Herbert Groiss, Michael Dobrovnik

Business Process Management with @enterprise

Groiss Informatics GmbH

Preface

Business Process Management is a very broad and multi-faceted term, so we want to clarify the main focus and the scope of this book. It will cover the basics and concepts of the enactment of business processes by means of business process management systems (BPMSs). We emphasize the technical realization aspects and not the economic or organizational issues.

The presentation of the implementation will be based on the BPMS @ *enterprise* . All examples and screen shots have been created with this system. Comparison with other products is not intended and is conducted rather sparingly.

So it is possible to tackle the topic of the technical implementation of business process management in a concise and focused manner.

Besides the basics, some current trends like Mobile Process Management, Social BPM and Cloud Computing are discussed.

The intended audience of the book includes project managers, process analysts, business process consultants and developers, simply everybody concerned with enactment of business process management.

We will use the following structure: the first chapter explains the terminology, gives a motivation for the deployment of BPMSs and presents a case example. Chapter 2 deals with the modeling aspects of business process management, while the execution of such models via technical business process management systems is discussed in chapter 3. The potential for optimization of business processes via enactment are presented in chapter 4. We conclude with chapter 5 concerned with specifics of the phases of a business process management project.

The BPMS @enterprise can be downloaded from the homepage of Groiss Informatics GmbH *http://www.groiss.com*, or can be tried out on line. A 60-day evaluation license is provided free of charge, thereby giving a hassle free opportunity to practically exercise and implement the techniques presented in this book.

Klagenfurt, June 2014 Herbert Groiss, Michael Dobrovnik

Contents

Chapter 1

Introduction

1.1 Business Process Management

Process management is concerned with the elicitation, documentation, design and optimization of business processes (Wikipedia).

Examples for such business processes are order processing, travel applications and expense administration or billing and invoice activities. More and more, processes are viewed as being in a central focus in a modern business environment. We can distinguish the following aspects:

- ❑ Process capturing: the first step is the analysis of the as-is processes. How are operations carried out, which tasks are executed, who is involved, under which constraints; that are some of the main questions to be answered here. The results are documented in an appropriate manner. The form of this documentation and the notations used are discussed in detail in chapter 2.
- ❑ Process execution: the area of workflow management specifically deals with the execution of the processes by means or with the support of process centered IT systems. Such systems control the concrete execution of process instances, thereby coordinating manual as well as automatic work steps (activities). The documented as-is processes have to be designed in a sufficiently detailed manner to allow for the interpretation by a workflow management system.

- ❑ Process monitoring and analysis: Workflow systems document the execution of each process instance (case), enabling insight into processes. Data about every step is gathered and compacted, thereby allowing to get vital information about process performance and the ability to react with appropriate measures in a timely manner.
- ❑ Process optimization: Data from the analytical component can also be used to adapt the process models themselves. This business process reengineering would lead to a more efficient way of carrying out the business (faster, cheaper, better).

Since the optimization is a kind of re-modelling, the steps can be depicted as a cycle like in fig. 1.1. Repeated and consequent analysis and optimization can lead to continuous improvement.

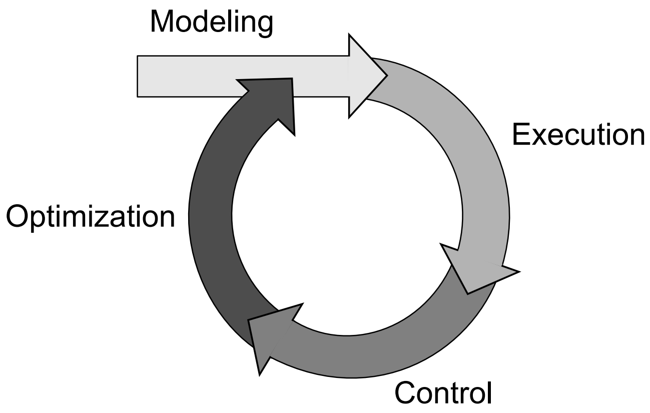


Figure 1.1: Business Process Management Cycle

To present the business advantages of carrying out the tasks of process management, we will now look at the phases in somewhat more detail.

1.1.1 Process Capturing and Modeling

Explicit documentation of processes prescribe how things should be done. Such documentation (a "process handbook") can serve as a guideline for everybody and facilitate the communication between the involved parties over organizational boundaries. Especially for new employees or in the case of new procedures, such documents provide vital orientation.

The idea of a documentation for the ways an organization operates is quite old. An early example are the instructions in the "Regula Benedicti" by Saint Benedict of Nursia, dating from the sixth century (see figure 1.2). One of the main reasons of writing down the operational principles would clearly be that in an international organization with wide spread locations (especially considering contemporary communication round trip times) it is virtually impossible for a 'chief executive' to literally explain the work procedures to each novice.

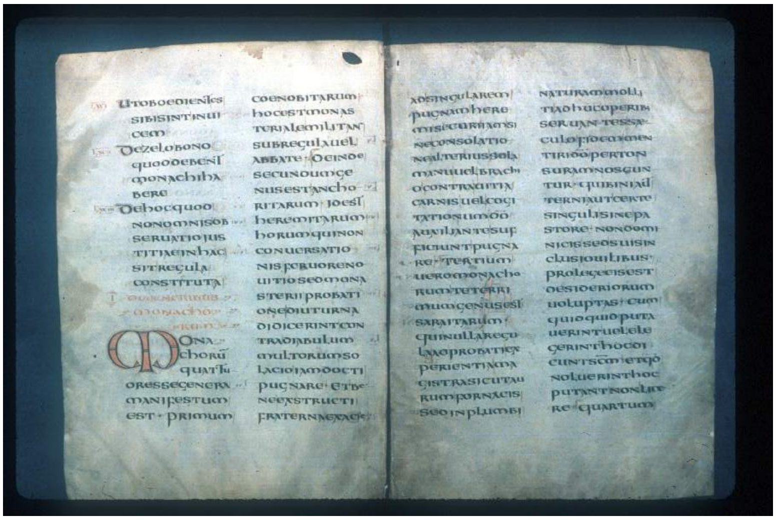


Figure 1.2: Regula Benedicti, early example of process instructions, 8th century copy

To be able to present proper documentation about processes is also a condition for industrial certification efforts like ISO9000 or for accreditation via governmental imposed constraints like e.g. the rules of the US-american Food and Drug Administration (FDA), which governs the pharma and food sectors.

The collection of an organizations documented processes is called its *process handbook*. Chapter 2 will be concerned with state of the art process notation.

1.1.2 Process Execution

The second step of the process management cycle is the execution of the business processes via IT systems. Such systems are known as Business Process Management System (BPMS) or Workflow Management System (WFMS).

Concerning the terminology: Workflow system or workflow management system are the traditional names for IT systems focused on process automation and on "real" execution of processes. Since 1993, the Workflow Management Coalition (*http://www.wfmc.org*), an organization of vendors of such systems, devotes itself to the creation and promotion of industry standards in this area.

The newer term business process management is to be understood in a somewhat broader perspective. It subsumes organizational aspects as well as business economics. Business process management systems can be mere drawing or semantically richer modeling tools, can be execution enactment systems on be analyzing and monitoring functions.

Since the terminology shifted from workflow management to business process management, and the workflow systems try for complete coverage of the business process management cycle, the current accepted general term seems to be business process management system [**?**].

Execution of processes in a BPMS means the system to be in control of the process. When a user decides, that a certain task is completed, she will inform the system. Then the BPMS will select the following task and the following agent according to the process definition.

While this implies that there will be an existing executable process definition, this needs not to be true in every case. Flexible BPMS allow the definition or the enhancement of the process path during execution. Section 3.5 will elaborate on that aspect.

On the other hand, not all defined processes will necessarily be automatically executed by a BPMS. We would like to give three criteria which determine if automated execution makes sense:

- Existence of a process structure: the tasks are performed in a logical or natural sequence by several participants based on division of work. Usually, there is no direct cooperation of several agents in a common location.

  If a certain task has to be performed by a single person, this situation does not necessarily call for a BPMS (exceptions might apply). Tasks that cannot be decomposed in a sensible manner into smaller actions are also not likely to be qualified for BPMS. This holds especially for tasks to be carried out somewhat creatively within a team or in a meeting of several people.

- Need to document process execution: when there are strict traceability requirements (i.e. to be able to determine who did which tasks at what point in time), the consequent usage of an BPMS really makes sense (even when the structure of a process is rather simple or virtually nonexistent). And we would like to emphasize here that transparent ways of work are almost always highly desirable.

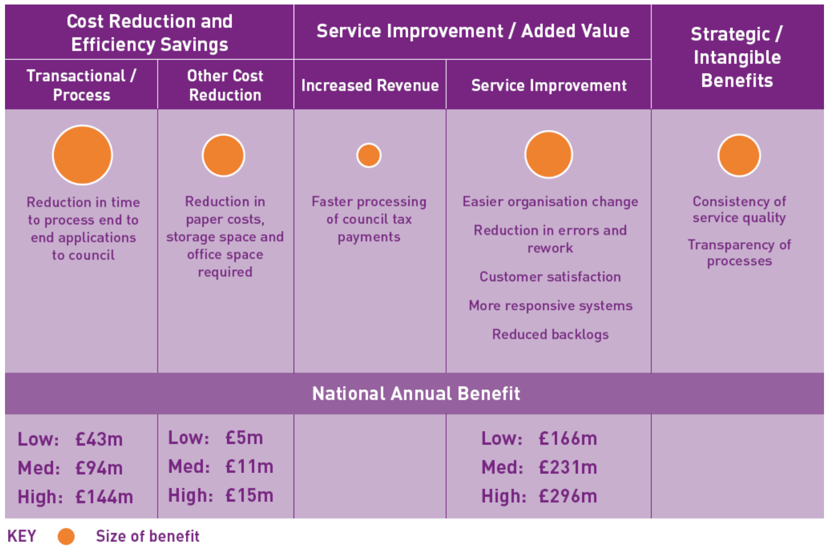
- Automation of some steps: each step of a structured process definition could be carried out manually or could be automatically be executed by an IT system. The BPMS is not responsible for the automated execution itself, but is concerned with data transfer and conversion to and from the other systems, to react upon reception of events from those systems, and of course with the overall coordination aspects.

If some of the three criteria apply, there are clearly potential advantages for deploying a BPMS:

- Quality: Process execution adheres to the process definition and thereby to the prescriptions of the process designers. While some deviations from the defined paths are possible for the sake of flexibility, those are carried out in a controlled manner and are always meticulously documented by the system.
- Swiftness: The process execution speed is increased based on the following factors:
  - no physical transport times (e.g. like paper transport)
  - potential for automation of whole tasks or some part of them
  - possibility for parallel processing
  - feasibility for continuous status reporting, and awareness or control of progress
- Traceability: every task completion action and each change of data is documented by the system. There is immediate potential to provide information about who did what change at what point in time, who signed off what item, etc.
- Optimization potential: The accumulated run time data of the process executions can be analyzed to find potential areas for enhancements. Answers to question like "Which steps do take unreasonable long time?", "Where are the bottlenecks?", "At which points do exceptions occur with a high frequency?".

The ultimate decision about the deployment of a BPMS is obviously a business judgment. Either the projected benefits of the combined effects of the before mentioned aspects is higher than the price tag attached to the introduction of such a system, or the potential loss from non-application is a justification for the implementation. Potential losses may result from non traceable or non repeatable execution paths, from violation of rules and constraints, or from processes being stuck a some arbitrary stage without being aware of them or any possibility to apply appropriate escalation measures.

The literature is a rich source of impressive examples of savings and other positive effects resulting from the deployment of workflow systems, e.g. see [?] or [?].



Cost Reduction and Efficiency Savings		Service Improvement / Added Value		Strategic / Intangible Benefits
Transactional / Process	Other Cost Reduction	Increased Revenue	Service Improvement	
●	●	●	●	●
Reduction in time to process end to end applications to council	Reduction in paper costs, storage space and office space required	Faster processing of council tax payments	Easier organisation change Reduction in errors and rework Customer satisfaction More responsive systems Reduced backlogs	Consistency of service quality Transparency of processes
National Annual Benefit				
Low: £43m Med: £94m High: £144m	Low: £5m Med: £11m High: £15m		Low: £166m Med: £231m High: £296m	

KEY ● Size of benefit

Figure 1.3: Potential of Workflow Management, from [?]

A study [?] undertaken by Capgemini for the UK government has scrutinized six workflow projects in the public administration sector. As can bee seen in figure 1.3, the most impressive effects were minimization of cycle time, but also service quality like flexibility and customer satisfaction could be significantly improved. Process transparency was definitely better. Additional savings in the areas of paper consumption and office space requirements were measurable.

How could someone be reluctant to apply workflow management technology? Surely, the handling of business processes with workflow systems does also bear some potential problems, e.g. elaborated in Wikipedia [**?**]. To summarize, the following arguments are presented there:

1. "Employees loose their individual responsibility and initiative, by strictly adhering just to the prescribed processes. This leads to a lesser degree of motivation and mere work to rule.
2. Increasingly turbulent business environments are a factor for a gap between workflow model and reality. Creativity and ideas for improvement are rather tightly framed by the technology.
3. Diminished or delayed ability to react upon less frequent or unforeseen events."

The first argument is concerned with the loss of control of the individual employees over the process. Alienation and bad motivation like in monotonous assembly-line work can be observed. Simply put, it is a management decision who is in control of a specific process, and to what degree this control is executed. Flexibility could be explicitly designed into the process, so that at least part of the control remains at the level of the individual participants and makes productive use of their experience and abilities.

A crucial issue for acceptance of a BPMS is the ergonomic design and implementation of the user interface. A clearly structured style of presentation of all information needed to carry out the tasks is essential. Along with the the possibility to trace the processes by the employees themselves and the increased transparency, this could outweigh the discontent about a stricter and more structured way of doing things. It is especially important to try to give decision making leeway to the users whenever possible, like e.g. decide on which task to operate next, or to take take influence on further tasks.

The second argument is build around lost agility and flexibility. The effort to implement a process changes in a BPMS might be higher than merely changing some paragraphs in a document. But the impact of the change is much higher and can be applied *instantaneously* throughout the whole organization, thereby achieving a maximum level of agility. Nevertheless, processes with a high rate of change are to be eyed with a healthy portion of awareness. Maybe consequent reliance on system inherent flexibility features could help in such cases.

The third area of concern is the ability to cope with exceptional situations, to allow for deviations from predefined and prescribed execution paths. This again calls for flexibility features in the BPMS as such as will be dealt with in section 3.5.

1.1.3 Monitoring and Optimization

Let us now have a look at the last two areas of the process cycle, the monitoring and the optimization of processes. During process execution with an BPMS, a large amount of data is generated; first for traceability requirements but also for ex-post analysis of processes performance. Compaction and adequate presentation of those data can lead to a level of insight which allows to make informed decisions and appropriate improvement actions for the processes but also at the scope of the whole organization. Effective continuous improvement by holistic process management is feasible on this basis.

So far we used the division of process management in modeling, execution and monitoring and optimization. Diverse variants of categorization and tasks of process management exist in the literature. Some of the classifications use a form of maturity model. We would like to briefly present the Business Process Maturity Model (BPMM) [**?**] of the Object Management Group. This model is based on the well known Capability Maturity Model from the area of software engineering. The BPMM distinguishes five levels of sophistication:

1. Initial: no process management, inconsistent and ad-hoc decisions
2. Managed: the processes are defined locally and in a repeatable manner
3. Standardized: enterprise wide global standardization of processes
4. Predictable: process performance is analyzed and managed
5. Innovating: continuous improvement of processes

A comparison of the previously discussed cycle model and the maturity model reveals some commonalities: the layers *Managed* and *Standardized* imply that the processes are standardized (at different organizational scopes). The most sophisticated layer *Innovating* resembles the optimization phase. The *Predictable* layer is roughly equivalent with the phase execution from the cycle model. BPMN defines targets and postulates prescriptions, but does not deal with implementation issues (like IT-support) at all. A requirement is e.g. ([**?**], page 350):

> Measures of process attributes and performance and quality results emerging from the organization's product and service work are collected on a periodic basis and stored in the organizational measurement repository.

From a pragmatic perspective, such data about process executions are only available in a ready usable form, when the processes are really implemented on the basis on a (technical) BPMS. Both presented models are rather similar to each other when viewed from a step back.

In the following section, we will have a look upon which classes of processes can be typically found in business organizations.

1.2 Process Classification

Several classification approaches for business processes exist (cf. [**?**] for an overview). Commonly, three groups of process types can be distinguished, like depicted in figure 1.4:

- ❏ Core Processes: those are the processes that are instrumental in fulfilling customer needs, which are essential to meet the business targets. They are highly specific within a business sector as well as within a single corporation.
- ❏ Management Processes: those are common processes for corporate management, reorganization and quality assurance
- ❏ Supporting processes: are situated in the areas of finance and controlling, in human resources, etc.

Similar classifications just differentiate between primary processes, which correspond to the category of core process mentioned above and secondary processes which subsume the other two categories, thereby avoiding the somewhat blurred difference between management processes and support processes.

Several process frameworks devote themselves to a single application domain or a business branch, a prominent example being the ITIL framework for processes in IT operations and services [**?**].

To gain an overall insight into all processes of an organization they are collected and structured in process maps, process handbooks or process libraries.

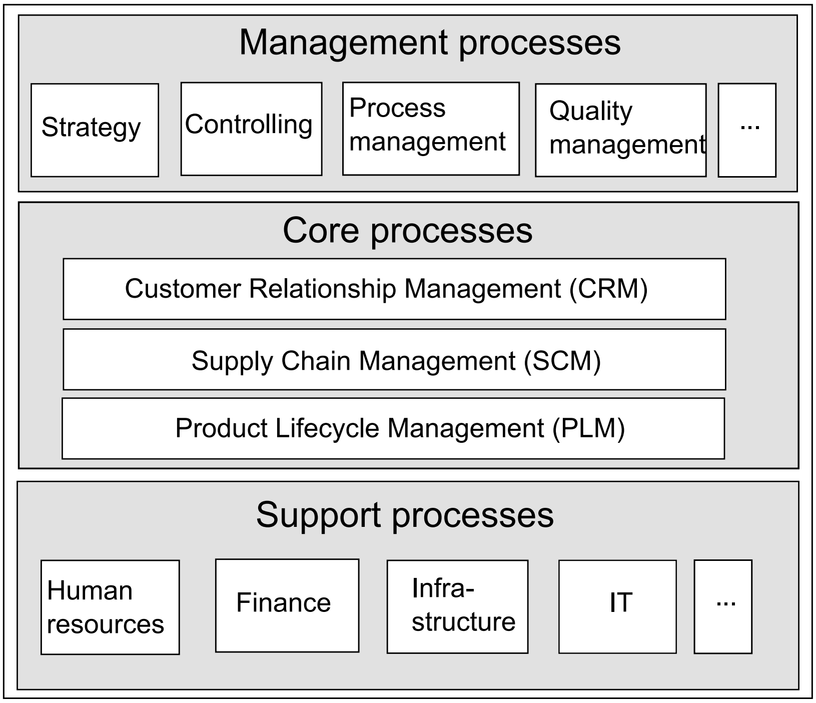


Figure 1.4: Processes in Business Organizations

1.2.1 Core Processes

The core processes are essential to achieve the goals of the organization as a whole. Their importance is captured by the term *primary processes*. In a corporation into manufacturing, one would find processes for production and sales in this category. For a health care organization this would be the processes for diagnostics and treatment of the patients.

A further subdivision of processes can be made along certain types of organizations. The Siemens Reference Process Framework [**?**] enumerates processes for manufacturing sector and distinguished three process groups.

