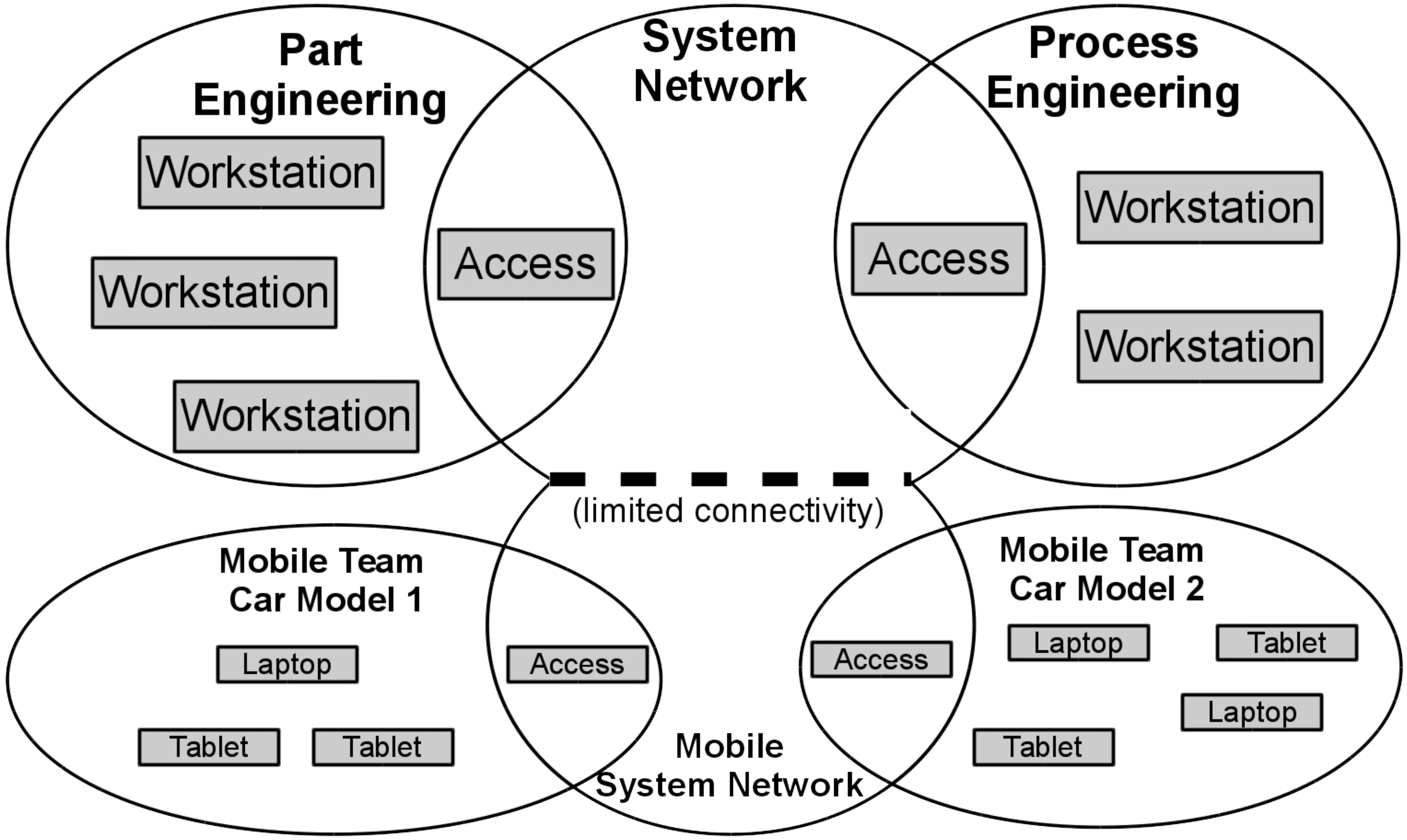


Figure 10.5: Structure of the deployed workflow management system using Jadex virtual networks and devices in the production preparation deployment scenario (based on [82])

The resulting system structure is show in Figure 10.5 in which the system components were omitted for brevity. This virtual system network, which contains the distributed and replicated components of the workflow management system, is separated into four different locations:

- A part of the system including workflow clients deployed in the part engineering department
- The process engineering department includes another part with workflow clients
- The main workflow management system components providing basic functionality in the IT department
- A mobile deployment used during assembly

The parts engineering department and the process engineering department each maintain a subset of the total system network. Their part of the system network primarily aims to allow the departments enhanced autonomy. For example, each of the departments includes a node with an authentication component in order to enable them to independently manage user accounts for their own employees. They also include local work item management which allows the integration of local legacy systems and a local access node for workflow clients.

The components in the departments' system networks is integrated with the main system network that has been installed permanently for the company as a whole. Here, most of the remaining functionality, such as monitoring, which is not provided by the departments is provided.

The final location for system network components is the mobile system network used at the assembly line. This part of the system network is temporary and is set up using mobile nodes whenever a production preparation assembly is in progress. Its primary job is to provide a local workflow management infrastructure for the mobile clients used during assembly and deal with the limited connectivity available by employing the distribution and resilience features of the workflow management system.

Workflow particpants can interact with the system from the client networks. Client networks are available for each department, allowing the workflow participants to access the workflow management system and request work items using their office workstations through the local department's access component node. This functionality is used during issue review, when the issues that have been identified during assembly are resolved.

The second kind of client network is the mobile client network which is the counterpart to the mobile system network. This client network is for the mobile devices which are employed during production preparation assembly. It is also possible to support multiple mobile client networks. This can again be useful for participants coming with their department-configured mobile devices. In addition, it allows mul-

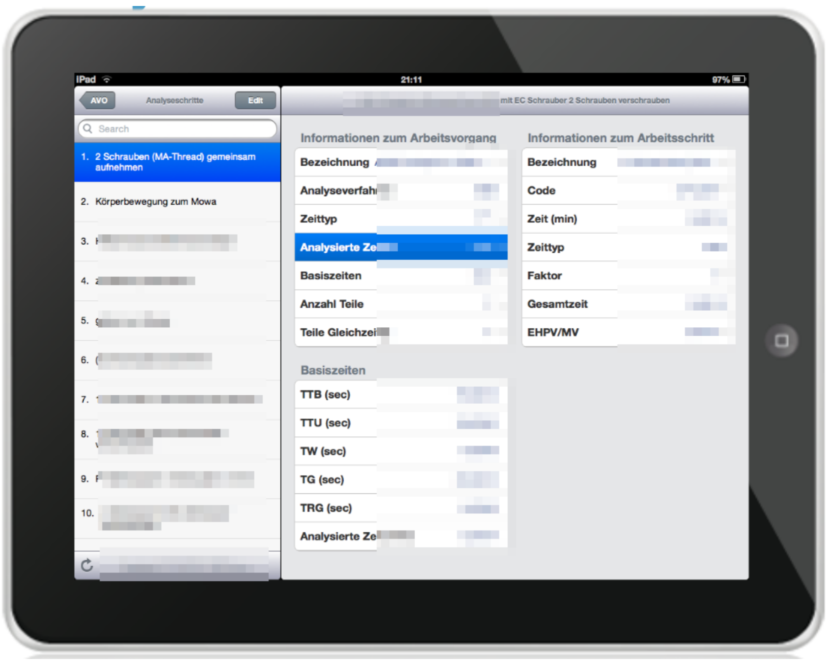


Figure 10.6: Tablet workflow client used during production preparation assembly, information partially obscured for business secrecy

tiple production preparation assemblies (e.g. for different car models) to share a mobile system network.