- ❑ Product Lifecycle Management (PLM)
- ❑ Supply Chain Management (SCM)
- ❑ Customer Relationship Management (CRM)

In order to fulfill the ultimate goal of revenue generation by selling the goods, the future sales items would have to be developed first. This is the area of Product Lifecycle Management where one can find the following processes:

- ❑ Planning
- ❑ Construction and Design
- ❑ Prototype and sample development
- ❑ Product approval and accreditation

When there are prototypes or samples, the production for customers could start. Relevant processes are:

- ❑ Order Processing
- ❑ Purchasing
- ❑ Production Planning and Production
- ❑ Shipment

Customer Relationship Management surely is a group of core processes:

- ❑ Customer Acquisition and Prospect Handling
- ❑ Quotes and Bids
- ❑ Marketing Campaigns

To illustrate the width and scope of the area, we will give some examples for core processes for specific economic sectors:

- ❑ Banking: Loan Management, Account creation
- ❑ Insurance: contracting and insurance policy modifications, claims approval and management
- ❑ Public Administration: File Management, travel document application processing, business registration, building permit processing

1.2.2 Supporting Processes

In contrast to the core processes, the supporting and management processes are much more uniformly applicable over all branches. They can be further subdivided into:

- Human Resources: administration of employee centered processes
  - Application processing and selection, Recruitment
  - Grants and revocations of access permissions and rights
  - Vacation administration
  - Business trip planning and accounting
  - Expense processing
  - Sickness leave handling
  - Work time and project time accounting
  - Employment termination administration
- Accounting: Processes to deal with invoices, dunning, payroll processing
- Infrastructure and IT-Processes:
  - Incident Management: Complaint management and classification
  - Problem Management: Problem solving and trouble shooting
  - Configuration management
  - Resource acquisition and maintenance management
  - Supplier management
  - Customer and user support

1.2.3 Management Processes

Examples for management processes are:

- Strategy: Development and regular evaluation of business strategies
- Workforce Planning and Development: Planning the quality and quantity of needed personnel, appraisals and evaluations, education and qualification measures

- Financial Planning and Controlling: planning for financial requirements, solvency assurance, capital and asset management
- Process management: analysis and optimization of business processes, leading and implementation of reengineering efforts
- Risk management: manage risks in the areas of finance, production, environmental issues
- Quality management: Elicitation and prescription of quality criteria, regular evaluation and correction measures, lead of overall quality enhancing efforts e.g. Six Sigma projects

Notably, this group also includes activities concerned with the processes themselves. Planning, elicitation, documentation and design of business processes are a central management task. Those topics have to be dealt with in an open manner ignoring organizational boundaries and must be carried out at all levels of the hierarchy, from management to the single employees in the business and staff departments.

1.3 The Process Centered Organization - A Case Study

The following small case study helps to illustrate, how successful and integrated application of BPM concepts might look like:

A Internet Provider "SuperSurfer" applies BPM principles universally. The core processes as well as the supporting and management processes are implemented. The sound base for this is the deployment of a BPMS which holds all relevant information in a data base.

The corporations core processes are:

- CRM: From the first contact with a prospect until order reception. The CRM module in the BPMS allows the administration of customer data, contact data, appointment management and contact protocols. Quoting, bid processing, and order processing are integrated.
- SCM: The order processing forms an interface to the area of production. According to the order, new processes like "Creation of an Internet

Connection" are started. Those processes consist of interactive and of automatic steps. Interactive steps are carried out by an employee, automatic steps are performed by some program, a third party system or by calling web services in external organizations like domain registrars. Production processes culminate in shipment and installation as well as in producing an invoice.

- Service: Ongoing customer care like inquiry and service request processing, and maintenance operations as well as appropriate customer contacts via sales conclude the cycle and hopefully result in further acquisition actions.

The supporting processes comprise:

- Human Relations: Regular and special vacation applications, flexitime administration, business trip applications and accounting, expense reporting, leave for sickness, general and project specific time accounting, employee selection and hiring.
- Infrastructure and IT: Acquisition of hardware and software, infrastructure maintenance.
- Project management: execution of projects e.g. in the areas of software development, quality assurance and process reengineering.

The spectrum of the processes contains highly structured, high volume processes which are deeply integrated with other in-house and third party IT systems as well as rather simple processes with hardly any structure at all. But all those diverse processes are performed within the controlling and logging capabilities of a state of the art BPMS, which facilitates some crucial advantages. Let us illustrate this via some example scenarios:

- Each employee can access all his tasks electronically, even when being in the road and by means of a smartphone. The tasks can be structured arbitrarily in hierarchical folders providing much needed orientation and focused views.
- Responsibility transparency: Each and every running process is assigned to one participant (parallel processing could lead to a multitude of performers). The ability to provide instantaneously information about the state of each process ("Where is it now?", "Who has it?", "When will it be ready?", ...) is gained.

- Integrated central calendar: The calendar is integrated with the BPMS and allows to present different views (vacations, business trips, leaves for sickness) linked together with arbitrary appointments and dates and deadlines from running processes.
- Central Database: All data relevant for customers and production are in a central database. In trouble-shooting scenarios, service personnel can access customer data, contact data, current orders, previous service cases and so forth. They have current and comprehensive information immediately on hand.
- Controlling: The reporting component can deliver all facts for intra-organizational controlling. Answers to questions like: "What is the load of the employees?", "How many orders were processed in the current period?", "How well did we do in comparison to last month?" can be provided. Flexible ad-hoc reporting is possible and also standardized repeatable reports are provided.
- Agility: When there arises the need to change a process, like e.g. to incorporate an additional checking step or some additional tasks when certain conditions hold, the process definitions get adapted. From this point in time, processes are executed adhering to the new definition. The whole organization can react quickly to changing requirements. Changes in the data structures, like new fields for customer data records are likewise deployed almost effortlessly.

The next chapters will deal with the essential steps to implement such a process oriented organization.

Chapter 2

Process Modeling

Process modeling is the transformation of the information about a process into a model. Data structures, process participants, the details of the individual process steps, and the flow of the data and control of the process are described. To put it bluntly:

> *Who* does *what when with what means*?

We have to define which actors perform which activities with which data using which tools and in what sequence. Before we will explicate the modeling steps, we have to ask ourselves, what the ultimate purpose of this modeling is. In the previous chapter, we presented the process management cycle with modeling being the initial step. Usually, the next step is the execution of the model under the control of a BPM system. But there could be circumstances, where an immediate execution capability is not intended. Reasons for that might be:

❑ the process is not suited for the execution in a BPMS, since it describes actions performed by a single person.

❑ the process is not (yet) described or understood in sufficient level of detail. The description should serve as a guidance for process executions, details will have to be elaborated and formalized later on.

In both cases, a thorough modeling of the process could nevertheless make sense in order to incorporate it into a complete process handbook which captures all business processes.

The different perspective from a modeling point of view is the level of detail needed. An executable process calls for complete and thorough definition of data structures, elaborations of conditions and technicalities of calls to external systems. For documentation purposes alone, such effort is not needed.

Before we delve into the center of process modeling, we will have a look into the organizational environment were such processes are executed. Subsequently we will discuss the process flow as a controlled sequence of activities, and then process data aspects will be treated with.

2.1 Defining the Organizational Structure

In order to be able to define business processes, we have to define the structural and functional elements or the organization, namely the organizational units and the roles within.

2.1.1 Organizational Units

Any organizational entity (corporations, government institutions) of significant size is structured in progressively smaller specialized units; examples are: accounting office, human resources department, production division.

Organizational units can be typed, like in a university, there are schools, departments, research centers, etc. The type of the organizational unit its called its *organizational class*.

Organizational hierarchy

The individual organizational units are usually arranged in a form of hierarchy, thereby forming the organizational tree which can be depicted in an organizational chart. Capturing this structure and to map it into the BPM system is of importance, since the processes often flow between the units and hierarchical layers.

There is a kind of potential conflict concerning the rather static boundaries implied by the organizational structure and the desire to implement corporate wide process flows. Where should the responsibility for process execution be placed? A viable compromise is the introduction of a *matrix organization*, where the hierarchical structure is enhanced and overlayed by virtual units stemming from process definitions. The pure (or extremal) form would be a *process structured organization*, where the original organizational structure is substituted by the business processes. This topic is discussed e.g. in [**?**].

2.1.2 People

For the mere modeling purposes, it is not necessary to deal with each and every person who will be participating in the process. We strive for the processes to be independent from concrete persons, we will use *roles* to provide a suitable abstraction concept.

At run time, for execution purposes, the persons that shall use the BPMS must be known to the system. Certain attributes, like name and unique id as well as other properties like e-mail address, phone number, street address are stored.

2.1.3 Roles

Each type of organization will have different bundles of functional tasks or capabilities which are assigned to members of this organization like secretary, head of department, software developer or spanish-speaking.

Such bundles are called roles and might also subsume or imply a set of rights. Consider the role "system administration" which will usually allow to carry out some privileged actions.

Besides the roles already existing in an organization, there are roles that must be defined specially for a process or for an application. Such process specific roles form a virtual group of persons, which potentially carry out certain tasks in a process. There might be a role "Approver" designating all persons that may act as approving instance in a particular process.

Roles can be differentiated along their relationship to the organizational structure. There are roles which apply within an organizational department, e.g. someone being a "Research Director" of a specific lab. Such kind of roles are said to be *local* roles.

Other roles might be significant within a particular department as well as all subordinate units below it in the hierarchy. Being the "Dean" of a school also implies being "Dean" of all departments within the school. Such roles with a deep scope are called *hierarchical* roles.

A third form of roles is applicable without any organizational context: "spanish-speaking" is not dependent on ones position. Such roles are said to be *global* roles.

When assigning roles to users, for local and for hierarchical roles an organizational unit must be stated to provide the proper scope. Assignments of global roles do not require this.

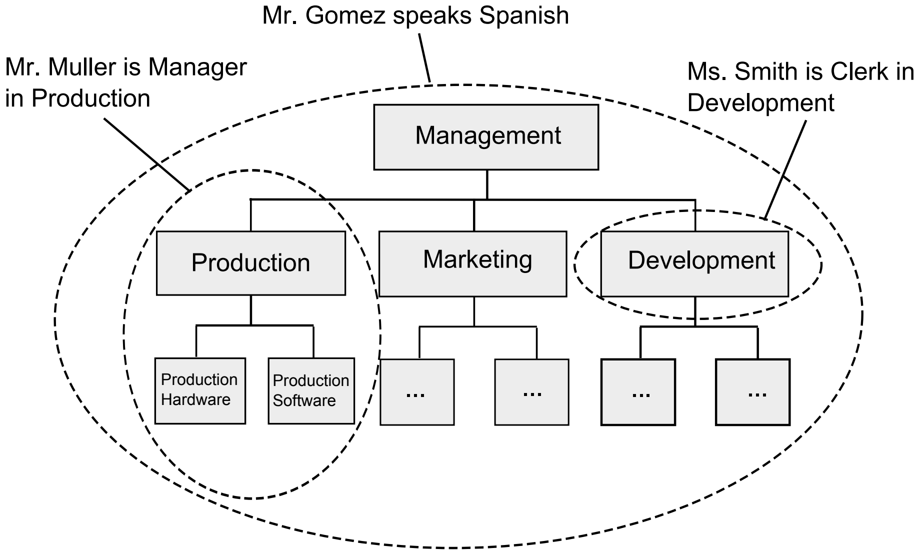


Figure 2.1: Organization and Roles

Figure 2.1 depicts the scope of the different role assignments. The role "Spanish-speaking" is a global role, therefore the scope of the role assignment is the whole organization tree. Role "Clerk" is a local role, a role assignment for this role requires an organizational unit and is only valid in this particular unit. The "Manager" role is a hierarchical one, role assignments are valid in the assigned organizational unit and in all subordinate organizational units as well.

A straightforward schema to define organizational structures is presented in figure 2.2. One can find the entity classes "User", "Role" and "Organizational unit". The class "Role assignment" defines the relationship between those three entity classes. A recursive 1:n relationship on "organizational unit" represents the organizational hierarchy.

Users can be assigned different roles in different organizational units, but there

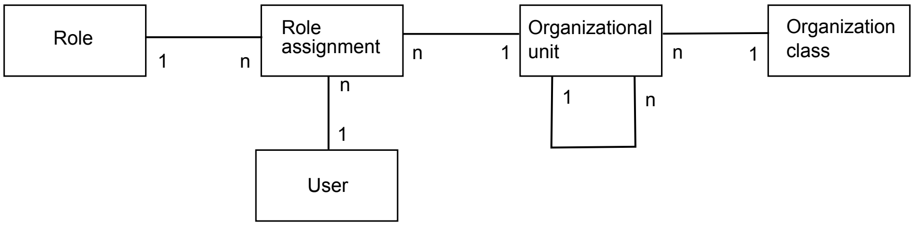


Figure 2.2: Schema of Organizational Data

are some constraints involved:

- ❑ Some roles can only be assigned once per organizational unit. This applies for roles which capture the aspects of a function or position (e.g. there can only be one "Head of Department" per department).
- ❑ A user can be assigned a particular role just once. This is obviously the case for global roles, and also often applies to functions or positions (one person cannot be "Head of Department" for several departments).

Since a particular user can be assigned roles in different departments, the modeling of matrix organizations is possible using this scheme. Besides the hierarchical organizational structure, there can be further "virtual" organizational units which subsume users from different "real" departments. There could be a joint "Project X" organizational unit which includes employees from marketing, from sales, from engineering, etc.

Two further aspect of organizational modeling, namely substitutions and authorizations will be dealt with in chapter 3 "Process execution".

2.2 Modeling the Flow

We would like to clarify some terms before we will turn towards the topic of process modeling:

- ❑ Process definition: this is the mapping of a business process to a model (e.g. "business trip administration").

- Process instance: this is a single case, a concrete instance of a process definition (e.g. the business trip of Mr. Smith to Washingtion from the Oct. 2nd, 2013).
- Activity: is a step of a process definition, can be elementary or can be structured in itself (being a subprocess definition, e.g. application procedure for trip funding).
- Task: is an elementary step in the process (e.g. approval of trip funding).
- Application: a set of topically related process definitions (e.g. "Personnel processes"

The terms are closely related to the Glossary of the Workflow Management Coalition [**?**].

2.2.1 Selection of a Process Modeling Language

The first consideration in process modeling will clearly be the selection of an appropriate modeling formalism and a corresponding notation. Since the field has a certain history and maturity, there are many viable candidates originating from diverse sources. We will mention just some of them:

- Formal Notations: The most prominent representative in the area or formal methods are Petri-Nets, a particular modeling language building on them would e.g. be YAWL [**?**].
- Notations emerging from products: ARIS, a tool (of Software AG) for process modeling being in widespread use, employs the "Event-driven process chains (EPC)".
- Standards: The widely known modeling method UML originating from software engineering, also allows to model processes via the UML "Activity Diagrams". In the domain of business processes, the UML method is being used less and less.

  The Business Process Model and Notation (BPMN) [1] [**?**] of the Object Management Group (OMG) has prevailed as *the* standard and meanwhile most of the products support this notation (or an extended subset of it).

[1]BPMN originally was the acronym for Business Process Model*ing* Notation, since version 1.2 more emphasis is put on the model, and the acronym now means Business Process Model *and* Notation.

> The Business Process Execution Language (BPEL) [?] is a language to specify the interplay of processes in which there are hardly any interactive steps at all. Main focus is the "orchestration" of automatic processes. While this notation has been promoted by large vendors for some time, it looses foothold against BPMN, especially since BPMN got an execution semantics in version 2.0.
>
> For some time, an extension of pure BPEL to also include interactive steps has been in the works (BPEL4People [?]).

All those methods (with BPEL being an exception here) rely on a graphical notation, the resulting process model being a picture, which are supposed to facilitate communication and understanding of the model with rather limited effort.

The main drawback of a comprehensible graphical notation is the omission of details which would be needed for implementing the processes in a BPMS. Additional textual annotations and details are needed for real technical enactment. This is an inherent dilemma in process modeling, as it serves two purposes: on one hand to be a representation of the process that is easily graspable by all (human) participants, on the other hand to be an exact and detailed base for the execution of the processes in an IT system.

A second problem of the graphical modeling methods is, that the notation is not really restrictive at all, one can "draw" quite a lot of absurdities: unstructured, never terminating processes, or processes being stuck under certain conditions e.g. because of omission of some edges and flows between steps. Sadly, this often is recognized just when the process should be deployed in a BPMS. The following citation taken from the definition of BPMN ([?], page 439)is revealing here:

> Not all BPMN orchestration Processes can be mapped to WS-BPEL in a straight-forward way. That is because BPMN allows the modeler to draw almost arbitrary graphs to model control flow, whereas in WS-BPEL, there are certain restrictions such as control-flow being either block-structured or not containing cycles.

In the light of those reasons, we will take a slightly different approach here: the description of a process corresponds to the description of an algorithm. In the area of precise description of algorithms, programming languages were introduced in the field. There are many different programming language families founded on diverse paradigms (procedural, object oriented, functional and logical being some examples). For the purposes of defining business processes, the structured paradigm seems to be the most promising one. Graphical representation of control structures is straightforward and can be conveyed with minimal effort.

But to formulate a business process (just) in a mere programming language, is not really appropriate. The modeling should be manageable also for persons without any special programming skills or groups not interested in technical details. As solution we represent the control structures of the procedural languages in a graphical manner like in the area of Visual Programming [**?**].

The approach used here is the definition of a procedural script language for business processes along with a corresponding graphical representation using the elements of BPMN [**?**]. The pure textual notation will be called Workflow Definition Language (WDL).

All needed control structures like sequential execution, alternative execution, repeated execution, parallel execution and event handling are representable in BPMN, as well as the definition of elementary activities. We will not use all BPMN constructs, but deliberately employ just a reasonable subset in a more constrained structure. The consequence is, that any modeled process is a correct BPMN process, but not every BPMN process complies to the structural restrictions imposed by us.

We will now introduce the BPMN notation along the language elements of WDL. The most basic elements of the graphic process model are nodes and edges. Nodes can be activities, events and gateways. For edges, we will singly use the control edge of the BPMN notation.

2.2.2 Activities

Let us start with a look at the different types of elementary activities:

Manual activities

are referred to as "Tasks" or interactive activities. They are executed by a person (usually abstracted and represented by members of roles) and are not divided or structured for process modeling purposes. Each task at least has a unique *id* and a *name*. The name does not need to be unique, the same name can be used in different locations within the process definition or in other process definitions.

Figure 2.3 shows the graphical representation. The box is marked with a person icon in the upper left corner. The inscribed text states the agent of the task (a role-id) and the tasks name.

```
Manager Approve();
```

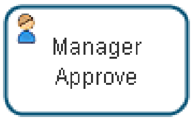


Figure 2.3: Manual Activity

In WDL notation a task is described as follows:

```
agentid taskid "(" [ form {"," form } ] ")" [ stepname ] ";"
```

The id of the agent is followed by the id of the task. The process data used in the step can optionally be designated as forms within the round brackets. The details of the specification of agents and process data will be dealt later on. For now it suffices to assume that an agent id is just a unique name of a role, and that the process data structures are empty.

System activities

are executed by a program. We differentiate between synchronous and asynchronous evaluation:

(a) synchronous: during process execution, some calculations or transformations are made directly or programs are called immediately. There is no need to wait for external third party systems. Figure 2.4 depicts the representation in BPMN and WDL.

```
system send.result()
      "Send results";
```

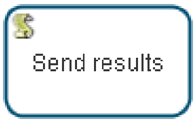


Figure 2.4: System activity

The somewhat subtle difference from the notation of a manual activity is the parchment scroll in the upper left corner. In WDL, after the keyword `system` the name and argument list needed to call a (Java-)method are needed:

```
"system" methodname([ arg {"," arg } ] ")" [ stepname ] ";"
```

(b) Asynchronous execution: a third party system will be contacted, the execution of the process will commence when the communication with this system has been completed. The graphical notation shows cog wheels, in WDL the keyword `batch` is used, followed by the name of a class which implements the details of the communication, cf. figure2.5.

```
batch demo.batch.Main
  "Process in external system";
```

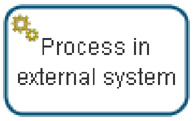


Figure 2.5: Batch Step

Subprocesses

Subprocesses can be used to break down larger processes into smaller, more manageable pieces. A possibly complex, interrelated part of the process is graphically removed from it. In the process, just the call to the subprocess remains, we abstract from the details of the subprocess here, the exact definition of the structure can then be found elsewhere. Besides hierarchical decomposition to achieve an uncluttered big picture, subprocesses are a means of reuse; recurring process parts just need to be defined once and can be referred to multiple times.

In WDL, the keyword `call` designates a subprocess:

```
"call" processname ([ arg {"," arg } ] ")" [ stepname ] ";"
```

The BPMN rectangle of a subprocess has a boxed "+" in the lower middle area, thereby symbolizing the hidden structure of the element.

```
call Approval();
```

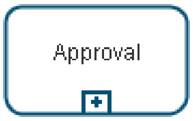


Figure 2.6: Subprocess

2.2.3 Control Structures

A possible course of execution for a process is denoted by composing single activities via special control structures. In BPMN, where are special nodes for control structures, the edges between the nodes represent the control flow.

Figure 2.7 shows a simple, but complete process. Each process begins with a start node and finishes with an end node. Graphically, the end node can be distinguished as being the one with a thicker border.

```
begin
 Role1 Task1();
 Role2 Task2();
end;
```

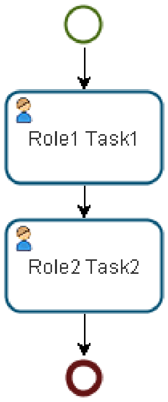


Figure 2.7: Sequence

Directed edges connect the nodes and define the control flow. The minimal, empty process consists of a start node, an end node and an edge leading from the start node to the end node. Further constructs can be inserted into this process by obeying to the following principle: replace an edge by a complete construct and by two edges, one edge leading to the construct and one edge leading from the construct to the original endpoints of the detached edge. This transformation rule is illustrated in figure 2.26, all such rules together constitute a graph grammar.

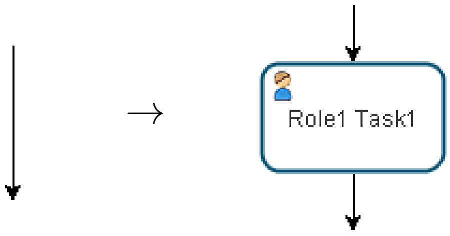


Figure 2.8: Transformation Rule of Graph Grammar

Each of the control structures we will be introducing in the following can be incorporated into the process graph in this manner. A process graph is valid in our notation when it can be constructed by repeated application of the transformation rules starting with the initial primitive graph. An immediate consequence of this approach is, that each graphical model can be translated to the scripted form (the WDL). Both notations are therefore semantically equivalent (the graphical notation is naturally richer concerning the position and sizes of the elements). The complete definition of the graph grammar will be given in section 2.2.11. The implementation of the graphical process editor of @enterprise follows the the rules imposed by this grammar, no syntactically incorrect processes can be created with the editor.

We will now discuss the control structures.

2.2.4 Sequence

A sequence is the succession of two or more steps, one after the other. In WDL, the activities are simply written in this textual order. In the graphical notation the activities are connected by a directed edge (an arrow) leading from the previous to the next step.

In figure 2.7, a complete process has been depicted, beginning with a start node, followed by a sequence of two activities, which will be executed consecutively, succeeded by an end node. In WDL, the start and end nodes are represented by the keywords `begin` and `end`.

2.2.5 Alternatives

There is a selection of *one* concrete execution path from two or more possible execution paths. We distinguish between a system imposed selection versus a selection carried out manually. The first form is called an *If*, the second one is termed *Choice*, thereby indicating the involvement of human judgment.

If

The selection of the execution path is accomplished by the system through evaluation of a predefined condition. The WDL notation corresponds to the one known from structured programming languages. In the graphical notation, there is a diamond shaped condition node annotated with the condition text. Two edges lead from this node. One edge is followed when the condition is true, the second edge (the one with the little diagonal line) is the one that is followed when the condition evaluates to false. In BPMN notation, such lines denote default edges.

In a colored model, the first edge (condition true) will be drawn in green, and the second one will be drawn in red. Both paths from those two edges converge in a single diamond box.

```
if (f.x = 1) then
  Role1 Task1();
else
  Role2 Task2();
end;
```

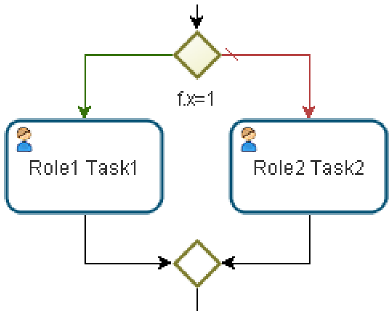


Figure 2.9: if construct

In general, a node has either one black edge (or maybe several of them) originating in it, *or* a green *and* a red edge leading from it. In the second case, the node is a condition node and depending on the outcome of the evaluation of the condition, one of those two edges is activated.

Multiple selections can be modeled with the *elsif* construct, in the graphical representation all the different paths will lead to one single node cf. figure 2.10.

For a technically sound and complete specification of the selection condition, we need a formalism which allows the precise denotation of boolean expressions. We will use a simple expression syntax or XPath, but will defer the presentation of this aspect until chapter 2.5.

Choice

The second form for selecting one of a set of alternative paths is the manual choice by the human agent of the task at the process instance run time. The current agent of a task determines, at which of the successor paths the process instance execution will continue. There are several successors. Each of the successor paths can be optionally annotated with a condition. The user can at run time choose one from those paths without a condition and from those paths where the condition evaluates to true.

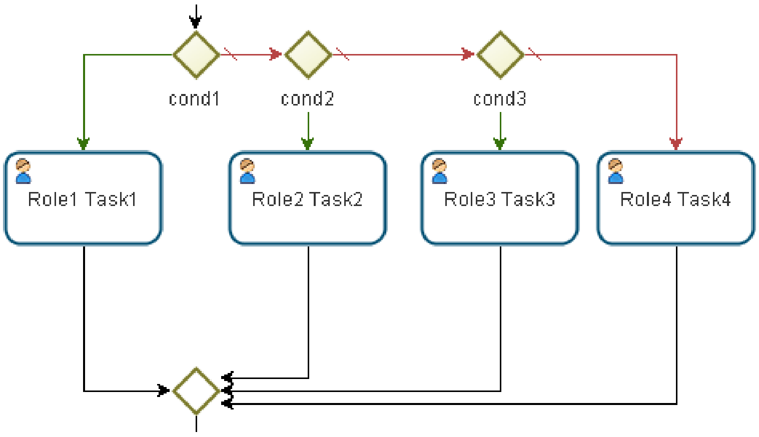


Figure 2.10: if .. elsif .. else

In the graphical notation, cf. figure 2.11, the BPMN intermediate event node is used (a double circle). The starting node of the choice construct contains a pentagon, symbolizing that one of many events will be waited for (the events being the manual choosing by the agent). Each of the paths starts with a node that contains a triangle, which means to wait for some event. Below each of those nodes, there are the conditions and an explanatory text. All the paths of this construct converge in a single diamond shaped node.

```
choice
 "order directly",
  f.amount < 500:
  Assistant Order(f);
 "check again":
   Manager Check(f);
 "Don't order":
  system test.fileIt()
    "Fie";
end;
```

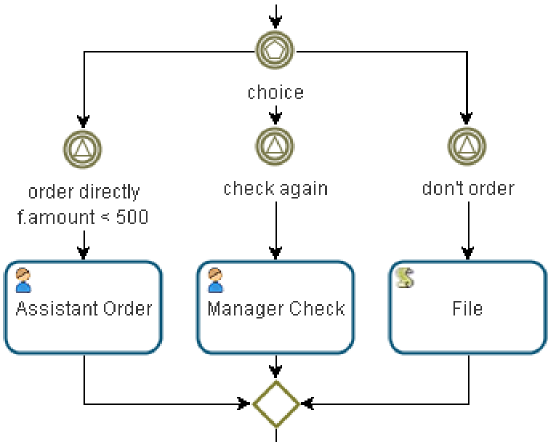


Figure 2.11: Choice

At run time there may arise two special cases:

❑ None of the choice paths is selectable, since all the conditions evaluate to *false*. Then the system will signal an error when the agents tries to finish the task leading to the choice.

❑ Just one of the paths is selectable, then there will be no selection mask presented to the user, and the successor step is started immediately, without any further user involvement.

2.2.6 Loops

Repeated executions of process paths are expressed in the model via loops. The actions within the loop (the loop body) are executed again and again as long or until a condition holds. There are several kinds of loops cf. figure 2.12.

In a *While* loop the condition is checked before the loop body is executed even once. The loop body is executed when the condition is *true*. From the end of the loop body an unconditional edge leads to the initial condition node of the loop construct, there is no special end node in while loops.

In a *Repeat* loop, the content of the loop body is executed at least once, the diamond shaped start node of a *repeat* unconditionally leads to the loop body. After the loop body, a condition node is responsible for checking the condition and thereby for deciding if the loop body should be executed again. If the condition is *false*, the loop body is executed again, else, the loop execution leaves the loop.

There is a third loop variant, depicted in figure 2.13, where the condition is neither checked before the loop body nor after it but rather at an arbitrary point within the loop. Graphically, it resembles the repeat loop, the difference being that the path from the condition checking node to the start of the loop is not a simple edge, but can be a whole path of constructs. In WDL, we denote the condition checking note by `exit when`.

2.2.7 Parallelism

Using parallelism, the execution path of a process can be split up in several independent and structurally different execution paths, each of which progresses at its own speed. All of the splitted paths converge in one join node. Again, there are several variants:

```
while (f.x = 0) do
  Role1 Task1();
  Role2 Task2();
end;
```

```
repeat
  Role3 Task3();
until (f.x = 0);
```

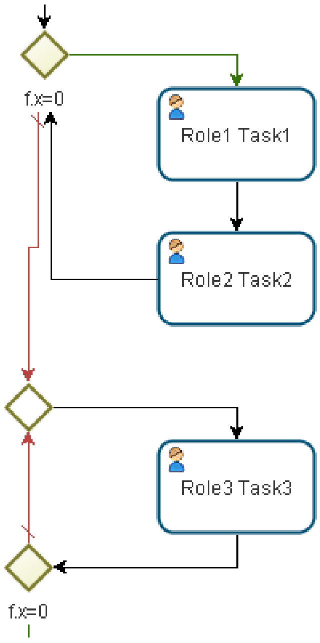


Figure 2.12: While Loops and Repeat Loops

```
loop
  Role1 Task1();
  exit when (f.finished = 1);
  Role2 Task2();
end;
```

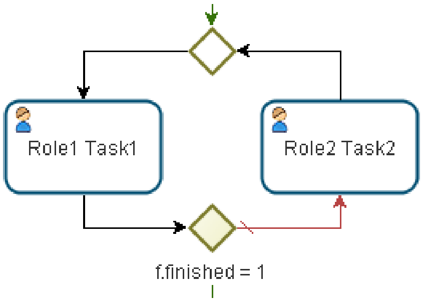


Figure 2.13: General Loop

1. The execution after the parallel construct commences when *all* paths are finished. This is called an *Andpar*.

2. The execution commences, when the *first* path is completed, this variant is termed to be an *Orpar*.

3. The execution commences, when m out of n paths are finished($1 \leq m \leq n$). n is the number of parallel branches, m is defined by the process modeler.

4. The execution commences, when m out of n paths are finished, with m being determined at run time.

In the last three cases, there has to be a an arrangement what should be done with activities within the independent execution paths which have not yet been finished, when the parallel construct finished. There are two possible courses of action, namely terminating the branches or letting them continue in their execution.

Figure 2.14 shows syntax and graphical representation of the *Andpar* construct, in figure 2.15 the *Orpar* is illustrated. The first node of a parallel construct is denoted via a diamond shape with an inscribed +. Unconditional (black) edges lead from it, indicating that all paths originating at this node are put into execution (at the same point in time). The diamond node with the inscribed * is the end node of the construct, all paths originating at the + node will arrive here. The end node is annotated with the condition for termination of the parallel construct. In the WDL, the different branches are separated via a vertical bar $|$.

```
andpar
    Role1 Task1();
 |  Role2 Task2();
end;
```

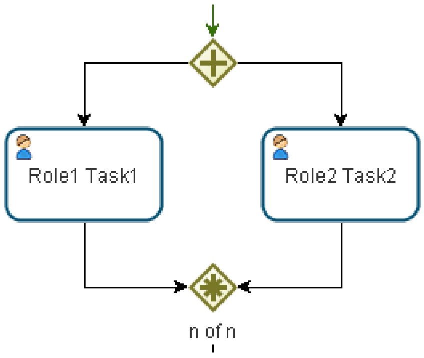


Figure 2.14: Andpar

The cases three and four are being modeled like an *Andpar*, the end node with the * will be annotated with a method or a description that is responsible to determine if the whole parallel construct is finished. This method will be executed every time one of the parallel branches is finished (when its execution arrived at the end node).

```
orpar
    Role1 Task1();
 |  Role2 Task2();
end;
```

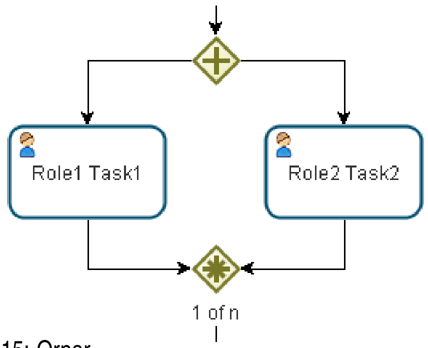


Figure 2.15: Orpar

The forth case - when the parallel termination condition is dynamically determined at run time - is the most relevant in practice, illustrated by the following two examples:

- An application is routed in parallel to three people responsible for decision. Unanimous vote is needed. Whenever a single negative decision is made, the outcome is determined, all other paths can be terminated.
- When more than the half of the involved agents is in favor or opposed to a decision, the outcome is determined, the process can commence after the parallel construct (vote by single majority).

In the graphical notation, the (arbitrary complex) condition is not shown but should be represented via a correspondingly meaningful annotation or node name.

Parfor

The *Parfor* construct is a further variant of parallel execution. Its applicable for the parallel execution of of several paths which are structurally identical. The number of those branches is known solely at run time.

There are two possible sources for this number of branches: 1)it is explicitly modeled in the data structures, or 2) it is determined dynamically via a function call.

The *Parfor* depicted in figure 2.16 first determines the number of data elements (e.g. number of lines in a table of a form). Then a corresponding number of

```
parallel for review in mainform.1 do
   review.agent Task1(review);
end;
```

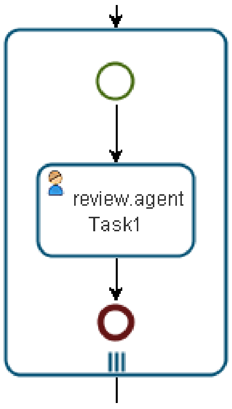


Figure 2.16: Parfor

uniformly structured branches is instantiated and activated. In each of the branches, one of the data elements (one *review* line) is worked on.

Graphically, the *Parfor* is represented as a large container node which envelopes the constructs of the parallel path. The parallel execution is further indicated via the three parallel lines in the lower middle border.

Process execution after a *Parfor* will commence, when all instantiated branches are finished. Additionally, a function (like in the forth case of parallelism) can be used to dynamically decide about process continuation.

Branch

A further form of parallelism is the *Branch*. From the main process flow, an independent execution flow is forked. This branched flow is not synchronized in any way with the originating process, the branch even continues to run when the process instance terminates. This control structure is quite useful for tasks of lesser importance like e.g. pure informational activities where a person is merely informed about a process state or progress, without technically acting on the process.

```
branch
   Role1 Task1();
end;
```

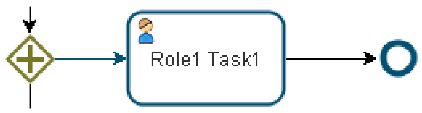


Figure 2.17: Branch

The graphical notation starts with a diamond shaped gateway node (like in the case of *Andpar* and *Orpar*). From this node, two edges sprout. The normal execution flow of the process is a black edge, the flow into the branch is drawn in blue. Additionally the end node of the branch is also inked in blue.

2.2.8 Synchronization with Events

When there are several parallel execution paths in a process, they are running independently from each other until the parallel construct ends. But often, some kind of coordination within the parallel paths is required; a way of synchronization is needed to assure that a certain step in one branch is not to be started before another step in a different branch has been finished.

In order to be able to model this kind of dependencies, we introduce the constructs *RaiseEvent* and *Sync*.

RaiseEvent is used to generate ("raise") an event, *Sync* can be used to wait for such an event. Figure 2.18 shows an example adopting those two elements: in a parallelism, activity `task4` should be started not before the parallel activity `task1` has been finished.

The *RaiseEvent* statement has the following parameters: the first one defines an event-id (`e1`, which must be correspondingly stated at the *Sync* node. The second argument is rather technical and refers to the transaction environment in which the event handlers are executed (`current_tx` is the current transaction). The third arguments provides a context, either the keyword `process` for the current process instance, or a designation of a data element (a form or a form field).

The *Sync* node must state the event-id, an event-handler and the context. The event handler is a set of functions which allows to properly react upon reception of an event via a program.

In the graphical representation, a BPMN event node with a blackened triangle is used for the RaiseEvent construct, the *Sync* is depicted as event node with an outlined triangle.

This completes the definition of the basic control structures. In the following, we will discuss some interesting combination of those elements in more complex situations.

```
andpar
 role1 task1();
 raiseEvent(e1,
   current_tx, process)
   "task1 completed";
 role2 task2();
|
 role3 task3();
 sync(e1,
  com.groiss.event.EventHandler,
  process)
          "wait for finished task1";
 role4 task4();
end;
```

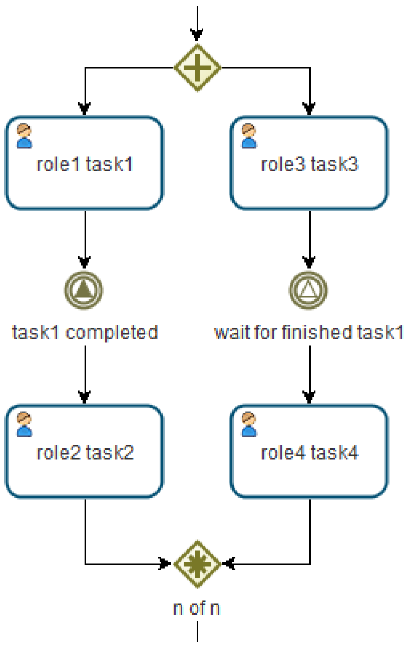


Figure 2.18: Synchronization

Parallelism within loops

When parallelism is used within a loop construct, and when the parallelism is finished when not all of the parallel branches are finished, and the remaining branches will not be aborted prematurely, then the following phenomenon like illustrated in figure 2.19 arises.

There we have a m of n parallelism within a general loop. Assume, that $m = 1$, and *task1* terminates. Then the whole parallelism terminates (and *task2* is still active). When the following condition evaluates to false, *task3* is executed. After this task is completed, a new instance of the parallelism is created, thereby instantiating one *task1* and one *task2* activity. What we would get at this point are two instances of *task2*, one originating from the previous loop execution and one stemming from the current loop.

This situation can be quite confusing for the users, therefore we choose to apply the event mechanism for the synchronization of the loop executions in the following manner: when all branches of an parallelism are finished,

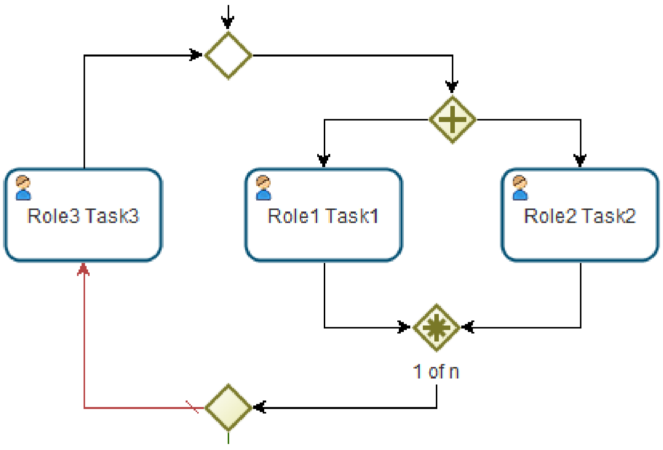


Figure 2.19: Parallelism within a Loop

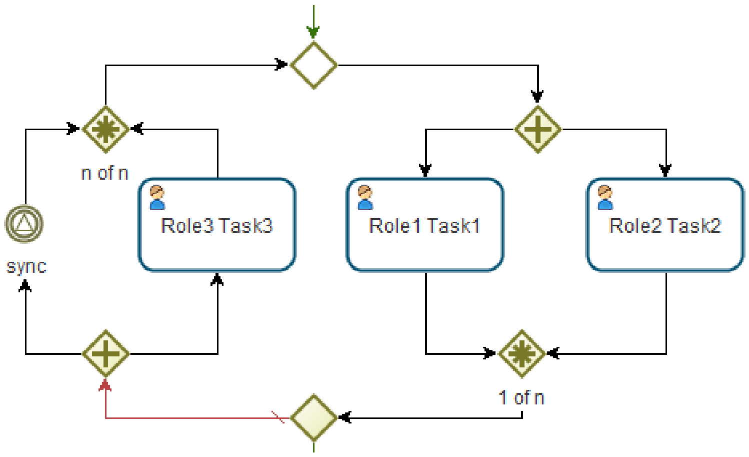


Figure 2.20: Parallelism within a Loop with Synchronization

the workflow engine will generate automatic events for *Andpar* and *Orpar* constructs. These events can be uses to synchronize upon before the next loop cycle starts. Figure 2.20 shows the implementation. The *Sync* node waits for completion of all the branches of the parallelism (*task1* and *task2*). The *Sync* node itself must be placed in a parallelism to *task3*, so no signal gets lost during the execution of *task3*.