Standard laptops can be used to access the workflow management system. In addition, researchers at Daimler also developed a special workflow client for mobile tablets which is shown in Figure 10.6. Tablets are particularly useful during the production preparation assembly since they can be carried on the person and taken around and into the vehicle. Furthermore, the built-in camera can be used to document parts issues when they are found with any photographs being automatically included in the parts issue report. The picture can then be included in the parts engineering database without the participant from parts engineering having to manually upload it.

The system using the described deployment scenario and including the custom tablet-based workflow client was evaluated on two separate occasions as part of an ongoing production preparation for a new S-Class product line (see [82]). During the evaluation, the system performed as expected including handling the limited connectivity at the prototype assembly line.

While company procedures, company scheduling constraints and limited sample size precluded a formal survey, a request for informal feedback by the paricipants was issued. The feedback received was very positive regarding the system. In particular it was noted that the system was non-invasive and did not interrupt the "usual" hands-on and dynamic approach taken by the participants. It was also noted that the ability to take pictures which are automatically included as well as the integration with different departmental IT systems was time-saving. As a result, the system was made available to Daimler AG's IT department for further development.

## 10.2 Conclusion and Future Work

Chapter 1 of this work introduced long-running autonomous business processes and noted considerable differences when compared to business processes such as traditional production processes which have been the primary target of business process management. Chapter 2 showed how these types of processes are particularly affected by typical business process management issues such as organizational structures.

In order to tackle the particular challenges of long-running autonomous business processes, this work defined six research goals that needed to be addressed in order to improve the viability of including such processes in business process and workflow management systems:

- Increased *workflow model agility* is necessary to allow more autonomy for the participants and enable the workflow to adapt to changing business situations during the long execution time of the process.
- Due to the high degree of autonomy for participants and the workflow itself, *strategic-operational cohesion* is required to ensure that all performed actions align well with the business goals for the process.
- A *balance of global control and local autonomy* ensures that the workflow participants gain the autonomy necessary for this type of processes while ensuring that an appropriate amount of global control and coordination is maintained.
- Long execution times require enhancements for *system robustness* to ensure successful execution of the workflow.
- Organizations do not always have ideal process-oriented structures and the existing structures may change over the course of a workflow execution, requiring the system to support *organizational agility* allowing it to change with the organization's structure.
- *Workflow instance agility* would allow a workflow instance to be changed during execution, allowing adaptation to unpredictable business changes. This research goal was excluded from the scope of this work.

In order to address these research goals, each part of the BPM lifecycle consisting of the analysis, design, implementation, execution and monitoring phases had to be considered. For each of these phases, different solutions were proposed and implemented that addressed the research goals based on the requirements of the particular phase the solution addressed.

As a result, four main contributions have been developed as shown in Figure 10.7. Since long-running autonomous business processes tend to be only used unmodified for a single enactment, they require additional assurance that the model used is valid for the intended purpose. For modeling more flexible workflows that support increased local autonomy while maintaining strategic-operational cohesion,

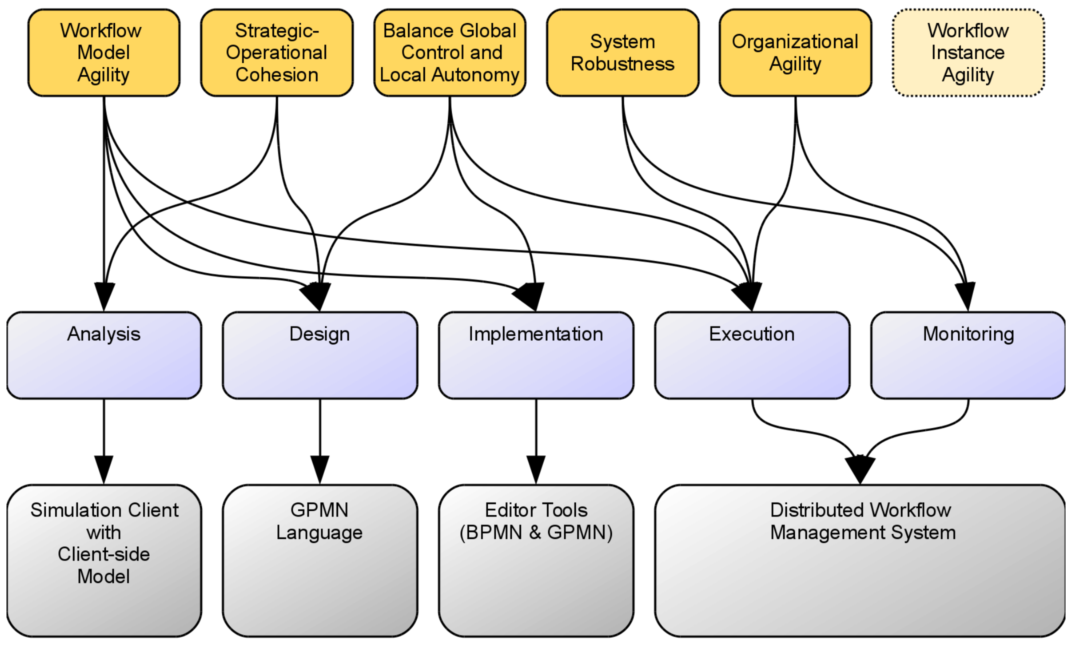

Workflow Model Agility
Strategic-Operational Cohesion
Balance Global Control and Local Autonomy
System Robustness
Organizational Agility
Workflow Instance Agility
Analysis
Design
Implementation
Execution
Monitoring
Simulation Client with Client-side Model
GPMN Language
Editor Tools (BPMN & GPMN)
Distributed Workflow Management System

Figure 10.7: The different phases of the BPM lifecycle are affected by a different set of the research goals for providing better support of long-running autonomous processes. Solutions have been developed to address the research goals in all of the affected phases.

a new goal-oriented workflow model language called Goal-oriented Business Modeling Notation (GPMN) has been developed (see chapter4).