The same situation can also arise in the context of the *Parfor* construct, it is handled in a similar manner.

2.2.9 Workflow Patterns

Starting on the foundation of the elementary control constructs, similar execution paths can be observed over and over again in business process modeling. Those similarities were first treated by the group of van der Aalst in the seminal article "Workflow Patterns" [**?**] in a systematic way. Starting with 20 such process patterns, they extended the pattern catalog to 43 items [**?**]. Those patterns are well suited to evaluate languages for process definitions and to be the base of a comparison framework for such languages.

The patterns were soundly presented as Petri nets. The main difference between this formalism and the presentation so far is that there are no explicit control structures like *If*, *Andpar* or similar constructs at all. Instead, Petri nets contain just the most elementary concepts of process graphs: single nodes and conditional edges. Several of the patterns are not directly mappable to the notation used here, often two patterns correspond to one control structure. E.g. the *If* construct is mapped to pattern nr. 4 *exclusive choice*, which corresponds to the *If* node and also to pattern nr. 5 *Simple Merge*, which corresponds to the end node finishing the *If* construct.

In a previous work [**?**], we have shown how those patterns can be implemented via the discussed notation and concepts. We will therefore abstain from an exhaustive presentation of the whole pattern catalog, but will exemplarily choose four of them and present their implementation within our modeling formalism.

Persistent Trigger

Two process paths should be synchronized via events. The event should be stored persistently, until the process gets to the synchronization point. This pattern can be modeled without support for persistent triggers, in fact it is a special case of the situation already depicted in figure 2.20.

Event storage is modeled by positioning the *Sync* node in such a place of the process, where the event can be received and handled. A process instance would run in parallel to the *Sync* node; the parallelism will end where the *Sync* would be placed when persistent events would have been used.

Multi-Choice

This is a parallelism with *n* branches, where just *m* of them get instanciated. This can be modeled via a combination of *Andpar* and *Ifs*. An example with two branches can be found in figure 2.21.

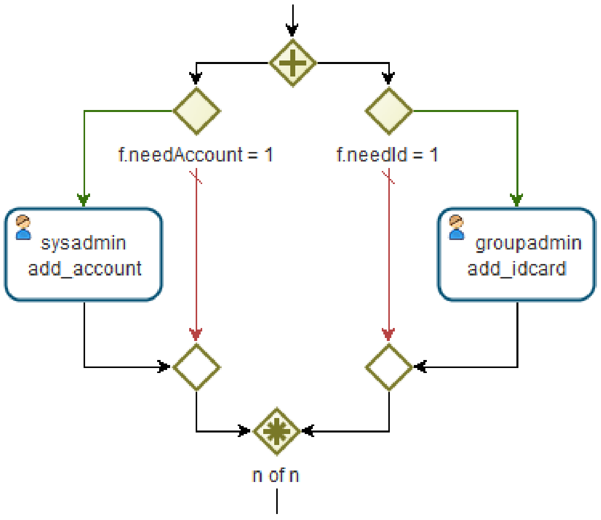


Figure 2.21: Multi-Choice

Multiple Instances without a Priori Run-Time Knowledge

Additional branches should be added to a *Parfor* construct at run time. In principle, this can be accomplished like presented in figure 2.22. In the left path of the parallelism, additional *Parfor* branches can be added by insertion of new *review* subforms followed by calling the function *addParforSteps*. The model does not explicitly show this, but when the modeling of data, function and access permissions will be done, one could see that the *addReviews* task lives up to its name, that is to allow for instantiation of additional subforms and *Parfor* branches.

A further interesting variant of this solution is, that the addition of *Parfor* branches should only be possible as long at least one of the existing *Parfor* branches is still being executed. This could be modeled with an additional synchronization, depicted in figure 2.23 for the BPMN and in figure 2.24 for the corresponding WDL.

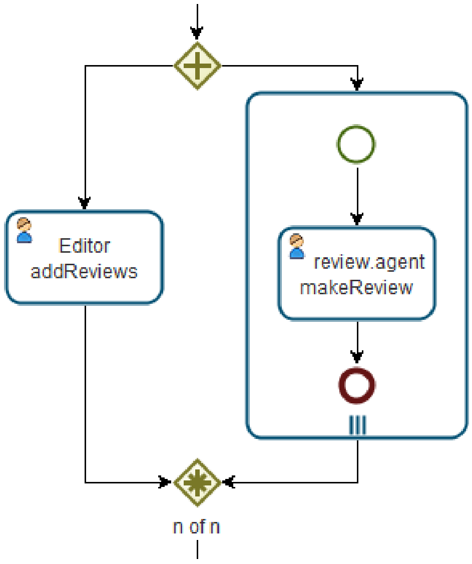


Figure 2.22: Parfor with additional Branches at Run Time

As long as the left branch (with the *addReviews* task) is still active, the branches in the Parfor (and consequently the whole *Parfor*) will be waiting for the completion of this task. As long as the task is active, additional reviews could be attached and treated in new *Parfor* branches. When the *addReviews* task completes, the event is triggered, all waiting *Parfor* branches will complete, too.

The *Sync* node is conditionally executed within an *If*, thereby avoiding dead ends when the left *addReviews* task has already been completed even when there are still active *makeReview* tasks (since events are not persisted in our environment).

Recursion

With subprocesses and calls to them, recursive processes can be defined. Example: for a loan application process, the approval is modeled as a distinct subprocess, which consists of the steps "Approve", "Credit assessment" and "involve superior". The last step is optional and is again a call to the subprocess itself, since the supervisor should again execute the process or be able to involve her supervisor, as long as there is one.

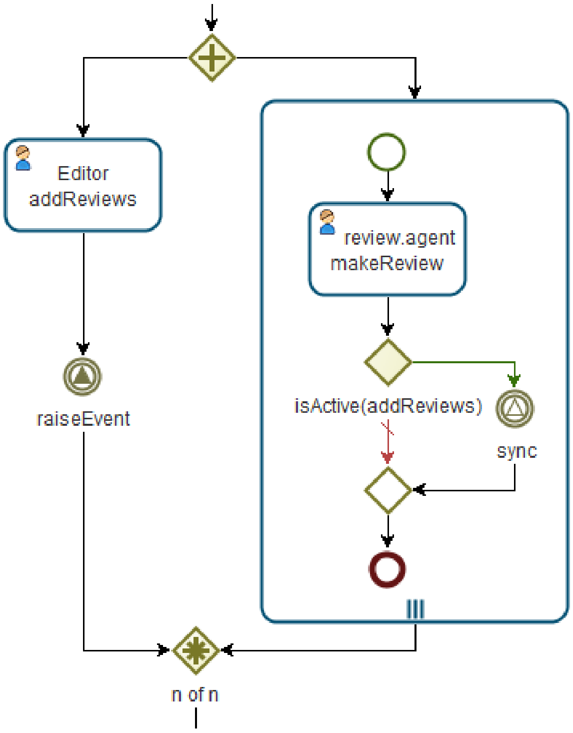


Figure 2.23: Parfor with additional Branches at Run Time and Synchronization

2.2.10 Power and Completeness

As a conclusion for the central part of process modeling, we would like to discuss the chosen approach in terms of of power and completeness. Since just a part of the manifold of language constructs of BPMN is being used and supported, there is the question if we can model every conceivable process on this compact base. The answer is a clear *Yes*, every potential program and therefore every possible process can be expressed with the set of constructs introduced her (the approach is Turing-complete). There remains the issue of how comfortable and how concise the expression gets, since we require the usage of structured constructs ("overhead"). In the area of programming languages, this discussion has been going on in the 1960s (cf. e.g. Edsger W. Dijkstra, "Go To Statement Considered Harmful", [**?**]). There, a broad consensus is established, that the additional effort by the systematic use of structured

```
andpar
  Editor addReviews(f);
  raiseEvent(finishParfor, current_tx, f);
|
  parallel for review in f.1 do
    review.agent makeReview(review);
    if com.groiss.wf.SystemAction.isActive("addReviews") then
       sync(finishParfor, com.groiss.event.EventHandler, f);
    end;
  end;
end;
```

Figure 2.24: Parfor with additional Branches at Run Time and Synchronization - WDL

elements and the restriction to them does far the disadvantages.

In the area of business process modeling, the discussion did not yet lead to such an agreement. Freund and Rücker show in [?] an example for a process (cf. figure 2.25), that does not lead itself to simple structured reformulation. But actually, its not that complicated; a reformulation can take place along the structure presented above in figure 2.18.

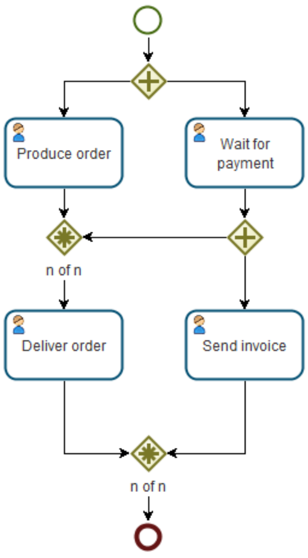


Figure 2.25: An unstructured Process

In the Business Process Execution Language (BPEL) already mentioned before, solely structured constructs are used. Granted, this language has the focus on execution and is not targeted as tool for process analysts. For quick sketch of a process, especially in the analysis phase, a unstructured approach without really formal restrictions is often easier to grasp and therefore appropriate. The ultimate formulation of a business process nevertheless should be carried out in a structured way.

2.2.11 Correctness of Process Diagrams

On what base can be decided if a given process graph is a syntactically correctly structured one? We construct a graph grammar like already mentioned above [**?**]. This formalism defines a start graph and a set of transformation rules. Each graph that can be generated from the start graph by successive application of the transformation rules, must be a correct one in terms of this graph grammar. Figure 2.26 shows the graph grammar on which our model of process definition is based.

For the definition of this graph grammar, we have to describe types of edges and types of nodes. The edge types are well known; there is the standard edge and there is the default edge leading from conditional evaluations.

Also the node types are well known at this point. But for the sake of a simple formulation of the grammar we will merge several node types together resulting in an *activity* node, and we will also differentiate between some nodes which share the same representation in BPMN. In particular, there are the following node types: *activity* denotes all elementary activities (Task, Subprocess, Batch, System step). *If, while, exit_when* are conditional nodes in the corresponding control structures. *par* and *branch* signify the start of parallelism; *andjoin* and *orjoin* are used at the end of parallelism; *end_if*, *end_choice*, *loop*: are nodes where the executions of two paths converge; *begin*, *end*, *end_branch*: process start and process termination.

In the graph grammar, we define the initial graph, which is just a *Begin* node and an *End* node connected by a standard edge. Then the individual rules follow. Most of them substitute an edge by a complete but rudimentary control structure; in the substituting structure, the edge type (variable x) will be retained for one edge. Every edge can be substituted by this principle, with the exception of the edge between the *choice* and *choice_branch* nodes. The rules for *par branch* and *choice branch* insert a new edge into those structures.

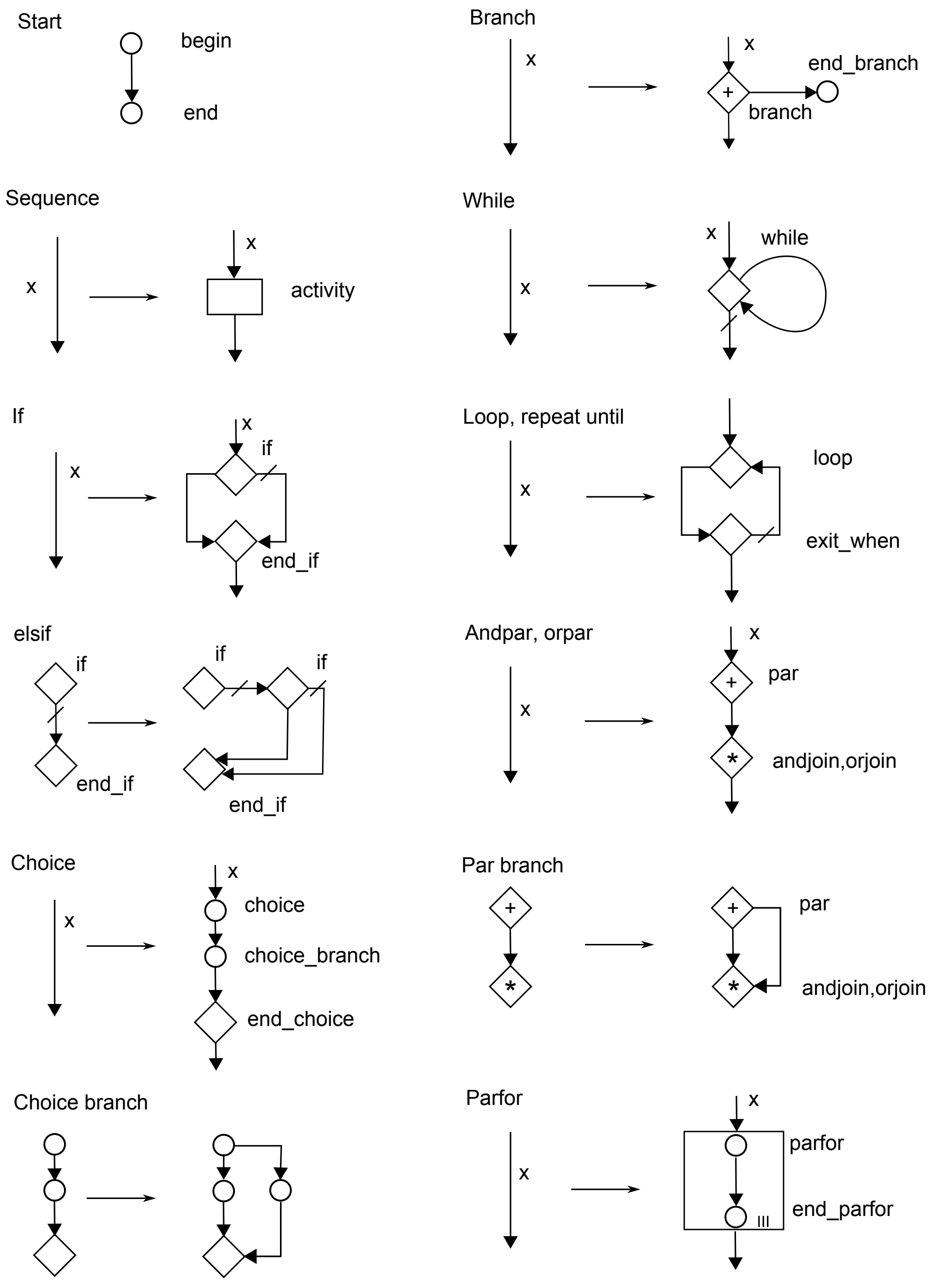


Figure 2.26: Graph Grammar

Normally the designer is not concerned with the syntactic correctness of the resulting process diagram, since it is constructed with the help of a tool that ensures the rule conformity (e.g. the @enterprise Process Editor).

2.2.12 Exception Handling

During process execution there can arise a number of exceptional situations, which should already be taken into account during modeling.

Temporal exceptions

will be handled by escalation actions. There are several constellations where exception handling would be needed:

- ❑ overrun process due dates,
- ❑ overrun due dates at tasks,
- ❑ timeout at waiting for an event,
- ❑ delayed start of a task

With such temporal exceptions, we can associate escalation actions that will be initiated with a temporal offset to the (potential) event, e.g. in anticipation some hours before the due date or reactively some days afterwards.

The escalation action can be to start a task or a process, it can be just the notification of responsible personnel via mail, or it could involve arbitrary complex processing via self written methods.

Error handling

Errors occurring in a step which is carried out automatically without user interaction (batch steps or hitherto unmentioned web service calls) can be handled by predefined actions. In the process we can associate an escalation path with those steps. Within this path an appropriate action can be taken.

All other errors occur in circumstances where a user action triggered them. The run time environment catches those errors, rolls back the changes imposed by the action and presents an error message to the user.

2.2.13 What will not be modeled?

Its highly advisable during process modeling to concentrate on the common and general case first. Thereby one can get an uncluttered big picture and coarse structure which is usually quite easy to comprehend. One interesting question then would be which and to what extend special cases, seldom occurring deviations and ad-hoc changes have to be incorporated into the model.

Let us start with a simple trouble shooting process (see figure 2.27).

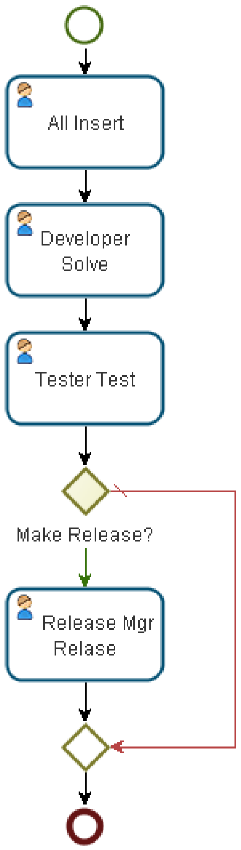


Figure 2.27: Trouble shooting Process

The *Solve* step is followed by a *Test* step. There is some (not too small) probability, that the tester would find some remaining issues and the process has to be returned to the developer. This could be explicitly modeled with a loop. Rather often, it is better to abstain from modeling the loop and to allow the ad-hoc returning (*go back*) to a previous step. In chapter 3 we will discuss the different deviations from the standard process execution path in somewhat more detail.

2.2.14 Process Definition using Rules

There is a class of processes for which the modeling with the aforementioned constructs is tedious. Usually this are unstructured processes or processes which are not completely structured. A simple example would be a process consisting of three tasks, which are not dependent on each other, they could be executed in an arbitrary order, or in an order to be determined at run time by a previous participant. Such "semistructured" processes are not easily modeled in a strictly structured way.

An example may show the practical relevance of such process classes:

Electronic File Processing - There are a number of different, more or less precisely defined tasks, e.g. *Approve*, *Handle*, *Check*, *Issue*, but no predefined sequence between the tasks. But there is a catalog of rules, when and under what conditions a particular task may be carried out. E.g. after an *Approve* step, there might be an *Issue* step but no further *Handle* step.

For processes which fall in the class of the above example, rule based description and modeling is most suitable. The process definition adheres to a set of rules; each rule has a condition part and an action part:

The condition part describes a certain state of the process (like: what is the current step, which step have already been carried out, what data is attached to the process) and denotes the circumstances under which the rule is applicable.

The action part is a possible course of further process actions (possible successor steps, agents to be assigned to those steps and updates to process data).

The execution of such a process works like this: upon finishing an activity, the current process state is passed to a rule engine, which selects the applicable rules and determines candidate successor actions. A user interaction would select the specific actions. If there are no applicable actions, the process would terminate.

Let us conclude the process modeling section with the comment that there are processes which cannot be modeled with all the mechanisms presented above. Quite simply this are processes where the further course of action is not known ex ante, neither as structure nor via existing rules. The process would be treated in a solely "ad-hoc" manner, either at the process start or even during run time. We will treat this aspect in section 3.5.

2.3 Definition of the Agents

Up to now, we focused on the sequence and flow of the activities. For each interactive activity, is must be stated by whom it is to be carried out. There are several possible classes to describe such agents:

User

The activity is to be performed by a specific user and this is to be modeled in the process definition. Hardly ever this makes sense. Even in the case of a single person available for the step, the particular person could change over time. In a process definition, do not use single persons as agent description; it is better to use "General Secretary" rather than "Ban Ki-moon" [2].

Role

The declaration of a role is the most important and widely used variant of agent description.

Role within organizational unit

The combined notation of role and organizational unit restricts the potential agent to those who were assigned the given role in the given organizational unit, e.g. "Head of Department" in organizational unit "Marketing".

Agent of previous step

This is a reference to the agent of a particular (and already finished) step in this process instance, e.g. when the agent most acquainted with the case from a previous participation is to be consulted again with this case, or when the applicant which started the process should be informed about its outcome.

[2] In contrast, determining a certain person at run time is perfectly sensible.

Agent from process data

Determining the agent can also be done in a very flexible ad-hoc manner. The agent of a successor step can be read from the process data (from a form field). In a previous step, the agent for another step has been determined and written to the form field. This agent could again be a user, a role, or a combination of role and organizational unit.

Agent from program

The agent can also be determined at run time via a call to a method, e.g.:

- ❑ Rotation principle: the tasks are evenly distributed upon the members of a role (round robin).
- ❑ Load based assignment: The assignment mechanism takes into account the workload of the users. The user with the least load "wins".
- ❑ Data driven assignment: like distribution by initial character of surname or via area of residence.

Combinations are also feasible; like stating a role in the process model and to determine the specific agent at task creation time via a program.

Empty agent

When no agent is stated at an interactive activity in the process model, the concrete agent must nevertheless be determined. This is ensured by the workflow engine; when a user finishes a step where the agent of the successor step in not known, the user is prompted to manually enter the agent of the successor step.

The example in figure 2.28 illustrates some of the possible agent descriptions:

An order process is to be started by every user (role *All*). The orders should be executed by arbitrary users. The issuer starts the order process and fills in an application form. One of the form fields *form.agent* is designated to contain the person who should process the order. After finishing the order task, the process will be routed to this person. If the designated agent is not able or willing to process the order, she could change the content of the *agent* form field and send send it on. If the order can be processed, form field *finished* is set, and the process execution resumes at the originating issuer (the actual performer of step *order*), in task *inform*.

```
begin
 <order> all order(form);
 loop
   form.agent dojob(form);
   exit when form.finished = 1;
 end;
if (form.notify = 1) then
 order:user inform(form);
end
```

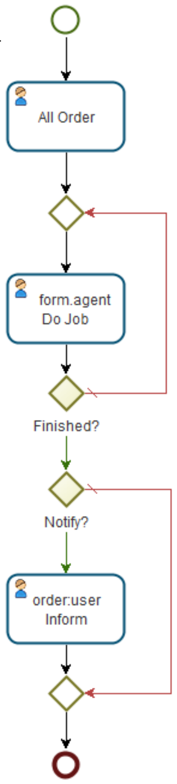


Figure 2.28: Job processing

The agent definitions in the process are: a role in the *order* step, an agent from process data in the *dojob* step, and an agent of previous step in the *inform* step.

An additional example demonstrating the different agent definitions can be found in section 2.8.

2.4 Data Modeling

A central feature of BPM systems is the ability to transport data along the process path, to react to external changes to the data and to apply changes to the data. Data that are directly related to a process are called *Process Data*. Data that are being used in the system but are concerning more than one process are called *Process Relevant Data*. The process data are usually being administered by the BPMS itself. The process relevant data can also be placed and administered in external systems. When the need arises, they could be retrieved by the BPMS, and possibly written back in a changed form to the original location. We will deal with the aspect of communication with external systems in section 3.8.

When creating a model for a process which manipulates data, also this data has to be modeled at a certain extend. Two aspects have to be taken into account; the intrinsic logical schema of the data and the data presentation. In the following we will deal with both areas; for the schema, we will use the form paradigm, the presentation will be based on the XForms standard.

2.4.1 Form based Data Modeling

The main classification of data is along the axis structured versus unstructured data.

Structured Data

The definition of the data structure is usually centered around a conceptual schema denoted by an Entity-Relationship diagram or a similar formalism. After such a data schema has been designed (the definition of it is outside the scope of this publication, see e.g. [**?**]), we separate the data into the categories of process data and process relevant data.

The following simple example will help to illustrate this. Figure 2.29 depicts the data schema for an order process. The process instance itself is represented as an entity. The order and the order items are the process data. There is exactly one order per process instance and vice versa (it is a 1:1 relation). Each order references one customer, each order item references one product. Customers and products are process relevant data.

Unstructured Data

Unstructured data are data which are not further decomposed in their parts within the BPMS, like pictures or text documents. Each of those unstructured

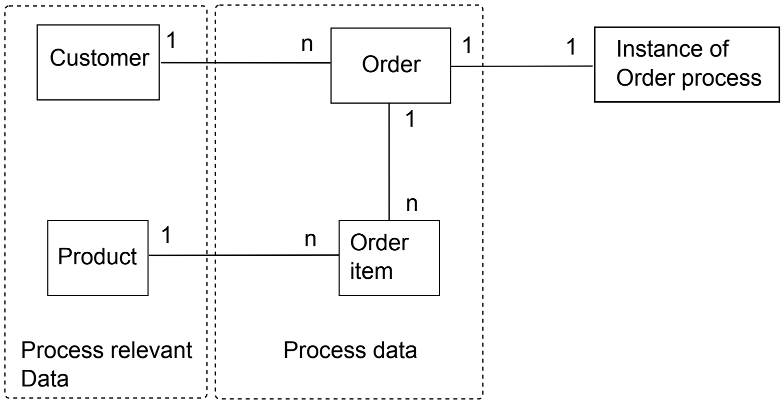


Figure 2.29: Data Schema for an Order Process

documents may have a set of (usually structured) meta data attached, which must be administered by the system.

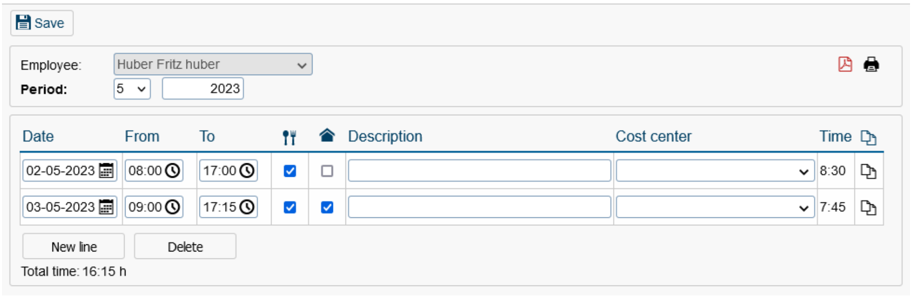
The kind and structure of the meta data depends on the nature (the type) of the data object. For an ocr-ed text document we may be interested in the creator, the recipient, the reception date etc., for a photo we would like to know when and where the photo has been taken.

Forms and Documents

To store structured data, we use the *Form* metaphor. A data structure instance is administered as a form, its structure is captured by a *form type*. One form type is employed to describe and store uniformly structured data of one type. A form type definition consists of a set of the fields of the form, each of which has a name and a data type, of information for the display of the form and of notions relevant for filling out those forms (data creation and updates). The fields are scalar ones, i.e. they have a single value and no further structure.

To represent tabular data in forms, we use *Subforms*. At the containing form - the main form - we define that a second form type relates to it in an 1:n relationship (the main form has a table consisting of n uniformly structured lines). When a user fills out the main form, she can insert, update and delete the subforms. In figure 2.30 such a main form with a subform is depicted.

For unstructured data, we use the *Document* metaphor. Since documents can contain meta data, we represent those metadata as form types as well. Such


Save

Employee:	Huber Fritz huber
Period:	5 2023

Date	From	To	🍴	🏠	Description	Cost center	Time
02-05-2023	08:00	17:00	☑	☐			8:30
03-05-2023	09:00	17:15	☑	☑			7:45

New line Delete

Total time: 16:15 h

Figure 2.30: Example Form Representation

form types would consist of fields for the structured meta data and also have an additional "content" field to hold the unstructured content of the document. In analogy to form types, we use the term *Document Type*.

A special kind of unstructured data are *Notes*. They can be attached to process instances or to documents. Usually they emerge during process execution as an ad-hoc kind of communication between the participants, serve as reminders, explanations, or logs of decisions.

Documents can be filed into *folders*. Folders also have the characteristics of forms like some meta data and a screen mask. The definitions of folders are called *Folder Types*.

Data Types

For the individual fields of the forms, the data types must be stated. The usual standard data types like Boolean, String, Integer, Float, Date etc. can be used. The type of a field can also be a (reference to a) form, or can refer to some of the master data of the BPMS like users, roles, organizational units.

Enumeration types, also called list of values are a further kind of types. Those types can only hold values from a predefined set (e.g. languages, countries, product groups). In practice, the representations of those enumerations can vary over time. *Value-List Forms* support this change by providing a flexible mechanism to define such sets of values.

Each value list has a unique id and a set of values. For each logical value a fixed value for storage and a changeable display value can be stated.

Connecting Data and Processes

The connection between processes and forms is part of the detailed process definition. In WDL, there will be a form declaration line in the process header, the syntax is explained on the base of the following example:

```
forms order orderform, del deliveryform;
```

After the keyword `forms` there is a set of comma separated pairs of form name and form type. Each of the form names can be used in the rest of the process definition like a variable name (it can e.g. be used to construct conditions for alternatives or loops). The form fields can be accessed via a dotted notation (e.g. `order.customer`). When a process instance is being started, for each of the form variables a form instance is created correspondingly. The forms can be acted upon via a user interface.

Documents attached to the process are deposited in a folder that belongs to the process instance in a 1:1 relationship.

For process relevant data, we have to model initial navigation roots. That can be particular folders in the process independent part of the document management system which themselves can contain folders and documents, or can be links to other data like tables of master data.

The next step is the mapping of the conceptual schema to the form paradigm. The entities will be represented as form types. The relationships between the entities can be mapped as follows:

❑ 1:n Relationships: there are several alternatives, according to existence dependency of the objects:
  – Subforms: are employed for relationships to *n* dependent objects, e.g. an order holds *n* order items or a company has *n* street addresses. Subforms are attached to main forms when the main form is defined.
  – Storage of the foreign key: for relationships with independent objects, e.g. consider a publication having exactly one author, the key of the author object is stored as foreign key in a field of the publication objects form type.
  – Folder: this kind of representation is advantageous when the dependent entities are heterogeneous (have several different types) or are unstructured data (ie. documents). There would be a folder as container object in which the diverse entities can be inserted.

- ❑ n:m Relationships: for each of such relationships, a separate relation form type must be created. This form type could be defined as a subform of one of the related entity types. The other entity type could be referenced via a foreign key. The decision which form type to use as the main form of the relationship subform is influenced by logical dependencies or the preferred way of presentation and manipulation. As an example, the assignment of roles to users is usually administered from the viewpoint of the user entity, ie. we would add a subform *UserRole* to the user main form. The *UserRole* form contains a field for selection of the related role object and further optional fields.

Relationships that relate more than two entity types (like n:m:k relationships) are treated like binary n:m relationships, we create a relationship type and reference the form types of the related entities via foreign key fields.

On the basis of the order process example presented in figure 2.29, this means that Customer, Product, Order and OrderItem are defined as form types. OrderItem is a subform of Order, a Customer form is referenced in the Order form, a Product form is referenced by the OrderItem form.

In the document management system, there will be a folder for customers where the Customer form instances will be stored as well as a folder where Product form instances will be inserted.

The discussed approach allows to define arbitrary complex data schemata on the basis of the elementary data structures and to relate data and processes in an appropriate manner.

2.4.2 Data Presentation with XForms

Besides the data schema and the mapping to data structures of the system, the data representation is the second aspect of data modeling. In this section, we will discuss how form data can be presented by application of the XForms standard of the World Wide Web Consortium (W3C) [**?**]. This standard describes a set of form elements and the associated logic.

The form elements can be embedded in an XML based language, with XHTML (the XML version of HTML) being the natural choice. The naming of the form elements is closely related to the corresponding HTML elements, like the following selection field:

```
<select1 ref="method">
  <label>Select Payment Method:</label>
  <item>
    <label>Cash</label>
    <value>cash</value>
  </item>
  <item>
    <label>Credit</label>
    <value>cc</value>
  </item>
</select1>
```

The element is used to select one of several alternatives. It is notable, that:

- ❑ There is no specification, whether the selection takes place via radio buttons or with a drop down list. The exact kind can be specified by the optional attribute `appearance`.
- ❑ Labels are part of the form elements. This is quite helpful for visibilities and input assistance mechanisms, etc.
- ❑ There is no `value` attribute. The *select* element references by means of the `ref` attribute to another element of an XML structure which contains the instance data. The reference is formulated as XPath expression. XPath [**?**] is a language to navigate in XML documents.

The structure of the form and the content (the form instance data) are separated from each other. The instance data is represented in an XML document, the structure of this document is not prescribed by the standard. In the form, the instance data is embedded in a `model` element, which also holds some additional information. The following XML fragment shows such a `model` element (from [**?**]):

```
<model>
  <instance>
    <payment>
      <method>cc</method>
      <number>1234539299549</number>
      <expiry></expiry>
    </payment>
  </instance>
  <submission action="http://example.com/submit" method="post"/>
</model>
```

The data is nested in a an `instance` element. The `submission` Element designates where the form should be send when the user submits it.

A complete XForm consists of a document embedded in a host language (e.g. XHTML), which has a `model` element in the header and where the form elements are positioned arbitrarily in the XHTML document.

Part of the presentation layer of the data is the logic in the context of insertion and update operations. The main aspects here are:

- ❑ Input check and assistance: assurance, that only valid values can be entered. Easy input of compound values e.g. combinations of date and time. Selection of existing data objects (e.g. Customers, Products).
- ❑ Dynamic rendering: fade in and fade out of form parts, depending on context (current step in current process instance), depending on values of other form fields, according to user privileges.
- ❑ Calculation of dependent values: summing up fields, etc.

In XForms, this logic can be specified with `bind` elements and XPath expressions. A `bind` element may have the following attributes, all of them (excluding the `type` attribute) are XPath expressions:

- ❑ `type`: specification of data type. The input can be restricted to permissible values, and appropriate input assistance can be provided automatically (e.g. a date picker).
- ❑ `constraint`: additional restrictions of the range of values can be expressed as constraints. Also dependencies between fields can be stated.
- ❑ `required`: denotes if the field is mandatory.
- ❑ `readonly`: denotes if the field can be changed.
- ❑ `relevant`: denotes if the field will be rendered at all.
- ❑ `calculate`: an expression can be used to calculate the fields value.

A special feature of XForms is the declarative nature of those definitions, there is no need to use other scripting languages like JavaScript.

The elements for form entry at the user interface are the usual ones provided by such frameworks:

- ❑ Input fields: single line fields, multi line text areas, fields used for passwords
- ❑ Single Selections: Radio-Buttons, Select-Lists
- ❑ Multiple Selections: Select-Lists, Checkboxes
- ❑ Sliders: to select numerical values

The run time component responsible for XForm rendering and processing recognizes the types of the fields and can adapt the input field accordingly, e.g. an input field for a boolean value is rendered as a checkbox, an input field for date values is decorated with a calendar widget.

The XForms `repeat` construct allows to operate on tables which are embedded into the form. Insertions, changes and deletions are possible, cf. subforms in section 2.4.1.

Figure 2.30 on page 62 already showed such a form with diverse elements. The fields Employee and Project are of type User and Project, therefore a selection list with search capabilities is rendered. The Period of Time field has a selection list for month numbers (with fixed set of values) and an input field for the year. The table lines below are placed within an `repeat` element, the table can be changed with the buttons positioned below it.

XForms Run Time Component

The processing of a form instance in the BPMS can be outlined as follows:

1. The XForm template for the form type is fetched from its storage location and is being parsed.
2. In the `model` element, an `instance` element with the form instance data and the context data is added. The fields of the form are addressable via XPath (e.g. via `/data/form/`*fieldname*).
3. The predefined field visibilities are inserted as `bind` elements of the `model` element.

4. Appropriate submit buttons and corresponding URLs are added to the actions.
5. The XForm is rendered as HTML page: the individual form elements are translated into their HTML equivalents, are filled with the data from the `model` and are rendered according to the visibilities.

The following example shows the `model` of a form instance with the fields `name`, `country` and `amount` in the context of a process for business trip approval:

```
<xf:model>
  <xf:instance>
    <data xmlns="">
      <form object="com.dec.avw.appl.wiztest_1:1000074412">
        <transactionId>4</transactionId>\
        <avwcreatedby>herbert groiss</avwcreatedby>
        <avwcreatedat>2009-04-06T07:05:22Z</avwcreatedat>
        <avwchangedby>herbert groiss</avwchangedby>
        <avwchangedat>2009-04-07T08:28:22Z</avwchangedat>
        <name>John Doe</name>
        <destination>Wien</destination>
        <cost>40011</cost>
      </form>
      <context>
        <activityinstance oid="42420">158</activityinstance>
        <processinstance oid="42417">158</processinstance>
        <task oid="10185" id="request">Anfordern</task>
        <processdefinition oid="10090" id="hr_businesstrip">
         Business Trip</processdefinition>
        <viewmode>view_text</viewmode>
      </context>
    </data>
  </xf:instance>
  <xf:bind nodeset="/data/form/name" required="false()"
    type="string" />
  <xf:bind nodeset="/data/form/country" required="false()"
    type="string" />
  <xf:bind nodeset="/data/form/amount" required="false()"
    type="decimal" />
  <xf:submission replace="instance" validate="false"
    action="com.groiss.storegui.FormWrapper.updateNoAction"
    id="submit0" method="post" />
  <xf:submission method="post" id="submit1"
```

```
    action="com.groiss.storegui.FormWrapper.updateAndAction?\
    javaAction=finish&afterSubmit=top.right.location=comingFrom"
    />
  <xf:submission afterSubmit=parent.parent.changeTab()"
    action="com.groiss.storegui.FormWrapper.updateAndAction?\
    method="post" id="submit2" validate="false"/>
</xf:model>\
```

Besides the form fields, some context is inserted to to allow for convenient and immediate access to the most important environmental data:

- ❑ `activityinstance`: the object id (oid) and the textual representation of the current activity instance
- ❑ `processinstance`: the oid of the process instance and the process (instance) id
- ❑ `task`: oid, id and name of the current task
- ❑ `processdefinition`: the oid, the id and the name of the process definition
- ❑ `viewmode`: the viewmode indicates whether the form is being opened for viewing, for editing or for printing

We will illustrate the application of XForm concepts by giving several examples of form fragments. The particular path strings, URLs and access to list of values and configuration data correspond to the XForms implementation of @enterprise. Other implementations may differ in this aspects.

Example: Summing up Subforms. In a billing form there is a subform with the individual items. In the main form, the summarized amount is to be displayed. A `bind` element with a `calculate` attribute can be used for summation:

```
<xf:bind nodeset="/data/form/totalamount"
   calculate="sum(/data/form/subform/form/itemtotal)"/>
```

Example: Hiding of a field. Depending on the data of some form field, other fields should be either displayed or being hidden: the input field for description of transport should only be shown when the selected option for transport type was miscellaneous. In the `model`, this is defined by the following `bind` element:

```
<xf:bind nodeset="/data/form/transpdesc"
   relevant="/data/form/transporttype = 'misc'"/>
```

In the form, there is an output field defined for `transporttype`, while `transpdesc` is an input field:

```
<xf:select1 ref="/data/context/transporttype">
  <xf:item><xf:label>Train</xf:label>
     <xf:value>train</xf:value></xf:item>
  <xf:item><xf:label>Car</xf:label>
     <xf:value>car</xf:value></xf:item>
  <xf:item><xf:label>Miscellaneous</xf:label>
     <xf:value>misc</xf:value></xf:item>
</xf:select1>
<xf:input ref="/data/form/transpdesc">
  <xf:label>Transport Description:</xf:label>
</xf:input>
```

Example: Conditionally set a field to read only. Field `curecost` is only editable when field `reason` has the value `cure`.

```
<xf:bind nodeset="/data/form/curecost"
readonly="/data/form/reason != 'cure'"/>
```

Example: Usage of value lists (cf. section 2.4.1). We will show how value lists can be put into use. The different types of absences (holidays, nursing leave, sickness, etc.) are captured in a value list.

The value list can be edited in the document management system of @enterprise. In a folder, one can create a new value list, define its id (*absencetype*) and create the individual values and their corresponding labels. In the XForm, we give a individual `model` element for the value list:

```
<xf:model id="valuelist">
  <xf:instance
   src="com.groiss.wf.html.ValueList.show?id=absencetype"/>
</xf:model>
```

As `src` attribute, this URL must be stated, the id parameter of the URL is the id of the value list. When more than one value list is used, the id parameter of the URL is the comma separated list of the ids of the value lists.

Within the `body` of the XForm, there is a selection element that references to a value list by giving the id of the model element (`valuelist`) and the id of the value list (`absencetype`):

```
<xf:select1 ref="/data/form/type"><xf:label>Typ:</xf:label>
   <xf:itemset model="valuelist"
     nodeset="/valuelists/list[@id='absencetype']/item">
      <xf:label ref="label"/>
      <xf:value ref="value"/>
   </xf:itemset>
</xf:select1>
```

In the former example, we do not explicitly state the data in the `model` element but rather specify an URL as a method for obtaining the data.

Example: Configurations data. In the form, we would like to display the currency symbol that has been chosen in the system configuration, and the kilometer allowance should be calculated based upon the rate which has also been defined in the configuration.