For modeling as well as implementation, editor tools for GPMN as well as BPMN plans were developed which maintain the goal-oriented and flexible structure when generating executable workflow models (see chapter 6) based on BDI (Belief, Desire, Intention) software agents. For execution and monitoring, which is affected by most of the research goals, a distributed workflow management system was developed (see chapters 7 and 8). This system is capable of executing the goal-oriented workflow models generated by the editor tools and utilizes a BDI-agent reasoning cycle to provide flexible execution semantics. The distributed nature of the system also allows for a high degree of local autonomy for the participants as well as providing organizational agility by allowing organizational units to maintain different parts of the system independently as well as introducing redundancy, allowing parts of the system to be shut down and restarted if the organizational structure changes. The redundancy also provides for additional system robustness in case of unanticipated node failures. Finally, a simulation client has been developed which allows the workflow designer to simulate the execution of the workflow before it is deployed in the real world (see chapter 9).

As a result, the approach presented in this work is capable of maintaining the fixed research goals through the full BPM lifecycle and therefore enhance the support for long-running autonomous business processes, which were not adequately addressed previously by traditional business process management tools.

Nevertheless, the system currently includes two major limitations which can be the starting point for further enhancements: First, system robustness could be further enhanced by persisting the process state. This could be accomplished through a checkpointing approach which uses the already available distributed storage to permanently store the process state.

The second issue is the currently unaddressed research goal of workflow instance agility. This research goal concerns itself with changes in the business situation that cannot be predicted in advance, preventing the workflow designer to include them in a workflow model during design and implementation. This means that the enacted workflow instance needs to be adapted while it is executing on the system. In section 3.10, ADEPT2 was introduced which uses a runtime approach for modifying the workflow instance during execution; however, the approach uses a task-based approach.

An interesting avenue for adding workflow instance agility to goal-oriented models may be offered by the plan-level of goal-oriented workflows: If a particular goal can no longer be reached due to an unpredictable change in the business situation, the user could be enabled by the system to add one or more additional plans to the goal which are adapted to the changed situation and are capable of reaching the original goal under the new circumstances.

A more complex approach would also allow the replacement of whole subhierarchies if the decomposition of business goals require change as well. This approach would be more difficult since it has to take active goal instances into account and deal with them appropriately, perhaps by dropping them deliberately or some other rollover process supplied by the user.

In particular the plan addition would also enable non-technical users to make modifications to workflow instances in a safe way if they are provided with a sufficiently comprehensive plan library. This would provide an additional advantage over the ADEPT2 approach for runtime adaption of workflow instances for non-technical workflow partipants.

Overall, the developed system provides better support for collaborative and creative business processes that have a long execution time, requiring a large degree of autonomy for the workflow participants as well as a high degree of runtime agility and flexibility. This promotes an improved application of business process management in areas that were previously difficult to accommodate and which were therefore largely informally organized.

# Publications

- [81] K. Jander and W. Lamersdorf. GPMN-Edit: High-level and Goal-oriented Workflow Modeling. In *WowKiVS 2011*, volume 37, pages 146–157. Electronic Communications of the EASST, 2011
- [82] K. Jander and W. Lamersdorf. Jadex WfMS: Distributed Workflow Management for Private Clouds. In *Conference on Networked Systems (NetSys)*, pages 84–91. IEEE Xplore, 2013
- [83] K. Jander and W. Lamersdorf. Compact and Efficient Agent Messaging. In M. Dastani, J. F. Hübner, and B. Logan, editors, *Programming Multi-Agent Systems*, volume 7837 of *Lecture Notes in Computer Science*, pages 108–122. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38699-2
- [84] K. Jander and W. Lamersdorf. Distributed Event Processing for Goal-oriented Workflows. In D. E Camacho, L. Braubach, S. Venticinque, and C. Badica, editors, *8th International Symposium on Intelligent Distributed Computing (IDC-2014)*, volume 570 of *Intelligent Distributed Computing*, pages 49–58. Springer Berlin, Heidelberg, 1 2015
- [85] K. Jander, L. Braubach, and A. Pokahr. EnvSupport: A Framework for Developing Virtual Environments. In *Seventh International Workshop From Agent Theory to Agent Implementation (AT2AI-7)*, pages 471–476. Austrian Society for Cybernetic Studies, 2010
- [86] K. Jander, L. Braubach, A. Pokahr, and W. Lamersdorf. Validation of Agile Workflows using Simulation. In M. Dastani, A. E. F. Seghrouchni, J. Hübner, and J. Leite, editors, *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, pages 39–55. Springer Berlin / Heidelberg, 8 2011
- [87] K. Jander, L. Braubach, A. Pokahr, W. Lamersdorf, and K.-J. Wack. Goal-oriented Processes with GPMN. *International Journal on Artificial Intelligence Tools (IJAIT)*, 20(6):1021–1041, 12 2011
- [88] K. Jander, L. Braubach, and W. Lamersdorf. Distributed Monitoring and Workflow Management for Goal-oriented Workflows. *Journal of Concurrency and Computation: Practice and Experience*, pages 1324–1335, 2015. ISSN 1532-0634

- [89] K. Jander, L. Braubach, and A. Pokahr. Extending the Communication Capabilities of Agents. *Journal of Computing and Informatics*, 34:1001–1029, 2015

- [24] L. Braubach, A. Pokahr, K. Jander, W. Lamersdorf, and B. Burmeister. Go4Flex: Goal-Oriented Process Modelling. In *Intelligent Distributed Computing IV - Proceedings of the 4th International Symposium on Intelligent Distributed Computing - IDC 2010, Tangier, Morocco, September 2010*, pages 77–87, 2010

- [25] L. Braubach, A. Pokahr, and K. Jander. JadexCloud - An Infrastructure for Enterprise Cloud Applications. In *Multiagent System Technologies - 9th German Conference, MATES 2011, Berlin, Germany, October 6-7, 2011. Proceedings*, pages 3–15, 2011

- [26] L. Braubach, K. Jander, and A. Pokahr. A Practical Security Infrastructure for Open Multi-Agent Systems. In M. Thimm M. Klusch, M. Paprzycki, editor, *Proceedings of Ninth German conference on Multi-Agent System TEchnologieS (MATES-2013)*, pages 29–43. Springer, 2013

- [27] L. Braubach, K. Jander, and A. Pokahr. A Middleware for Managing Non-Functional Requirements in Cloud PaaS. In *International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 83–92. IEEE, 2014

- [28] L. Braubach, K. Jander, and A. Pokahr. High-Volume Data Streaming with Agents. In Filip Zavoral, Jason J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII*, volume 511 of *Studies in Computational Intelligence*, pages 199–209. Springer International Publishing, 2014. ISBN 978-3-319-01570-5

- [131] A. Pokahr, L. Braubach, and K. Jander. Unifying Agent and Component Concepts: Jadex Active Components. In *Multiagent System Technologies, 8th German Conference, MATES 2010, Leipzig, Germany, September 27-29, 2010. Proceedings*, pages 100–112, 2010

- [132] A. Pokahr, L. Braubach, and K. Jander. The Jadex Project: Programming Model. In L. C. Jain and M. Ganzha, editors, *Multiagent Systems and Applications*, pages 21–53. Springer Berlin/Heidelberg, 2012. ISBN 978-3-642-33322-4

# Bibliography

[1] Wil Van Der Aalst and Kees Van Hee. *Workflow Management: Models, Methods, and Systems.* MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-01189-1.