To use configuration parameters in an XForm, the `model` element must get an additional `instance` element with an includes a `configuration` element. In this nested element, all the names of the needed configuration properties are stated. The names consist of application id prefix, a colon, and the parameter name within the application. For system parameters, no prefix is given. The current values of those parameters will be appropriately inserted at run time.

```
<xf:instance>
  <data xmlns="">
    <configuration>
      <property name="staffprocs:kmallowance" />
      <property name="staffprocs:dailyallowance.domestic" />
      <property name="staffprocs:currency.symbol" />
    </configuration>
  </data>
</xf:instance>
...
<xf:bind id="currency"
  nodeset="//property[@name='staffprocs:currency.symbol']"/>
<xf:bind nodeset="/data/form/kmamount"
  calculate="//property[@name='staffprocs:kmallowance'] *
             /data/form/km"/>
```

To summarize: the tasks in data modeling start with the definition of a conceptual data schema which also means to classify the data according to its usage in the processes and their interdependencies. On this base, form types and their relationships can be defined. For each form type there is an XHTML page with the corresponding XForms elements, like input fields and bind elements. The creation of such forms in @enterprise will be presented in section 2.7.4.

2.4.3 Form Visibilities

The definition of the visibilities of form field with XForm concepts, namely with XPath expressions in the three attributes `relevant`,`readonly` and `required` is rather cumbersome. Especially when we need to define the visibilities for each formfield for each process step. We map the four sensible combinations of the attributes to four form field modes:

Mode	relevant	readonly	required
invisible (inv)	false	false	false
read only (ro)	true	true	false
read write (rw)	true	false	false
mandatory (man)	true	false	true

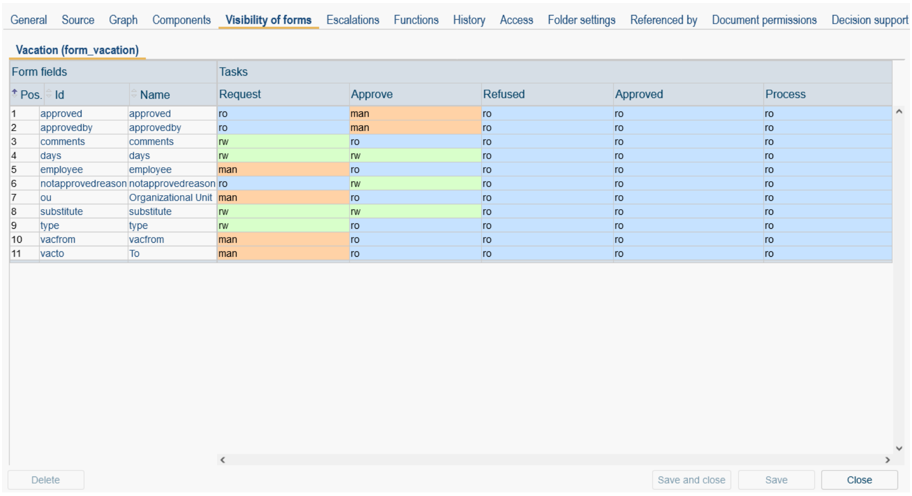
For subforms (which are embedded tables) there is an additional mode. The individual forms are editable but deletion of subform instances or creation of new subform instances is forbidden.

Those modes can be defined for all form variables per form field and per process activity in a set of tables.

Figure 2.31 shows the visibilities of a form in a process definition. For each form variable, a tab would be shown. The columns of the table are the individual process activities, the lines of the table are the form fields. In the tables cells the aforementioned modes can be found.

2.5 Conditions

During process modeling we had to formulate conditions. Naturally those conditions reference process data, therefore we deferred the treatment of this aspect until the data modeling has been discussed. Now we can catch up on the topic of conditions. Those occur at the following locations in a process definition:



Form fields			Tasks				
Pos.	Id	Name	Request	Approve	Refused	Approved	Process
1	approved	approved	ro	man	ro	ro	ro
2	approvedby	approvedby	ro	man	ro	ro	ro
3	comments	comments	rw	ro	ro	ro	ro
4	days	days	rw	rw	ro	ro	ro
5	employee	employee	man	ro	ro	ro	ro
6	notapprovedreason	notapprovedreason	ro	rw	ro	ro	ro
7	ou	Organizational Unit	man	ro	ro	ro	ro
8	substitute	substitute	rw	rw	ro	ro	ro
9	type	type	rw	ro	ro	ro	ro
10	vacfrom	vacfrom	man	ro	ro	ro	ro
11	vacto	To	man	ro	ro	ro	ro

Figure 2.31: Form Field Visibilities

- ❑ Alternatives
- ❑ Loops
- ❑ Selection (Choice)

Another part of process definitions are postconditions of tasks, that are conditions that must hold after a task is completed and which will be checked when a participant wants to finish the task.

All those conditions can refer to data from several form variables and to other data as well. We provide three mechanisms to define conditions. In WDL, simple conditions can be constructed directly by combining comparison expressions with boolean operators; the WDL examples for *If*, *Choice* and the different forms of loops already contained such conditions (cf. section 2.2.5).

The second possible way to formulate conditions is via XPath expressions, those have already been mentioned in the context of the XForms. For such XPath conditions in the process context, we provide the access to all process form instances and additional information via predefined variables:

- ❑ `pi`: the process instance

- `ai`: the activity instance
- `user`: the current agent
- `form_{id}`: the form variable with the stated id
- `configuration`: the system configuration
- `configuration_{applid}`: the configuration of the application with the stated id

Starting with this variables, we can now formulate conditions.

Example: The value in field totalamount of form order must be greater than 3000:

```
$form_order/totalamount > 3000
```

Naturally, the variable `form_order` references the process form variable `order`, there must be a formfield with the name `totalamount`.

Example: Is the value of the field `toagent` in form `order` equal to the agent who started the process?

```
$form_order/toagent = $pi/agent
```

The third way for condition formulation are arbitrary (Java-) methods, especially used when complex requirements exhaust the power of XPath with respect to formulation of conditions (e.g. access to external systems). In this methods, an easy way to access the process context is provided as well.

2.6 Functions

Interactive tasks require the agent to perform certain operations, e.g. to fill in the process forms or to attach documents. But also programs can be called, thereby achieving partial automation of the task. Data from an external system or from another process instance could be used or inserted into the current process instance.

Part of modeling is also to determine which functions should be provided in which tasks to deliver effective assistance to the agents and to relieve them from routine operations. Those functions are to be defined according to their interfaces to the system, but also concerning their integration in the user interface.

Besides task related functions, there are also global functions which could be executed independent of a certain process context or state. Usually those functions are in the area of process relevant data, like triggering data transfers between the BPMS and external systems.

2.7 Process Modeling with @enterprise

In this section we will sketch the concrete modeling of processes with @enterprise.

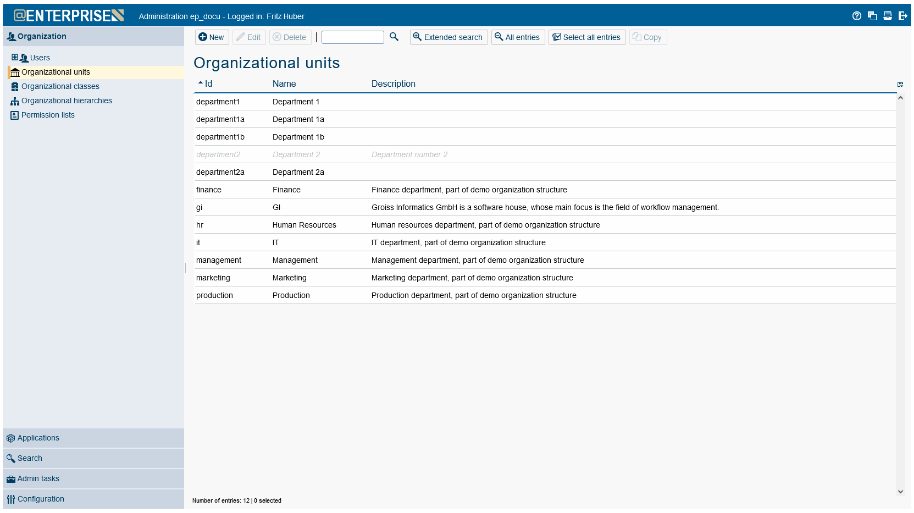
The first step is to get access to an @enterprise installation. This can be achieved after after a quick (no obligations or costs apply) registration via the web site of Groiss Informatics GmbH (*https://www.groiss.com*). Then the system can be downloaded for local installation or an online demo instance can be created.

In both variants, an account is created by choosing a user name and a password. This can be used to log into the system, where the workflow client interface of @enterprise is displayed (see figure 3.7 on page 96). Clicking on the administration link (🖵 in the right corner of the top toolbar open the user settings, choose administration) opens the administration console view, depicted in 2.32.

Starting with this mask, all functions for modeling are accessible. A detailed description can be found in the @enterprise administration manual. We will give just a brief overview here. In the left of the window, there is the navigation area with links to all the administrative functions. In the top area there is the tool bar which presents all functions applicable for the current view in the main area.

2.7.1 Organizational Modeling

The first group of links in the navigation area can be used to capture the organizational data. Editing of the individual object types follows a common pattern. Clicking the navigation link presents a table view of the respective objects. A double click on a line opens the detail view of the selected object. In the tool bar, the functions *Create*, *Edit* (corresponds to the double click action), *Delete* and *Quick Search* are available.



Organizational units

Id	Name	Description
department1	Department 1	
department1a	Department 1a	
department1b	Department 1b	
department2	Department 2	Department number 2
department2a	Department 2a	
finance	Finance	Finance department, part of demo organization structure
gi	GI	Groiss Informatics GmbH is a software house, whose main focus is the field of workflow management.
hr	Human Resources	Human resources department, part of demo organization structure
it	IT	IT department, part of demo organization structure
management	Management	Management department, part of demo organization structure
marketing	Marketing	Marketing department, part of demo organization structure
production	Production	Production department, part of demo organization structure

Number of entries: 12 | 0 selected

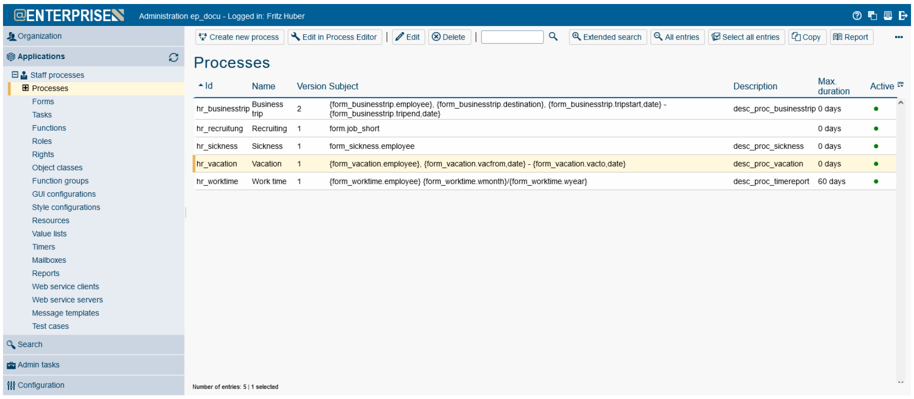
Figure 2.32: @enterprise Administration

In the opened detail windows, the mandatory fields (which can be recognized via their labels in bold face) must be filled in. The edit operation is completed by clicking either *OK* or *Apply*.

Each object has a key value in the *Id* field. This field must not be empty, may not contain special characters or white space. It serves as unique identification label of the object. In the area of organizational data, there is a second mandatory field, the *Name* field.

We start with the entry of organizational units. This can be accomplished like described above from the table of organizational units, or it can be initiated from the organizational hierarchy. In the detail view of the default organizational tree, a click on the *Hierarchy* tab opens this view where new organizational units can be entered and be placed in the tree.

Then we define the users and assign them to organizational units. In the navigation we click on *Users* and then on tool bar function *New*. After entering *Id*, *First Name* and *Surname*, the user object can be stored by clicking the button *Apply*. The second tab of the user object detail mask (*Role assignments*) can be used to assign the newly created user to an organizational unit. For this, the role *Home* must be assigned and the desired department must be selected.



Processes

Id	Name	Version	Subject	Description	Max. duration	Active
hr_businesstrip	Business trip	2	{form_businesstrip.employee}, {form_businesstrip.destination}, {form_businesstrip.tripstart,date} - {form_businesstrip.tripend,date}	desc_proc_businesstrip	0 days	●
hr_recruitung	Recruiting	1	form.job_short		0 days	●
hr_sickness	Sickness	1	form_sickness.employee	desc_proc_sickness	0 days	●
hr_vacation	Vacation	1	{form_vacation.employee}, {form_vacation.vacfrom,date} - {form_vacation.vacto,date}	desc_proc_vacation	0 days	●
hr_worktime	Work time	1	{form_worktime.employee} {form_worktime.wmonth}/{form_worktime.wyear}	desc_proc_timereport	60 days	●

Number of entries: 5 | 1 selected

Figure 2.33: Processes of an Application

Additional role assignments would require the definition of additional roles first, this will be described later on.

2.7.2 Process Modeling

The second group of navigation links is devoted to an *Application* which is the term used to describe a group of related process definitions and their supporting components. Examples for such process groups are the ones from chapter 1: personnel processes, IT service processes. Before we can define a process, we have to create such an *Application*.

After an application has been edited (filling out the *Id* and *Name* fields is sufficient), a new group of navigation links will appear below the name of the application in the navigation area. The first link leads to the list of processes (ie. to the process definitions) see figure 2.33. Process definitions can be created by means of a process editor, which is activated via the first link in the tool bar.

Process Editor

The process editor is the tool that allows to model the process flow in a graphical manner. Related components like tasks and roles can also be created with it.

After defining a complete process with the editor, instances of it can be immediately created and executed in the workflow engine. It is also possible to use the process editor in an early modeling stage, without the need to define it completely at the moment. Incomplete process definitions often do not exactly specify the data or are sketchy concerning the conditions in decision nodes. In a later modeling stage, enhancements, amendments and details can be applied to the process definition successively until it is fully defined and executable.

When creating a new process definition, an *Id* and a *Name* must be assigned by selecting the properties item in the process menu and entering both information in the mask.

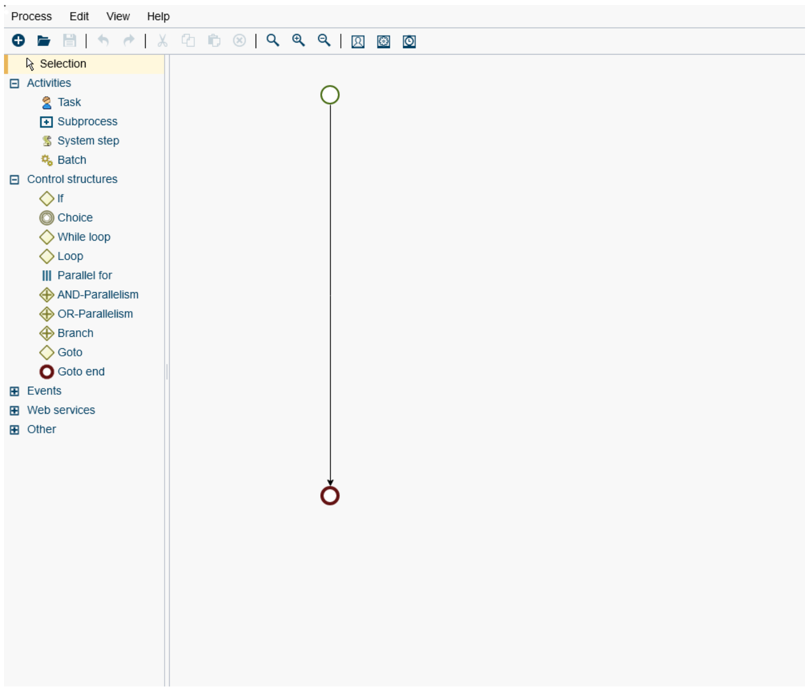


Figure 2.34: Process Editor

As discussed during the treatment of the graph grammar, a new empty process consists of a start node and an end node connected by a directed edge like in figure 2.34. New process nodes are created by selecting one of the constructs in the left area and then clicking an edge. A node or structure of the selected type is inserted at this place. The original edge is replaced by edged properly connecting the new nodes. Deletion of a node carries out the inverse operation.

Generally, every edge can be used as the target of an insertion operation, with the exception of the edge between a *Choice* node and a subsequent condition node.

For most node types, it is necessary to define some properties after creation of new node instances. Double clicking a node opens the type specific property window.

If the process is not yet intended to be executable, the entry of just a description would be sufficient here. When executeability is desired, the following information is needed in a correct manner:

- ❑ Tasks: selection of an existing task or creation of a new one. Entry of an agent.
- ❑ Subprocess: selection of a defined subprocess to be called.
- ❑ Conditions: for loops and alternatives, the conditions must be defined.
- ❑ Choice: each path must have a name, the condition is optional.

More detailed instructions for operation of the process editor can be found in the administration manual of @enterprise [**?**].

2.7.3 Tasks and Roles

Tasks and roles can be created during process definition in the process editor, and can also be independently created via the navigation links below the application object.

Especially roles like *Head of Department* or *Clerk* which are rather independent from particular process definitions will be created in this manner. We recommend to concentrate process independent roles in an application (e.g. the *Default*-application).

2.7.4 Forms

Data modeling is accomplished via the navigation link *Forms* in the applications navigation tree (see figure 2.33).

Clicking on the first tool bar item "Create new form type" opens the form editor (see figure 2.35), in which both the layout and the data types can be defined:

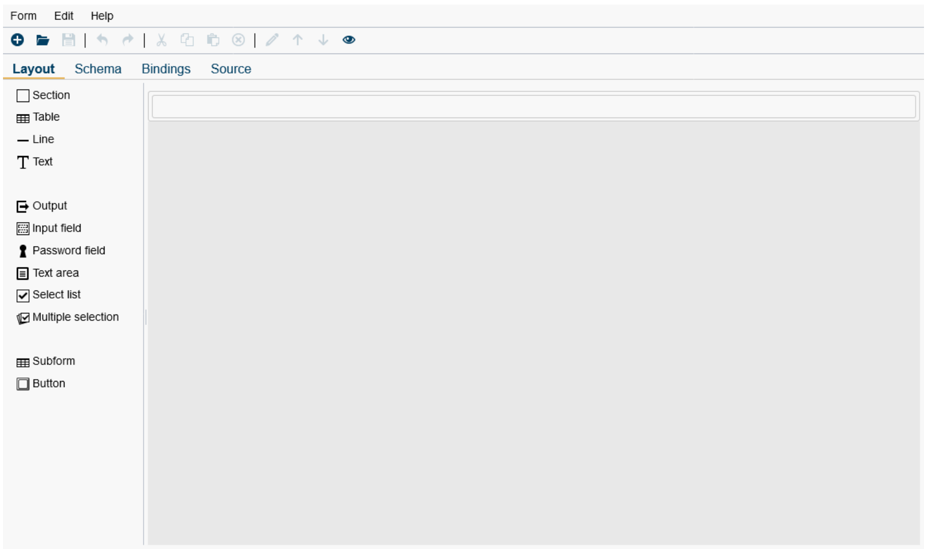


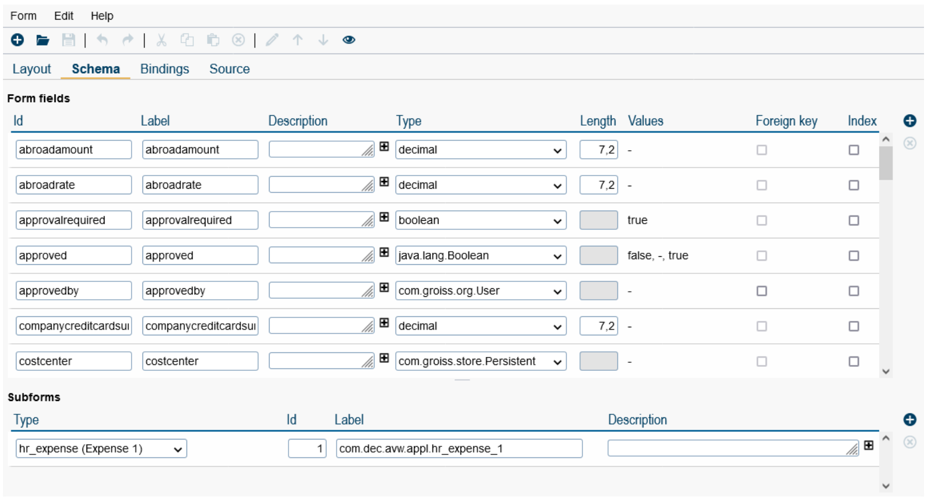
Figure 2.35: Form Type Creation

- ❑ The HTML layout is defined in the first tab. The individual input elements (see the left area in the figure) can be added to the form with drag-and-drop.
- ❑ In the second tab - Scheme - the individual database fields can be defined: names, types, length of the fields in the database etc., see Fig. 2.36. The layout can also be created from this schema information (Menu function: Create standard layout).
- ❑ In the Bindings tab, XForms bindings can be defined for the fields.
- ❑ In the last tab, Source, the HTML can be edited directly.