[2] SAP AG. SAP Exchange Infrastructure (SAP XI). URL http://searchsap.techtarget.com/definition/SAP-Exchange-Infrastructure. Last visited 2016-04-26.

[3] G. Agha. *Actors: A Model of Concurrent Computation in Distributed Systems.* MIT Press, Cambridge, MA, USA, 1986. ISBN 0-262-01092-5.

[4] Alfresco. Activiti BPM Platform. URL http://activiti.org/. Last visited 2016-04-28.

[5] T. Ami and R. Sommer. Comparison and evaluation of business process modelling and management tools. *International Journal of Services and Standards*, 3(2):249–261, 2007.

[6] A.S.M.E. Special committee on standardization of Therbligs, Process Charts, and Their Symbols. A.S.M.E. standard operation and flow process charts, 1947.

[7] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries, J. Kern, B. Marick, R. C. Martin, S. Mellor, K. Schwaber, J. Sutherland, and D. Thomas. Manifesto for Agile Software Development, 2001. URL http://agilemanifesto.org/. Last visited 2016-04-26.

[8] F. Bellifemine, F. Bergenti, G. Caire, and A. Poggi. JADE - A Java Agent Development Framework. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 125–147. Springer, 2005.

[9] F. Bellifemine, G. Caire, and D. Greenwood. *Developing Multi-Agent systems with JADE.* John Wiley & Sons, 2007.

[10] T. Bellwood. UDDI Version 2.04 API Specification, jul 2002. URL http://www.uddi.org/pubs/ProgrammersAPI-V2.04-Published-20020719.htm. Last visited 2016-04-27.

[11] *BPMN Poster*. Berliner BPM-Offensive (BPMB), 2.0 edition, 2011. URL http://www.bpmb.de/index.php/BPMNPoster. Last visited 2016-04-26.

[12] E. Best and M. Koutny. Petri Net Semantics of Priority Systems. In *Selected Papers of the Second Workshop on Concurrency and Compositionality*, pages 175–215, Essex, UK, 1992. Elsevier Science Publishers Ltd.

[13] H. H. Bi and J. L. Zhao. Applying Propositional Logic to Workflow Verification. *Information & Software Technology*, 5:293–318, July 2004. ISSN 1385-951X.

[14] P. V. Biron, A. Malhotra, D. Peterson, S. Gao, C. M. Sperberg-McQueen, and H. S. Thompson, editors. *XML Schema Part 2: Datatypes.* W3C Recommendation. W3C, April 2012. URL http://www.w3.org/TR/xmlschema11-2/. Last visited 2016-04-28.

[15] R. Bordini, Jomi F. Hübner, and R. Vieira. Jason and the Golden Fleece of Agent-Oriented Programming. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 3–37. Springer, 2005.

[16] R. H. Bordini and J. F. Hübner. BDI Agent Programming in AgentSpeak Using Jason. In *Proceedings of 6th International Workshop on Computational Logic in Multi-Agent Systems (CLIMA VI). VOLUME 3900 OF LNCS*, pages 143–164. Springer, 2005.

[17] M. Bratman. *Intention, Plans, and Practical Reason.* Harvard University Press, 1987.

[18] L. Braubach. *Architekturen und Methoden zur Entwicklung verteilter agentenorientierter Softwaresysteme.* Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 1 2007.

[19] L. Braubach and A. Pokahr. Conceptual Integration of Agents with WSDL and RESTful Web Services. In *Programming Multi-Agent Systems - 10th International Workshop, ProMAS 2012, Valencia, Spain, June 5, 2012, Revised Selected Papers.*

[20] L. Braubach and A. Pokahr. Representing Long-Term and Interest BDI Goals. In *Proc. of (ProMAS-7)*, pages 29–43. IFAAMAS Foundation, 5 2009.

[21] L. Braubach and A. Pokahr. Addressing Challenges of Distributed Systems Using Active Components. In *Intelligent Distributed Computing V - Proceedings of the 5th International Symposium on Intelligent Distributed Computing - IDC 2011, Delft, The Netherlands - October 2011*, pages 141–151, 2011.

[22] L. Braubach, A. Pokahr, and W. Lamersdorf. Jadex: A BDI Agent System Combining Middleware and Reasoning. In R. Unland, M. Calisti, and M. Klusch, editors, *Software Agent-Based Applications, Platforms and Development Kits*, pages 143–168. Birkhäuser, 2005.

[23] L. Braubach, A. Pokahr, D. Moldt, and W. Lamersdorf. Goal Representation for BDI Agent Systems. In *Proc. of (ProMAS 2004)*, pages 44–65. Springer, 2005.

[24] L. Braubach, A. Pokahr, K. Jander, W. Lamersdorf, and B. Burmeister. Go4Flex: Goal-Oriented Process Modelling. In *Intelligent Distributed Computing IV - Proceedings of the 4th International Symposium on Intelligent Distributed Computing - IDC 2010, Tangier, Morocco, September 2010*, pages 77–87, 2010.

[25] L. Braubach, A. Pokahr, and K. Jander. JadexCloud - An Infrastructure for Enterprise Cloud Applications. In *Multiagent System Technologies - 9th German Conference, MATES 2011, Berlin, Germany, October 6-7, 2011. Proceedings*, pages 3–15, 2011.

[26] L. Braubach, K. Jander, and A. Pokahr. A Practical Security Infrastructure for Open Multi-Agent Systems. In M. Thimm M. Klusch, M. Paprzycki, editor, *Proceedings of Ninth German conference on Multi-Agent System TEchnologieS (MATES-2013)*, pages 29–43. Springer, 2013.

[27] L. Braubach, K. Jander, and A. Pokahr. A Middleware for Managing Non-Functional Requirements in Cloud PaaS. In *International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 83–92. IEEE, 2014.

[28] L. Braubach, K. Jander, and A. Pokahr. High-Volume Data Streaming with Agents. In Filip Zavoral, Jason J. Jung, and Costin Badica, editors, *Intelligent Distributed Computing VII*, volume 511 of *Studies in Computational Intelligence*, pages 199–209. Springer International Publishing, 2014. ISBN 978-3-319-01570-5.

[29] H.-J. Bullinger, G. Wiedmann, and J. Niemeier. *Business reengineering: aktuelle Managementkonzepte in Deutschland: Zukunftsperspektiven und Stand der Umsetzung*. IRB-Verlag, 1995.

[30] B. Burmeister, M. Arnold, F. Copaciu, and G. Rimassa. BDI-agents for agile goal-oriented business processes. In *AAMAS '08*, pages 37–44. IFAAMAS, 2008.

[31] M. Calisti and D. Greenwood. Goal-Oriented Autonomic Process Modeling and Execution for Next Generation Networks. In *Proc. of MACE '08*, pages 38–49, Berlin, Heidelberg, 2008. Springer. ISBN 978-3-540-87354-9. doi: http://dx.doi.org/10.1007/978-3-540-87355-6_4.

[32] C. Castelfranchi. Guarantees for Autonomy in Cognitive Agent Architecture. In *Proceedings of the Workshop on Agent Theories, Architectures, and Languages on Intelligent Agents*, ECAI-94, pages 56–70, New York, NY, USA, 1995. Springer-Verlag New York, Inc. ISBN 3-540-58855-8.

[33] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber. Bigtable: A Distributed Storage System for Structured Data. *ACM Trans. Comput. Syst*, 26(2):4:1–4:26, 2008. ISSN 0734-2071.

[34] J. Clark and M. Murata, editors. *RELAX NG Specification.* Committee Specification. OASIS, December 2001. URL http://relaxng.org/spec-20011203.html. Last visited 2016-04-28.

[35] T. H. Davenport and J. E. Short. The New Industrial Engineering: Information Technology and Business Process Redesign. *Sloan Management Review*, 31(4): 11–27, 1990.

[36] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev*, 41(6):205–220, 2007. ISSN 0163-5980.

[37] F. Demuth. Verteilter Speicherdienst für Nutzer-integrierende dynamische Cloud-Umgebungen. Diplomarbeit, Distributed Systems and Information Systems Group, Computer Science Department, University of Hamburg, November 2014. (in German).

[38] F. DeRemer and H.H. Kron. Programming-in-the-Large Versus Programming-in-the-Small. *IEEE Transactions on Software Engineering*, 2(2):80–86, 1976. ISSN 0098-5589.

[39] L. P. Deutsch. DEFLATE Compressed Data Format Specification version 1.3. RFC 1951, 5 1996. URL http://www.ietf.org/rfc/rfc1951.txt. Last visited 2016-04-28.

[40] Merriam-Webster Online Dictionary. Autonomy, 2014. URL http://www.merriam-webster.com/dictionary/autonomy. Last visited 2016-04-28.

[41] M. Earl and B. Kahn. How new is business process redesign? *European Management Journal*, 12(1):20–30, 3 1994.

[42] T. Erl. *Service-Oriented Architecture: A Field Guide to Integrating XML and Web Services.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131428985.

[43] T. Erl. *Service-Oriented Architecture: Concepts, Technology, and Design.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2005. ISBN 0131858580.

[44] T. Erl, B. Carlyle, C. Pautasso, and R. Balasubramanian. *SOA with REST - Principles, Patterns and Constraints for Building Enterprise Solutions with REST.* The Prentice Hall service technology series. Prentice Hall, 2013. ISBN 978-0-13-701251-0.

[45] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616, jun 1999. URL http://www.ietf.org/rfc/rfc2616.txt. Last visited 2016-04-28.

[46] C. Forgy. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem.

[47] Charles Lanny Forgy. *On the Efficient Implementation of Production Systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979.

[48] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *18th IEEE international conference on automated software engineering, Montreal, Canada, 2003*, 2003. ISBN 0-7695-2035-9.

[49] Apache Foundation. Apache ODE (Orchestration Director Engine). URL http://ode.apache.org/. Last visited 2016-04-26.

[50] Apache Foundation. Apache Derby, 2014. URL https://db.apache.org/derby/. Last visited 2016-04-26.

[51] Eclipse Foundation. Eclipse BPMN Modeler. http://www.eclipse.org/bpmn/, Last visited 2016-04-26, archived project, 2009.

[52] Eclipse Foundation. BPMN2 Modeler. http://eclipse.org/bpmn2-modeler/, Last visited 2016-04-26, 2014.

[53] FIPA. *FIPA SL Content Language Specification.* Foundation for Intelligent Physical Agents (FIPA), December 2002. URL http://www.fipa.org. Last visited 2016-04-26.

[54] S. Franklin and A. C. Graesser. Is it an Agent, or Just a Program?: A Taxonomy for Autonomous Agents". In J. Müller, M. Wooldridge, and N. Jennings, editors, *Proceedings of the 3rd Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages (ATAL 1996)"*, pages 21–35. Springer, 1997.

[55] J. R. Galbraith. *Designing Matrix Organizations that Actually Work: How IBM, Procter & Gamble and Others Design for Success.* Jossey-Bass, 2008. ISBN 9780470316313.

[56] M. R. Genesereth and S. P. Ketchpel. Software Agents. *Commun. ACM*, 37 (7):48–ff., July 1994. ISSN 0001-0782.

[57] S. Gilbert and N. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-tolerant Web Services. *SIGACT News*, 33(2):51–59, June 2002. ISSN 0163-5700.

[58] F.B. Gilbreth and L.M. Gilbreth. *Process Charts.* American Society of Mechanical Engineers, 1921.

[59] C. Girault and R. Valk. *Petri Nets for System Engineering: A Guide to Modeling, Verification, and Applications.* Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001. ISBN 3540412174.

[60] A. Goh, Y.-K. Koh, and D. S. Domazet. ECA rule-based support for workflows. *AI in Engineering*, 15(1):37–46, 2001.

[61] C. F. Goldfarb. *The SGML Handbook.* Oxford University Press, Inc., New York, NY, USA, 1990. ISBN 0-10-853737-9.

[62] J. Gosling, B. Joy, G. Steele, G. Bracha, and A. Buckley. *The Java Language Specification Third Second Edition.* Oracle Inc., 2014. URL http://docs.oracle.com/javase/specs/jls/se8/html/. Last visited 2016-04-27.

[63] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), apr 2007. URL http://www.w3.org/TR/soap12/. Last visited 2016-04-28.

[64] W. Sesselmann H. J. Schmelzer. *Geschäftsprozessmanagement in der Praxis.* Hanser Fachbuchverlag, 2008.

[65] B. Hayes-Roth. An Architecture for Adaptive Intelligent Systems. *Artificial Intelligence*, 72(1–2):329–365, 1995.

[66] H. H. Hinterhuber, G. Handlbauer, and K. Matzler. *Kundenzufriedenheit durch Kernkompetenzen: eigene Potentiale erkennen, entwickeln, umsetzen.* Hanser Fachbuchverlag, 1997.

[67] W. Hoffmann, J. Kirsch, and A.-W. Scheer. Modellierung mit Ereignisgesteuerten Prozeßketten: Methodenhandbuch. *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, 101, 1992.

[68] D. Hollingsworth. Workflow Management System Reference Model, 1995. URL http://www.wfmc.org/. Last visited 2016-04-27.

[69] J. R. Holmevik. Compiling Simula: A historical study of technological genesis. *IEEE Annals in the History of Computing*, 16(4):25–37, 12 1994.

[70] C. Homburg and B. Rudolph. Wie zufrieden sind Ihre Kunden tatsächlich? Harvard Business Manager, 1995.

[71] N. Howden, R. Rönnquist, A. Hodgson, and A. Lucas. JACK Intelligent Agents - Summary of an Agent Infrastructure. In *Proceedings of the 5th ACM International Conference on Autonomous Agents (AGENTS 2001)*, 2001.

[72] M. Imai. *Gemba Kaizen: A Commonsense Approach to a Continuous Improvement Strategy.* McGraw-Hill, 2012. ISBN 9780071790352.