When saving, you have to assign a name and an ID, then the Java class and a database table are created and the HTML is stored in the file system.

2.7.5 Process Documentation

Generated documentation for processes modeled with @enterprise can be displayed with function "Process Overview". In a single HTML page or a PDF



Form fields

Id	Label	Description	Type	Length	Values	Foreign key	Index
abroadamount	abroadamount		decimal	7,2	-	☐	☐
abroadrate	abroadrate		decimal	7,2	-	☐	☐
approvalrequired	approvalrequired		boolean		true	☐	☐
approved	approved		java.lang.Boolean		false, -, true	☐	☐
approvedby	approvedby		com.groiss.org.User		-	☐	☐
companycreditcardsu	companycreditcardsu		decimal	7,2	-	☐	☐
costcenter	costcenter		com.groiss.store.Persistent		-	☐	☐

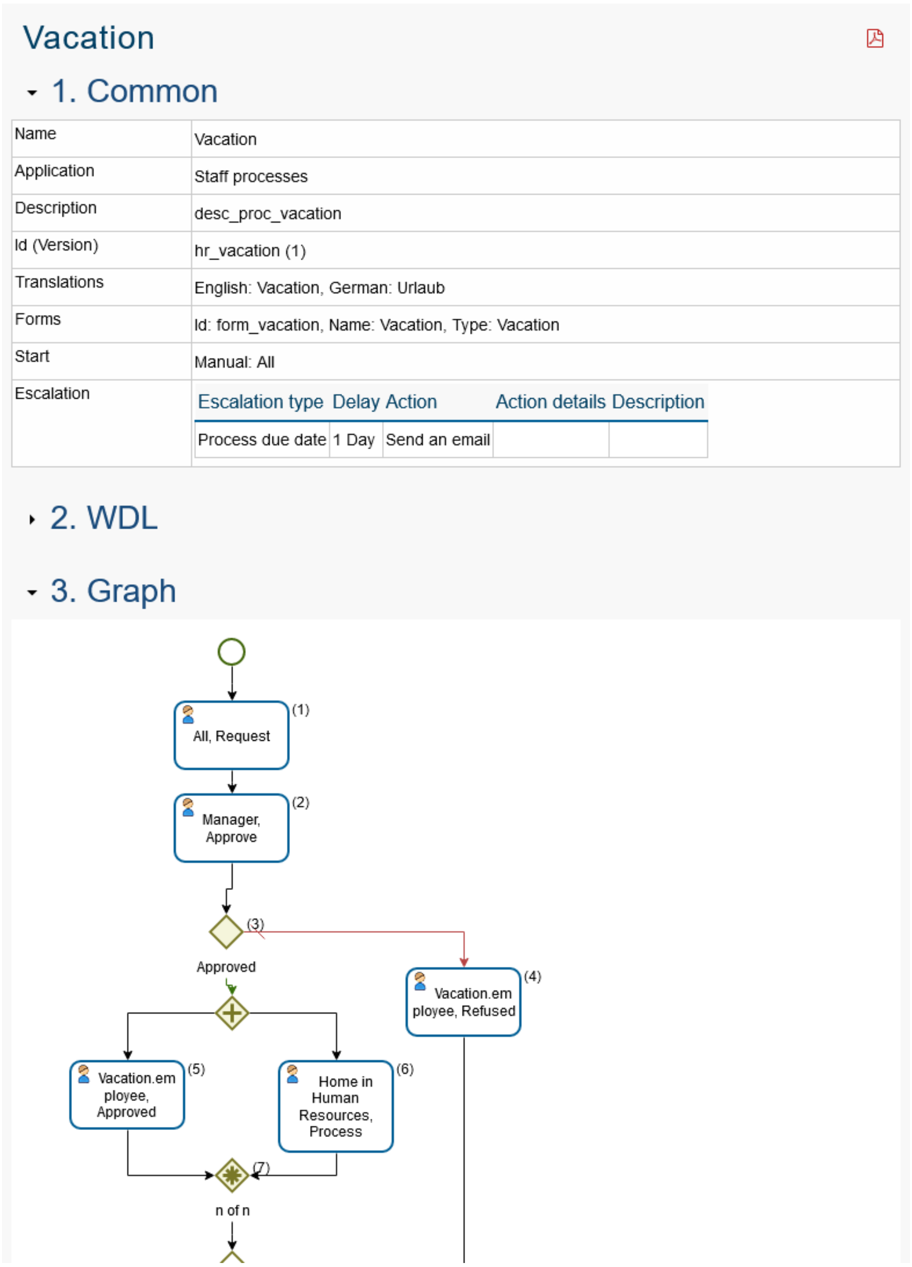
Subforms

Type	Id	Label	Description
hr_expense (Expense 1)	1	com.dec.avw.appl.hr_expense_1	

Figure 2.36: Form Type Creation - Determining Fields and Types

document all aspects of a process are depicted: graphical process definition, WDL, tasks, roles, forms, form field visibilities. Figure 2.37 shows the HTML format, clicking the icon in the right upper corner would generate the PDF document.

The process overview is meant as documentation and can facilitate the discussion with process participants. In section 4.1.3 we will introduce the process cockpit, which combines a representation of the defined process types with current run time data of the process instances.



Vacation

1. Common

Name	Vacation
Application	Staff processes
Description	desc_proc_vacation
Id (Version)	hr_vacation (1)
Translations	English: Vacation, German: Urlaub
Forms	Id: form_vacation, Name: Vacation, Type: Vacation
Start	Manual: All
Escalation	

Escalation type	Delay	Action	Action details	Description
Process due date	1 Day	Send an email		

2. WDL

3. Graph

All, Request
(1)
Manager, Approve
(2)
(3)
Approved
Vacation.employee, Refused
(4)
Vacation.employee, Approved
(5)
Home in Human Resources, Process
(6)
(7)
n of n

Figure 2.37: Process Documentation

2.8 A complete Example

To conclude our discussion of the process modeling, we will present a small but complete example. The requirements for our process are:

We need a process to facilitate the requesting, approval and granting of permissions to employees. A permission in this context is the assignment of a resource to an employee. This can be in several functions (or roles). E.g. for the resource "ERP system" the function "super user" can be assigned.

Requests should be initiated by the applicants themselves. The request is to be approved by a manager. There can be partial approvals, that is a manager can selectively remove individual application items from a request. After approval, the resource assignment takes place by an employee responsible for this resource. Finally, the requester is informed about the completion of the process.

2.8.1 Data

The designed data schema is depicted in figure 2.38.

Resources have a name and a responsible employee. There is an 1:n relationship between resources and functions. For functions, @enterprise roles could be used directly; an approach which would give more flexibility for later adaptions and extension would be the definition of a separate function entity class.

The permission request has an applicant field, a 1:n relationship to permissions and a boolean field to indicate approval or denial of the request.

One permission request can be used to apply for several permissions, each of those permissions reference a resource and a function.

The mapping to forms is as follows: each entity type is mapped to a form type, permissions are a subform of permission request, functions are a subform of resource.

The forms can be created with the form wizard. The resources are placed in a folder of the document management system. The figures 2.39 and 2.40 show the source of the permission request form and its rendering in the browser.

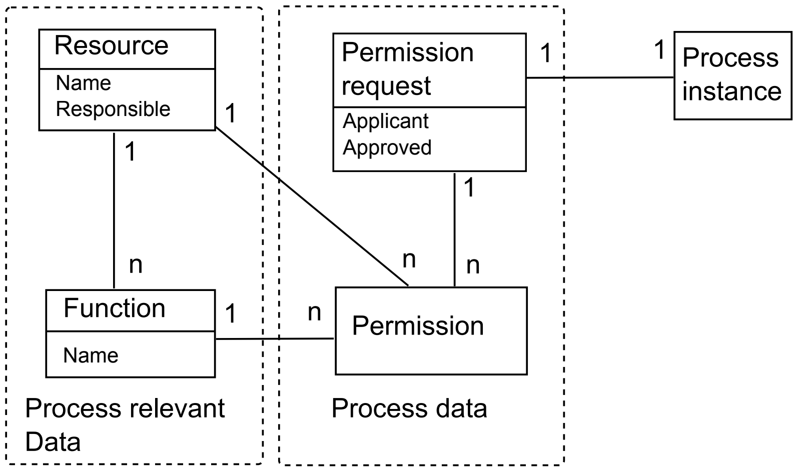


Figure 2.38: Data Schema for Permission Request Process

2.8.2 Process

The process graph can be found in figure 2.41, the WDL script is shown in figure 2.42.

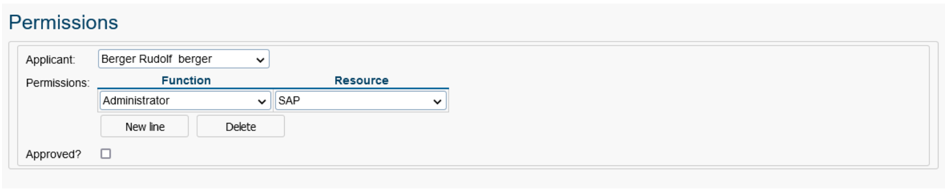
The definition of the agents should be noted. The first step is assigned to the role `all`, each employee who has been assigned this role, can start the process. The approval is carried out by a `manager`. The grant operations are carried out in parallel for each permission via the `parfor` construct. The agent is the employee responsible for this particular resource. Since this cannot be expressed in WDL directly, a function is used for agent assignment in this step. The last step references the agent of the first step (the original applying employee).

After process flow and data structures have been defined, the field visibilities can be determined for each step. The table in figure 2.43 shows the form visibilities used here.

Each line shows the same pattern, a field is first invisible (it is not relevant in an early process step), then is is editable (`rw`) or mandatory (`man`) in one step, and in later steps it is just read only (`ro`). Normally, a process form will be filled in from top to bottom, ie. the fields relevant for the first step are located at the top of the form, then the fields relevant for the second step will be positioned below them, etc. Therefore, a typical pattern for the form visibility table is a

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:xf="http://www.w3.org/2002/xforms">
 <head>
  <link rel="stylesheet" type="text/css"
    href="../html/avw.css"></link>
  <xf:model></xf:model>
 </head>
 <body id="theform">
  <div id="title" class="title">Permission Request</div>
  <table id="formtab">
   <tr><td>
     <xf:input ref="/data/form/applicant">
       <xf:label class="label100">Applicant</xf:label>
     </xf:input>
   </td></tr>
   <tr><td>
     <xf:repeat formtype="com.dec.avw.appl.demo_permisson_1"
       subformid="1">
      <xf:label class="label100">Permissions</xf:label>
      </xf:repeat>
   </td></tr>
   <tr><td>
     <xf:input ref="/data/form/approved">
      <xf:label class="label100">approved</xf:label>
     </xf:input>
   </td></tr>
  </table>
 </body>
</html>
```

Figure 2.39: Form for Permission Request


Permissions

Applicant:	Berger Rudolf berger	
Permissions:	Function	Resource
	Administrator	SAP
	New line	Delete
Approved?	☐	

Figure 2.40: Permission Request Display

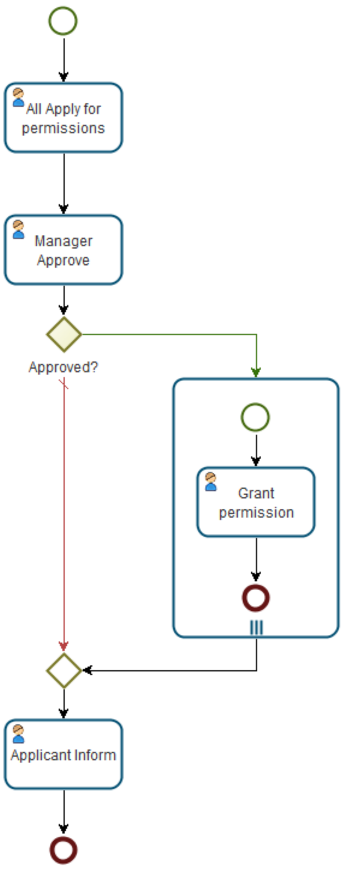


Figure 2.41: Permission Request Process Graph

diagonal area from the upper left corner to the lower right corner, where the field visibilities are `rw` or `man`. The lower left triangle area has many `inv`, the upper right triangle area consists mainly of `ro` entries.

```
process br_assignpermissions()
version 5;
name "Permission Request";
forms request br_permissions "Form";
application permissions;
begin
   <Applicant> all br_request(request) "Apply for permissions";
   br_manager br_approve(request) "Approve";
   if com.groiss.wf.SystemAction.getFormFieldValue(
        "request.approved") then "Approved?"
     parallel for permission "Permission" in request.1 do
       com.groiss.wf.SystemAction.getFormFieldValue(
         "permission.resource.responsible")
          br_grant(permission, request) "Grant permission";
     end;
   end;
   Applicant:user br_inform(request) "Inform";
end
```

Figure 2.42: WDL of Permission Request Process

Form Fields	Request	Approve	Grant	Inform
Applicant	man	ro	ro	ro
Permissions	rw	rw	ro	ro
approved	inv	man	ro	ro

Figure 2.43: Form field visibilities

Chapter 3

Process Execution

After discussing process modeling in the previous section, this chapter will deal with process execution within a BPM system. We will start with the description of the architecture of a BPMS and a workflow engine, will introduce the user interface and essential functions of the system and will also discuss the permission system and aspects of systems integration.

3.1 Architecture

A BPMS consists of all software components that are needed to model and execute business processes and to analyze their run time behavior. The already mentioned Workflow Management Coalition (*WfMC)* has developed a reference model as part of their glossary [**?**]. Our model presented in figure 3.1 is based on this reference works. It contains the following elements:

- ❑ Modeling: the modeling component allows to define processes, data structures, organizational structures and additional application elements.

- ❑ Execution: the run-time component (the *workflow engine*) is responsible for the execution of the processes. The *document management system* allows to store and retrieve documents. A *timer* is used to support time based events and actions.

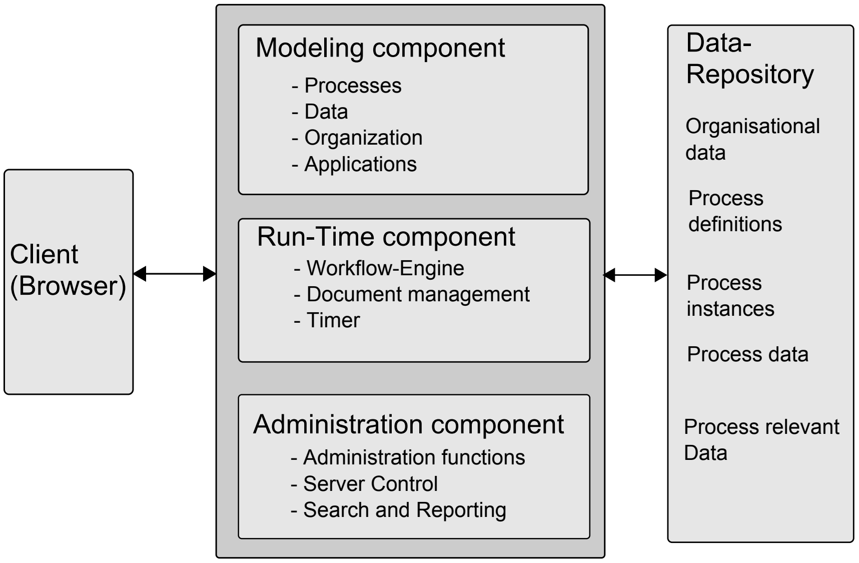


Figure 3.1: Architecture of a BPMS

- Administration: there is an *administration component* for management of organizational structures, user accounts, etc.; it provides search and reporting capabilities.
- Data-Repository: includes definition data, master data and run time data.
- Client: Usually, clients are standard web browsers, maybe with additional plugin components. But also special BPM clients could be used alternatively.

In the following sections, we will describe the components in more detail.

3.2 The Workflow Engine

The workflow engine is responsible for the control and execution of the processes. It is an interpreter for process definitions and creates process instances and activity instances.

At a high level, the operation of the workflow engine can be described by just two functions: `startActivity` and `finishActivity`. A brief pseudo code representation of the two functions is given in figure 3.2.

When a process instance is started, the procedure *startActivity* is called with the initial node of the process graph. An activity instance gets created. The behavior of the function depends on the type of this node. When the type is `begin`, `par`, `loop`, `end_if` or `end`, function `finishActivity` is called immediately (cf. below). When the node has a condition (node type is `if`, `while`, `exit_when`), the condition is evaluated and depending on the result, either the `then` path or the `else` path is followed; `finishActivity` with the condition node and the path to be followed is called. The two types of nodes that end a parallel construct `orjoin` and `andjoin`) are treated like this: for an `orjoin`, the execution will immediately continue at the successor node, when an `andjoin` is reached, nothing happens until all parallel branches are finished. When the node is of type `task`, the optional preprocessing method is executed, an agent is assigned and the function terminates. While the agent would find a new work item in the corresponding worklist, the engine stops here and waits for the agent to complete the task. For a `system` step, the defined method will be called, and then again `finishActivity` is executed. For nodes of type *process*, the first step of the subprocess is determined and the method gets called recursively with this step.

When a user completes an activity (with function *finish* from the worklist client), the function `finishActivity` is called. All successor nodes of current node are determined. The second parameter of `finishActivity` states the type of the edge to be followed (`normal`, `then`, `else`). For all those corresponding successors, `startActivity` is executed.

For the sake of a principal understanding, the functions were presented in a simplified manner, we deliberately abstained from describing the workings for the node types `batch`, `parfor`, `sync` and `event`.

3.2.1 Structure of Run Time Data

When a process is started, a corresponding *ProcessInstance* object with a unique process id is being created in the database. Fore each activity (every time the function *startActivity*) is called, an *ActivityInstance* is created. Those objects are the core of the *run time data*, since they are being instantiated during the run time of a process instance. In contrast, the data objects created during the modeling phase are named *build time data*. The life cycles of process instances