[73] IBM Inc. WebSphere Process Server, . URL http://www-01.ibm.com/software/integration/wps/. Last visited 2016-04-28.

[74] Microsoft Inc. Microsoft BizTalk, . URL http://www.microsoft.com/en-us/server-cloud/products/biztalk/. Last visited 2016-04-28.

[75] Oracle Inc. Oracle BPEL Process Manager Data Sheet, 2009. URL http://www.oracle.com/us/products/middleware/application-server/bpel-process-manager-ds-066554.pdf. Last visited 2016-04-26.

[76] Oracle Inc. Java Remote Method Invocation, 2010. URL http://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmiTOC.html. Last visited 2016-04-28.

[77] Oracle Inc. *MySQL 5.7 Reference Manual Replication*. Redwood Shores, CA, USA, 2014. URL http://dev.mysql.com/doc/refman/5.7/en/replication.html. Last visited 2016-04-27.

[78] D. Ings, L. Clément, D. König, V. Mehta, R. Mueller, R. Rangaswamy, M. Rowley, and I. Trickovic. WS-BPEL Extension for People (BPEL4People) Specification Version 1.1. OASIS Committee Specification, August 2010. URL http://docs.oasis-open.org/bpel4people/bpel4people-1.1.html. Last visited 2016-04-27.

[79] D. Ings, L. Clément, D. König, V. Mehta, R. Mueller, R. Rangaswamy, M. Rowley, and I. Trickovic. Web Services Human Task (WS-HumanTask) Specification Version 1.1. OASIS Committee Specification Draft 12 / Public Review Draft 05., July 2012. URL http://docs.oasis-open.org/bpel4people/ws-humantask-1.1.html. Last visited 2016-04-27.

[80] K. Jander. Validating Agile Business Processes with Simulation-based Testing. Master's thesis, University of Hamburg, 2009.

[81] K. Jander and W. Lamersdorf. GPMN-Edit: High-level and Goal-oriented Workflow Modeling. In *WowKiVS 2011*, volume 37, pages 146–157. Electronic Communications of the EASST, 2011.

[82] K. Jander and W. Lamersdorf. Jadex WfMS: Distributed Workflow Management for Private Clouds. In *Conference on Networked Systems (NetSys)*, pages 84–91. IEEE Xplore, 2013.

[83] K. Jander and W. Lamersdorf. Compact and Efficient Agent Messaging. In M. Dastani, J. F. Hübner, and B. Logan, editors, *Programming Multi-Agent Systems*, volume 7837 of *Lecture Notes in Computer Science*, pages 108–122. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38699-2.

[84] K. Jander and W. Lamersdorf. Distributed Event Processing for Goal-oriented Workflows. In D. E Camacho, L. Braubach, S. Venticinque, and C. Badica, edi-

tors, *8th International Symposium on Intelligent Distributed Computing (IDC-2014)*, volume 570 of *Intelligent Distributed Computing*, pages 49–58. Springer Berlin, Heidelberg, 1 2015.

[85] K. Jander, L. Braubach, and A. Pokahr. EnvSupport: A Framework for Developing Virtual Environments. In *Seventh International Workshop From Agent Theory to Agent Implementation (AT2AI-7)*, pages 471–476. Austrian Society for Cybernetic Studies, 2010.

[86] K. Jander, L. Braubach, A. Pokahr, and W. Lamersdorf. Validation of Agile Workflows using Simulation. In M. Dastani, A. E. F. Seghrouchni, J. Hübner, and J. Leite, editors, *Languages, Methodologies, and Development Tools for Multi-Agent Systems*, pages 39–55. Springer Berlin / Heidelberg, 8 2011.

[87] K. Jander, L. Braubach, A. Pokahr, W. Lamersdorf, and K.-J. Wack. Goal-oriented Processes with GPMN. *International Journal on Artificial Intelligence Tools (IJAIT)*, 20(6):1021–1041, 12 2011.

[88] K. Jander, L. Braubach, and W. Lamersdorf. Distributed Monitoring and Workflow Management for Goal-oriented Workflows. *Journal of Concurrency and Computation: Practice and Experience*, pages 1324–1335, 2015. ISSN 1532-0634.

[89] K. Jander, L. Braubach, and A. Pokahr. Extending the Communication Capabilities of Agents. *Journal of Computing and Informatics*, 34:1001–1029, 2015.

[90] JBoss. jBPM Business Process Management Suite. URL http://www.jbpm.org/. Last visited 2016-04-27.

[91] N. R. Jennings and M. J. Wooldridge. *Agent Technology Foundations, Applications, and Markets.* Springer Verlag, Springer Berlin / Heidelberg, 1998. ISBN 3540635912.

[92] K. Jensen and L. M. Kristensen. *Coloured Petri Nets.* Springer Verlag, Springer Berlin / Heidelberg, 2009. ISBN 9783642425813.

[93] R. S. Kaplan and D. P. Norton. *The Balanced Scorecard: Translating Strategy into Action.* Harvard Business Review Press, 1996. ISBN 9780875846514.

[94] R. S. Kaplan and D. P. Norton. Using the Balanced Scorecard as a Strategic Management System. *Harvard Business Review*, (January-February):75–85, 1996.

[95] G. Keller, M. Nuttigens, and A.-W. Scheer. Semantische Prozessmodellierung auf der Grundlage: Ereignisgesteuerter Prozessketten (EPK). *Veröffentlichungen des Instituts für Wirtschaftsinformatik*, 89, 1992.

[96] F. Klügl. *Multiagentensimulation - Konzepte, Werkzeuge, Anwendung.* Addison Wesley, 2001.

[97] D. Krafzig, K. Banke, and D. Slama. *Enterprise SOA: Service-Oriented Architecture Best Practices (The Coad Series).* Prentice Hall PTR, Upper Saddle River, NJ, USA, 2004. ISBN 0131465759.

[98] Olaf Kummer. *Referenznetze.* Logos Verlag, 2002.

[99] F. Leymann. Web Services Flow Language (WSFL 1.0). Technical report, IBM, May 2001.

[100] F. Leymann and D. Roller. *Production workflow concepts and techniques.* Prentice Hall PTR, 2000. ISBN 0130217530.

[101] X. Li. What's So Bad About Rule-Based Programming? *IEEE Software*, 8(5): 103–105, 1991.

[102] Y. Lin. *Semantic Annotation of Process Models: Facilitating Process Knowledge Management.* PhD thesis, Norwegian University of Science and Technology, 2008.

[103] JGraph Ltd. JGraphX Diagram Framework. URL http://www.jgraph.com/. Last visited 2016-04-28.

[104] M. Luck, P. McBurney, O. Shehory, and S. Willmott. *Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing).* AgentLink, 2005.

[105] P. Maes. Agents that Reduce Work and Information Overload. *Communications of the ACM*, 37(7):30–40, 1994.

[106] P. Maes. Artificial Life Meets Entertainment: Lifelike Autonomous Agents. *Communications of the ACM*, 38(11):108–114, 1995.

[107] M. Maloney, D. Beech, N. Mendelsohn, S. Gao, C. M. Sperberg-McQueen, and H. S. Thompson, editors. *XML Schema Part 1: Structures.* W3C Recommendation. W3C, apr 2012. URL http://www.w3.org/TR/xmlschema11-1/. Last visited 2016-04-26.

[108] J. Marino and M. Rowley. *Understanding SCA (Service Component Architecture).* Addison-Wesley Professional, 1st edition, 2009. ISBN 0321515080, 9780321515087.

[109] J. Martin and J. J. Odell. *Object-Oriented Methods.* Prentice Hall PTR, Upper Saddle River, NJ, USA, 1st edition, 1994. ISBN 0136308562.

[110] P. Massuthe, W. Reisig, and K. Schmidt. An Operating Guideline Approach to the SOA. *Annals of Mathematics Computing and Teleinformatics*, 1:35–43, 2005.

[111] B. M. McCarthy and R. Stauffer. Six Sigma: Enhancing Six Sigma Through Simulation with iGrafx Process for Six Sigma. In *Proceedings of the 33Nd Conference on Winter Simulation*, WSC '01, pages 1241–1247, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7803-7309-X.

[112] D. McCarthy and U. Dayal. The Architecture of an Active Database Management System. *SIGMOD Rec.*, 18(2):215–224, June 1989. ISSN 0163-5808.

[113] J. Meng, S. Y. W. Su, H. Lam, A. Helal, J. Xian, X. Liu, and S. Yang. DynaFlow: a dynamic inter-organisational workflow management system. In *Int. Journal of Business Process Integration and Management (IJBPIM)*, volume 1, pages 101–115. Inderscience Enterprises Ltd., 2005.

[114] A. Miguel. WS-BPEL 2.0 Tutorial. http://eclipse.org/tptp/platform/documents/design/choreography_html/tutorials/wsbpel_tut.html, Oct 2005.

[115] J. P. Müller. *The Design of Intelligent Agents - A Layered Approach.* Springer, 1996.

[116] R. Müller and E. Rahm. Rule-Based Dynamic Modification of Workflows in a Medical Domain. In Alejandro P. Buchmann, editor, *Datenbanksysteme in Büro, Technik und Wissenschaft (BTW), GI-Fachtagung, Freiburg, 1.-3. März 1999, Proceedings. Informatik Aktuell*, pages 429–448. Springer, 1999.

[117] I. Nunes, C. J. P. De Lucena, and M. Luck. BDI4JADE: a BDI layer on top of JADE, 2011.

[118] WSBPEL 2.0. *Web Services Business Process Execution Language (WSPBEL) Specification.* OASIS, version 2.0 edition, 2007.

[119] OMG. *The Common Object Request Broker: Architecture and Specification.* Object Management Group (OMG), revision 1.1 edition, December 1991.

[120] BPMN 1.1. *Business Process Modeling Notation (BPMN) Specification.* Object Management Group (OMG), version 1.1 edition, February 2008. URL http://www.bpmn.org/Documents/BPMN_1-1_Specification.pdf. Last visited 2016-04-26.

[121] BPMN 2.0. *Business Process Modeling Notation (BPMN) Specification.* Object Management Group (OMG), version 2.0 edition, January 2011. URL http://www.omg.org/spec/BPMN/2.0/. Last visited 2016-04-28.

[122] CMMN 1.0. *Case Management Model and Notation (CMMN).* Object Management Group (OMG), version 1.0 edition, May 2014. URL http://www.omg.org/spec/CMMN/1.0/. Last visited 2016-04-27.

[123] M. Osterloh and J. Frost. *Prozessmanagement als Kernkompetenz: Wie Sie Business Reengineering strategisch nutzen können.* Gabler Verlag, 2006. ISBN 9783834902320.

[124] C. Ouyang, M. Dumas, A. ter Hofstede, and W. van der Aalst. From BPMN Process Models to BPEL Web Services. In *Proc. of ICWS '06*, pages 285–292. IEEE, 2006. ISBN 0-7695-2669-1. doi: http://dx.doi.org/10.1109/ICWS.2006.67.

[125] P.Alpar, R. Alt, F. Bensberg, H. L. Grob, P. Weimann, and R. Winter, editors. *Anwendungsorientierte Wirtschaftsinformatik: Strategische Planung, Entwicklung und Nutzung von Informationssystemen.* Springer Vieweg, Wiesbaden, 2014. ISBN 978-3-658-00520-7.

[126] E. Pierce. A simple flowchart for troubleshooting a broken lamp. http://en.wikipedia.org/wiki/File:LampFlowchart.svg, May 2006.

[127] A. Pokahr and L. Braubach. From a Research to an Industrial-Strength Agent Platform: Jadex V2. In Hans-Georg Fill Hans Robert Hansen, Dimitris Karagiannis, editor, *Business Services: Konzepte, Technologien, Anwendungen - 9. Internationale Tagung Wirtschaftsinformatik (WI 2009)*, pages 769–778. Österreichische Computer Gesellschaft, 2 2009.

[128] A. Pokahr and L. Braubach. The Active Components Approach for Distributed Systems Development. *International Journal of Parallel, Emergent and Distributed Systems*, 28(4):321–369, 2013.

[129] A. Pokahr, L. Braubach, and W. Lamersdorf. A Goal Deliberation Strategy for BDI Agent Systems. In T. Eymann, F. Klügl, W. Lamersdorf, M. Klusch, and M. Huhns, editors, *Proceedings of the 3rd German conference on Multi-Agent System TEchnologieS (MATES-2005).* Springer, 2005.

[130] A. Pokahr, L. Braubach, and W. Lamersdorf. Jadex: A BDI Reasoning Engine. In R. Bordini, M. Dastani, J. Dix, and A. El Fallah Seghrouchni, editors, *Multi-Agent Programming: Languages, Platforms and Applications*, pages 149–174. Springer, 2005.

[131] A. Pokahr, L. Braubach, and K. Jander. Unifying Agent and Component Concepts: Jadex Active Components. In *Multiagent System Technologies, 8th German Conference, MATES 2010, Leipzig, Germany, September 27-29, 2010. Proceedings*, pages 100–112, 2010.

[132] A. Pokahr, L. Braubach, and K. Jander. The Jadex Project: Programming Model. In L. C. Jain and M. Ganzha, editors, *Multiagent Systems and Applications*, pages 21–53. Springer Berlin/Heidelberg, 2012. ISBN 978-3-642-33322-4.

[133] Alexander Pokahr. *Programmiersprachen und Werkzeuge zur Entwicklung verteilter agentenorientierter Softwaresysteme.* Dissertation, Universität Hamburg, Fachbereich Informatik, Vogt-Kölln-Str. 30, 22527 Hamburg, Germany, 1 2007.

[134] M. E. Porter. *Competitive advantage: Creating and sustaining superior performance.* Free Press, New York and London, 1985. ISBN 9780029250907.

[135] G. Probst and B. Büchel. *Organisationales Lernen: Wettbewerbsvorteil der Zukunft.* Dr. Th. Gabler Verlag, 1994. ISBN 9783409230247.

[136] M. Reichert and P. Dadam. Enabling Adaptive Process-aware Information Systems with ADEPT2. In Jorge Cardoso and Wil van der Aalst, editors, *Handbook of Research on Business Process Modeling*, pages 173–203. Information Science Reference, Hershey, New York, March 2009.

[137] W. Reisig. *Petri Nets: An Introduction.* Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 0-387-13723-8.

[138] S. Russell and P. Norvig. *Artifical Intelligence: A Modern Approach.* Prentice-Hall, 2003.

[139] W. Sadiq and M. E. Orlowska. Analyzing process models using graph reduction techniques. *Information Systems*, 25(2):117 – 134, 2000. ISSN 0306-4379. The 11th International Conference on Advanced Information System Engineering.

[140] A.-W. Scheer and M. Nüttgens. ARIS Architecture and Reference Models for Business Process Management. In *Business Process Management, Models, Techniques, and Empirical Studies.* Springer, 2000.

[141] A.-W. Scheer, O. Thomas, and O. Adam. Process Modeling Using Event-Driven Process Chains. In *Process-Aware Information Systems.* Wiley, 2005. ISBN 978-0-471-66306-5.

[142] C. Schroth and T. Janner. Web 2.0 and SOA: Converging Concepts Enabling the Internet of Services. *IT Professional*, 9(3):36–41, 2007. ISSN 1520-9202.

[143] G. Schuh, T. Friedli, and M. A. Kurr. *Prozessorientierte Reorganisation: Reengineering-Projekte professionell gestalten und umsetzen.* Hanser Fachbuchverlag, 2006. ISBN 9783446407206.

[144] N. Seel. *Agent Theories and Architectures.* PhD thesis, Surrey University, Guildford, UK, 1989.

[145] Y. Shoham. Agent-oriented programming. *Artificial Intelligence*, 60(1):51–92, 1993.

[146] P. C Sander T. P.J Berden, A. C Brombacher. The building bricks of product quality: An overview of some basic concepts and principles. *International Journal of Production Economics*, 67(1):3–15, 8 2000.

[147] S. Thatte. XLANG - Web Services for Business Process Design. Technical report, Microsoft, 2001.

[148] The Unicode Consortium. *The Unicode Standard.* Addison Wesley, 2006.

[149] Roger Tregear. Business Process Standardization. In J. vom Brocke and M. Rosemann, editors, *Handbook on Business Process Management 2*, International Handbooks on Information Systems, pages 307–327. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-01981-4.

[150] D. Vahs. *Organisation: Einführung in die Organisationstheorie und -praxis.* Schäffer-Poeschel, 2005. ISBN 9783791023571.

[151] R. Valk. Modelling of task flow in systems of functional units. 124 FBI-HH-B-124/87, "University of Hamburg", Hamburg, Germany, 1987.

[152] W. van der Aalst. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66, Feb 1998.

[153] W. van der Aalst, D. Moldt, R. Valk, and F. Wienberg. Enacting interorganizational workflows using nets in nets. In Jörg Becker, Michael zur Mühlen, and Michael Rosemann, editors, *Proceedings of the 1999 Workflow Management Conference Workflow-based Applications, Münster, Nov. 9th 1999*, Working Paper Series of the Department of Information Systems, pages 117–136, University of Münster, Department of Information Systems, Steinfurter Str. 109, 48149 Münster, 1999. Working Paper No. 70.

[154] W. M. P. van der Aalst and A. H. M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, June 2005.

[155] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag. ISBN 3-540-40318-3.

[156] W. M. P. van der Aalst, A. H. M. ter Hofstede, and M. Weske. Business Process Management: A Survey. In *Business Process Management*, pages 1–12, 2003.

[157] MG van't Veer. Management of improvements. *International Journal of Project Management*, 9(2):82–85, 5 1991.

[158] K. Vergidis, A. Tiwari, and B. Majeed. Business Process Analysis and Optimization: Beyond Reengineering. *Trans. Sys. Man Cyber Part C*, 38(1):69–82, January 2008. ISSN 1094-6977.

[159] A. Waller, M. Clark, and L. Enstone. L-SIM: simulating BPMN diagrams with a purpose built engine. In F. L. Perrone, B. Lawson, J. Liu, and F. P. Wieland, editors, *Winter Simulation Conference*, pages 591–597. WSC, 2006. ISBN 1-4244-0501-7.

[160] B. Weber, S. Sadiq, and M. Reichert. Beyond Rigidity - Dynamic Process Lifecycle Support: A Survey on Dynamic Changes in Process-aware Information Systems. *Computer Science - Research and Development*, 23(2):47–65, May 2009.

[161] M. Weske. *Business Process Management Concepts, Languages, Architectures.* Springer Verlag, 2007. ISBN 978-3-540-73521-2.

[162] C. Wiesner, S. Lhomme, and J. Cannon. Extensible Binary Meta-Language (EBML). Website, http://ebml.sourceforge.net/, 2012. URL http://ebml.sourceforge.net/. Last visited 2016-04-28.

[163] M. Wooldridge. *An Introduction to Multiagent Systems.* Wiley, 2002. ISBN 0-471-49691-X.

[164] M. Wooldridge and N. Jennings. Intelligent Agents: Theory and Practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.

[165] *Workflow Standard Process Definition Interface – XML Process Definition Language.* Workflow Management Coalition (WfMC), version 2.2 edition, August 2012. URL http://www.xpdl.org/standards/xpdl-2.2/XPDL%202.2%20(2012-08-30).pdf. Last visited 2016-04-27.

[166] W3C. *Extensible Markup Language (XML).* World Wide Web Consortium (W3C), February 2004. URL http://www.w3.org/TR/2004/REC-xml-20040204. Last visited 2016-04-26.

[167] WSDL 2.0. *Web Services Description Language (WSDL) Version 2.0.* World Wide Web Consortium (W3C), version 2.0 edition, June 2007. URL http://www.w3.org/TR/wsdl20-primer/. Last visited 2016-04-27.

[168] K. Wüllenweber, D. Beimborn, T. Weitzel, and W. König. The impact of process standardization on business process outsourcing success. *Information Systems Frontiers*, 10(2):211–224, 2008.

[169] F. Yergeau. UTF-8, a transformation format of ISO 10646. RFC 3629, 11 2003. URL http://www.ietf.org/rfc/rfc3629.txt. Last visited 2016-04-26.

[170] M. Zairi and M. A. Youssef. Quality function deployment: a main pillar for successful total quality management and product development. *International Journal of Quality & Reliability Management*, 12(6):9–23, 1995.

## Eidesstattliche Versicherung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Dissertationsschrift selbst verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. I hereby declare, on oath, that I have written the present dissertation by my own and have not used other than the acknowledged resources and aids.

____________________

(Kai Jander)