```
function startActivity(node)
   ActivityInstance ai := create_instanceof(node);
   if typeOf(node) in {begin, par, loop, end_if, end} then
      finishActivity(ai, "normal");

   elsif typeOf(node) in {if, while, exit_when} then
      if executeExpression(node)
         then finishActivity(ai,"then");
         else finishActivity(ai,"else");
      end if;

   elsif typeOf(node) = orjoin then
      if this is the first finished branch then
         finishActivity(ai, "normal");
      end if;

   elsif typeOf(node) = andjoin then
      if this is the last finished branch then
         finishActivity(ai, "normal");
      end if;

   elsif typeOf(node) = task then
      executeProcedure(ai);
      assignAgent(ai);

   elsif typeOf(node) = process then
      node1 := initial node of process node
      startActivity(node1);

   elsif typeOf(node) = system then
      executeProcedure(ai);
      finishActivity(ai, "normal");
   end if;
end;

function finishActivity(ai, edgetype)
   if no successors of ai then
      finishActivity(parent(ai));
   else
      for all successor nodes succ of the ai node with type edgetype do
         startActivity(succ);
      end do;
   end if;
end;
```

Figure 3.2: Central Functions of the Workflow Engine

and activity instances are depicted in the state transition diagrams in figures 3.3 and 3.4.

After a process has been started, it is in state *started*; when is is completed (its last step has been finished) the state changes to *finished*. A running process can be canceled by the administrator, it is then in state *aborted*. Processes can be reactivated from both the *finished* and the *aborted* state.

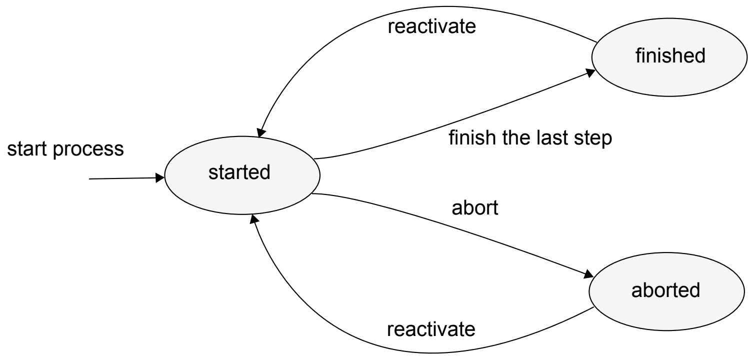


Figure 3.3: Process Instance States

An *ActivityInstance* (see figure 3.4) starts its life in state *started* (when the agent is a role) or in state *active*, when the agent is a user. Taking an item from a role worklist changes the state of the corresponding *ActivityInstance* from *started* to *active*, giving it back effects to the reverse state transition. Temporal interruptions on the work items lead to state *suspended* and the placing of the item in the suspension list. The "recall" operation reverses the transition and restores the previous state. Finishing an activity leads to state *finished* in the corresponding *ActivityInstance* in the general case. The state *waiting* is an intermediate state resulting from the need to interactively select an agent when there is no agent specified at build time or from a half-finished *Choice* node where the user has not ultimately decided on the path to be taken. After completion of those actions, the state changes to *finished*. When a process is aborted, the corresponding unfinished *ActivityInstances* are transitioned to state *aborted*. A finished *ActivityInstance* can be marked as *compensated*, when the function "Go back" is applied to the process.

Let us now discuss the interplay between process instances and activity instances. As stated before, during the start of a process, a process instance is created for it; each start of an activity creates an activity instance. The process

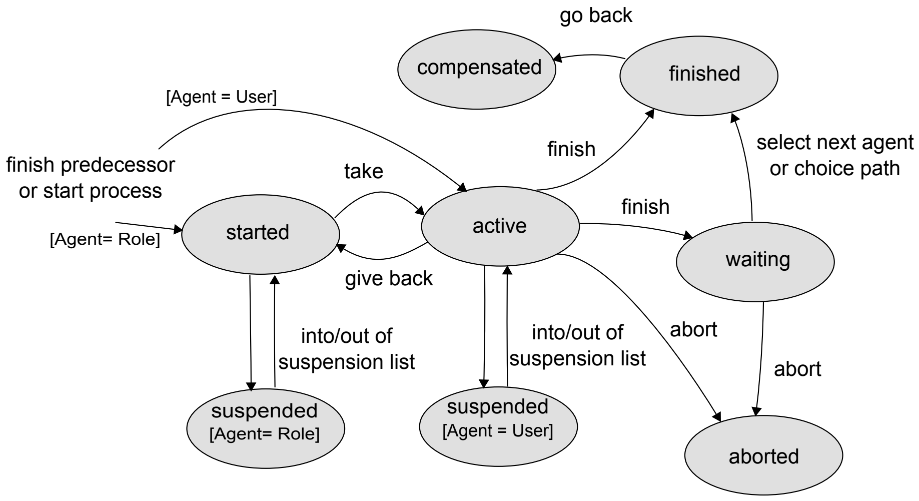


Figure 3.4: Activity Instance States

instances and the activity instances are in a 1:n parent-child-relationship. When there is a subprocess call, the created activity instance is both the child of the super process instance as well as the parent for all the activity instances arising from the steps in the definition of the subprocess.

Figure 3.5 presents an example of a run time data graph. The nodes are activity instances and process instances. The solid edge represents an instance of a corresponding flow edge from the process definition (a successor relation), the dashed edges indicate the child-parent (belongs to) relationship. We have a process instance *pi1*. The process has four steps, one of them (*pi2*) is a subprocess. The activity instances *ai1*, *ai2* and *ai3* are those of the main process *pi1*. The objects *ai4*, *ai5* and *ai6* are the activity instances of the subprocess *pi2*

On the basis of the data schema in figure 3.6, we will now discuss the interplay of build time data and run time data. The build time process definition graph is represented by the two entities *Step* (the nodes) and *Flow*, the edges and the directed relationships between them. A process step relates to an *Activity*, which can be an elementary *Task* or itself be a process in the case of a subprocess call.

The single entity *ActivityInstance* represents both process instances and activity instances. It captures the current state information. The recursive relationship

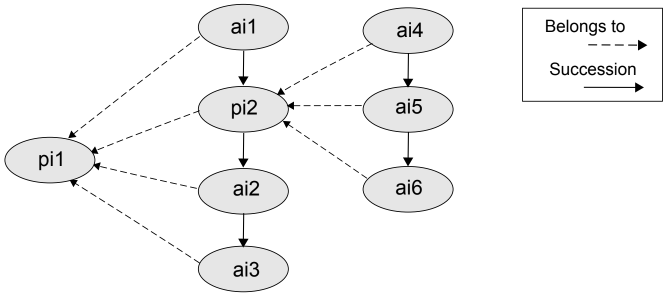


Figure 3.5: Run Time Object Tree

models the parent-child relation (cf. the dashed edges in figure 3.5).

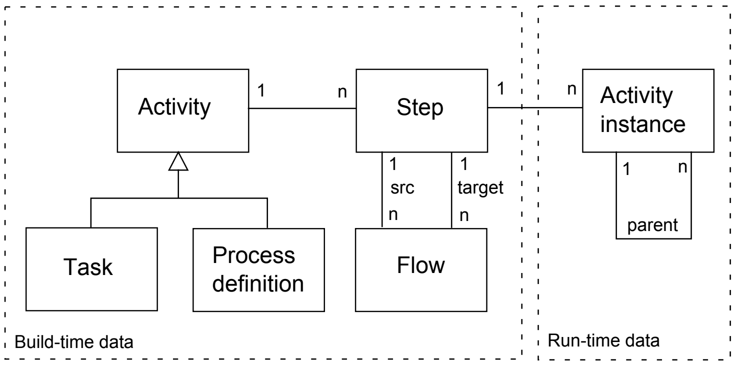
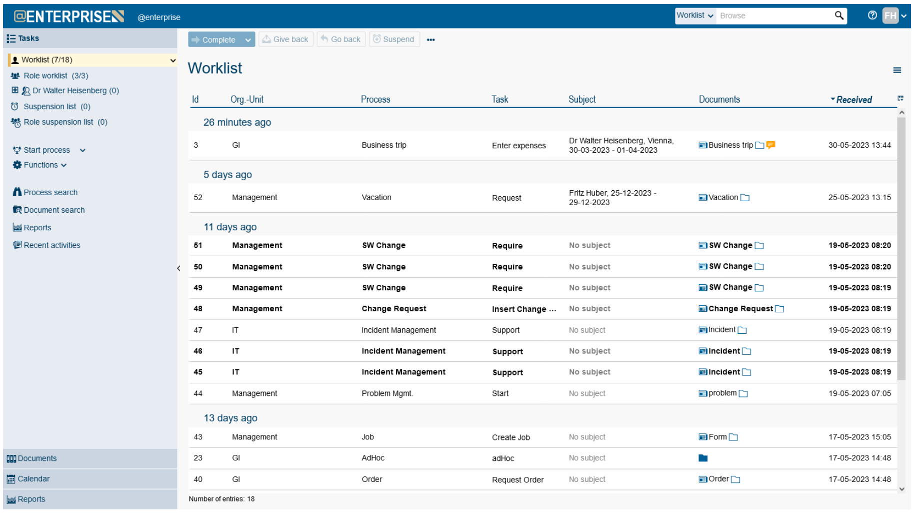


Figure 3.6: Schema for Build-time and Run-time Data

3.3 User Interface

The main functions of the user interface (the *client application*) in a BPMS are to display the list of tasks to be worked on (the *worklist*) and to provide appropriate means and tools to edit the process data and act on the tasks. In figure 3.7, the @enterprise user interface is depicted. This section will discuss central aspects of this interface.

The individual functions are accessible via a navigation area, where the following items can be found:



Worklist

Id	Org.-Unit	Process	Task	Subject	Documents	Received
26 minutes ago						
3	GI	Business trip	Enter expenses	Dr Walter Heisenberg, Vienna, 30-03-2023 - 01-04-2023	Business trip	30-05-2023 13:44
5 days ago						
52	Management	Vacation	Request	Fritz Huber, 25-12-2023 - 29-12-2023	Vacation	25-05-2023 13:15
11 days ago						
51	**Management**	**SW Change**	**Require**	**No subject**	**SW Change**	**19-05-2023 08:20**
50	**Management**	**SW Change**	**Require**	**No subject**	**SW Change**	**19-05-2023 08:20**
49	**Management**	**SW Change**	**Require**	**No subject**	**SW Change**	**19-05-2023 08:19**
48	**Management**	**Change Request**	**Insert Change ...**	**No subject**	**Change Request**	**19-05-2023 08:19**
47	IT	Incident Management	Support	No subject	Incident	19-05-2023 08:19
46	**IT**	**Incident Management**	**Support**	**No subject**	**Incident**	**19-05-2023 08:19**
45	**IT**	**Incident Management**	**Support**	**No subject**	**Incident**	**19-05-2023 08:19**
44	Management	Problem Mgmt.	Start	No subject	problem	19-05-2023 07:05
13 days ago						
43	Management	Job	Create Job	No subject	Form	17-05-2023 15:05
23	GI	AdHoc	adHoc	No subject		17-05-2023 14:48
40	GI	Order	Request Order	No subject	Order	17-05-2023 14:48

Number of entries: 18

Figure 3.7: User Interface, Worklist Client

- ❑ Worklists
- ❑ List of startable processes
- ❑ List of application specific functions
- ❑ Search functions and reports
- ❑ Document management area
- ❑ Calendar
- ❑ User specific properties
- ❑ Dashboard
- ❑ Organizational structure and process overview

The precise set of items that are visible and accessible depends on the roles of the agent as well as on the configuration of the client.

3.3.1 Worklist

This is the most central point of the user interface. It contains the tasks to be worked on. Usually this view is structured in several lists:

- personal worklist: contains activity instances that are either personally addressed to the user or have been taken by her from a role worklist
- role worklist: those activity instances that where addressed to a role (which has been assigned to the user)
- suspension list: personally addressed activity instances, but not ready to be worked on at the moment
- role suspension list: activity instances addressed to a role, but not ready to be worked on at the moment

The structure corresponds to the four states *started*, *active*, *suspended/user* and *suspended/role* depicted in the state diagram in figure 3.4 on page 94.[1]

The worklist contains the tasks to be completed by an individual employee; dependent on the degree of automation and the breadth of processes deployed in the system, a typical worklist can comprise several dozens of entries. To cope with the bulk, some features to organize the entries are provided. A hierarchical folder structure and be created to categorize the tasks e.g. as: to be done today, for tomorrow, urgent, waiting for Godot, ... Filtered views for selecting the entries upon their content values are available to gain additional overview and allow to focus on similar tasks, e.g.:

- process instances of one particular process definition
- all processes exept support processes
- received today

The worklists are tables, the table lines are activity instances (*work items*), the individual columns are essential properties of the items, like:

- Process instance id: the unique identification of the process instance

[1] Activity instances in the states *compensated, finished* and *aborted* cannot be acted on from this interface. Any activity instances in state *waiting* are displayed in the worklist indicating the action they are waiting for.

- ❑ Organizational unit: where the activity instance belongs to
- ❑ Agent: the current agent (in the personal worklist, this is always the user id, so this column is dispensable there)
- ❑ Process name: name of the process definition
- ❑ Task name: name of the task (the step in the process definition)
- ❑ Subject: a characterization of the process instance
- ❑ Functions: a list of specific functions available for this activity instance
- ❑ Forms and documents: links to access the process data
- ❑ Received: timestamp of arrival in the worklist
- ❑ Due at: the deadline for this activity
- ❑ Priority: process instance priority

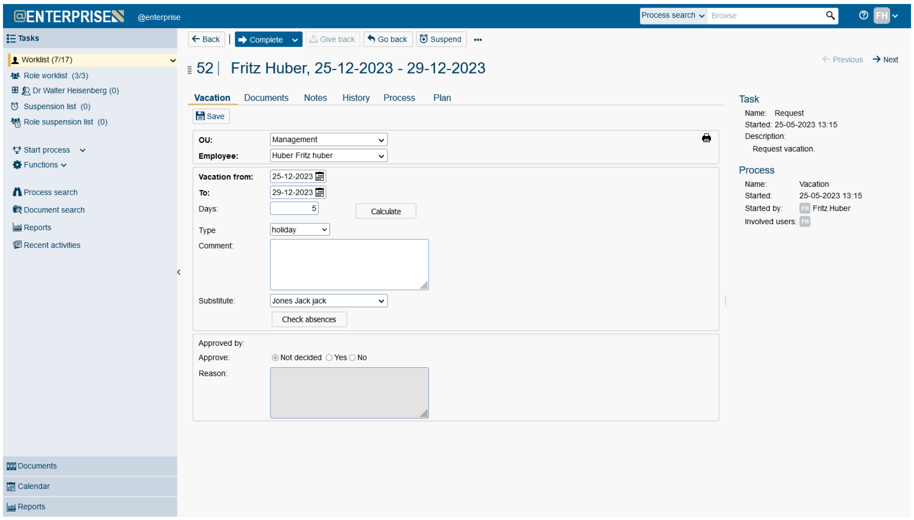
The tabular view does not necessarily show all those information, some columns may not be displayed at all, some columns may be links to the detailed data. Additional navigation links are provided for:

- ❑ location within the process instance: a graphical view with the current tasks highlighted
- ❑ previous flow: the history of the process instance, all the previously executed tasks and agents, including the corresponding time stamps and data
- ❑ process data: process forms, notes and documents attached to the process

Figure 3.8 shows the detail view of an activity instance, the functions will be discussed in the following section.

3.3.2 Functions in the Worklist

The worklist displays the work items and provides functions to handle them. The essential functions are process related calls of the workflow engine like *start process* or *complete* and also editing functions for process data (forms and documents).



52 | Fritz Huber, 25-12-2023 - 29-12-2023

OU: Management
Employee: Huber Fritz huber
Vacation from: 25-12-2023
To: 29-12-2023
Days: 5
Type: holiday
Comment:
Substitute: Jones Jack jack

Approved by:
Approve: [✓ Not decided] Yes No
Reason:

Task
Name: Request
Started: 25-05-2023 13:15
Description:
Request vacation.

Process
Name: Vacation
Started: 25-05-2023 13:15
Started by: FH Fritz Huber
Involved users: FH

Figure 3.8: Detail View of an Activity Instance

Process and activity related functions

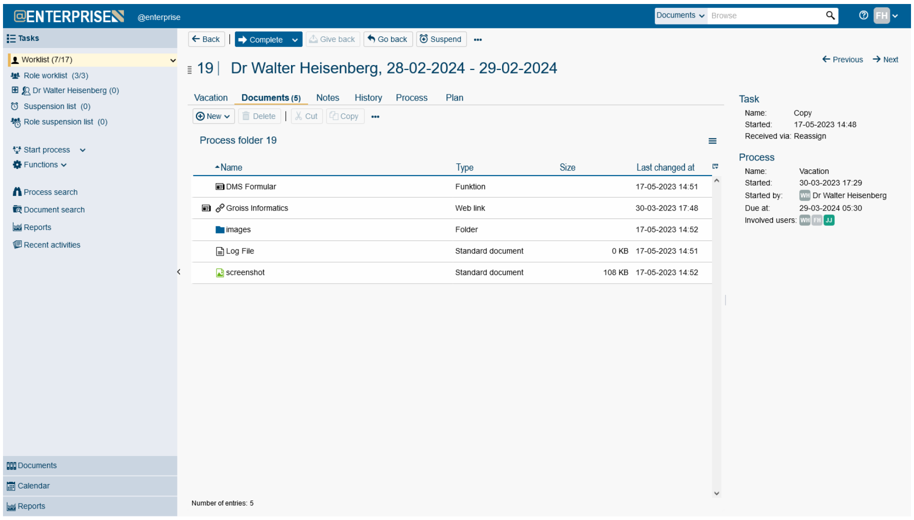
- Complete: the work item is completed by calling the corresponding function of the workflow engine (cf. function `finishActivity` in section 3.2).
- Complete and assign: When the next step in the process flow is an activity instance that is addressed to a role, it is often preferable to select a specific user from the set of users with this role. The user selection is triggered after the completion of the item.
- Go back: The process can be send back to a previous step. This is discussed in detail in section 3.5.1.
- Change agent: delegation of the work item to another user. The process remains in the current step, just the agent of the step is changed. This action is also traceable in the history.
- Copy to: sending of a read only copy of the process to another user, e.g. for informational purposes.
- Give back: if the activity has been taken from a role worklist, it can also be put back there.

- ❑ Suspend: if the activity instance cannot be worked upon at the moment, it can be suspended. This removes the work item from the personal worklist and puts it into the suspension list. Suspended items can be recalled manually as well as automatically by specifying a resume date.
- ❑ Abort process: the process is being deliberately finished, but in an abnormal way.

Data handling functions

Those functions are provided in the detail view of an activity instance. They are the central means to act on the work item (besides completing it in the sense of the workflow engine).

- ❑ Form handling: In the detail view, the leftmost tabs are the process forms. The forms will be displayed according to the form field visibilities specified for the particular step in the process definition.



19 | Dr Walter Heisenberg, 28-02-2024 - 29-02-2024

Process folder 19

Name	Type	Size	Last changed at
DMS Formular	Funktion		17-05-2023 14:51
Groiss Informatics	Web link		30-03-2023 17:48
images	Folder		17-05-2023 14:52
Log File	Standard document	0 KB	17-05-2023 14:51
screenshot	Standard document	108 KB	17-05-2023 14:52

Number of entries: 5

Task
Name: Copy
Started: 17-05-2023 14:48
Received via: Reassign

Process
Name: Vacation
Started: 30-03-2023 17:29
Started by: WH Dr Walter Heisenberg
Due at: 29-03-2024 05:30
Involved users: WH FH JJ

Figure 3.9: Process documents

- ❑ Process documents: the next tab after the form tabs in the detail view shows the documents attached to the process, cf. figure 3.9. For each process instance, there is also a folder for storage of unstructured data (like pictures or some texts). The available functions are: document upload, edit, delete, make version, copy, paste, etc.

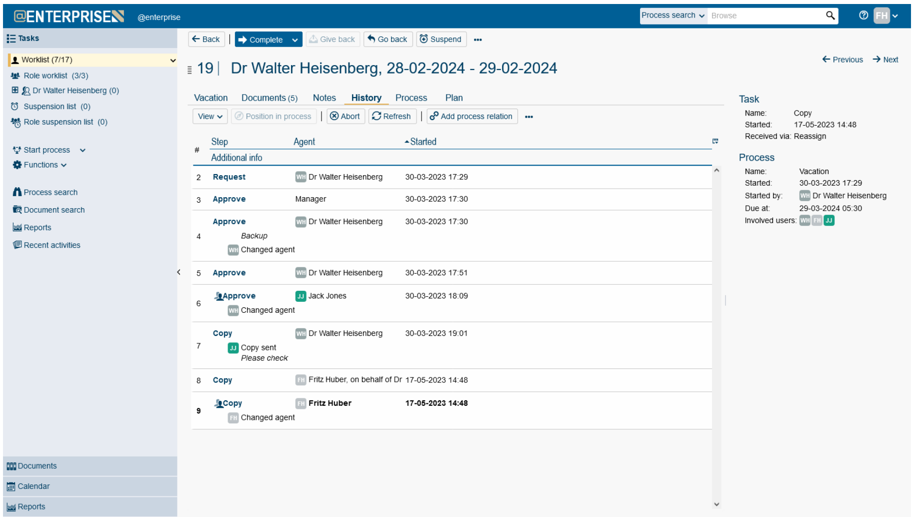
❑ Notes: short notes can be attached to processes, the set of notes of a process is accessible via a tab in the details view.

❑ Change process info: some data of the process instance can be changed at run time, like priority, the due date or the process instance description.

Additional Functions

Depending on the application, additional functional may be available, cf. section 2.6.

3.3.3 Versions and Traceability



19 | Dr Walter Heisenberg, 28-02-2024 - 29-02-2024

#	Step / Additional info	Agent	Started
2	Request	WH Dr Walter Heisenberg	30-03-2023 17:29
3	Approve	Manager	30-03-2023 17:30
4	Approve *Backup* WH Changed agent	WH Dr Walter Heisenberg	30-03-2023 17:30
5	Approve	WH Dr Walter Heisenberg	30-03-2023 17:51
6	Approve WH Changed agent	JJ Jack Jones	30-03-2023 18:09
7	Copy JJ Copy sent *Please check*	WH Dr Walter Heisenberg	30-03-2023 19:01
8	Copy	FH Fritz Huber, on behalf of Dr	17-05-2023 14:48
9	Copy FH Changed agent	**FH Fritz Huber**	**17-05-2023 14:48**

Task

Name:	Copy
Started:	17-05-2023 14:48
Received via:	Reassign

Process

Name:	Vacation
Started:	30-03-2023 17:29
Started by:	WH Dr Walter Heisenberg
Due at:	29-03-2024 05:30
Involved users:	WH FH JJ

Figure 3.10: Process History

Being able to trace, audit and verify executed process actions is one of the main reasons for BPMS deployment. For every process instance, there is detailed data about the historical process flow available. This *process history* contains the following information:

- ❑ Process flow: a tabular view of the executed steps; with the performing agent and corresponding timestamps for each manual step. Optionally the non-interactive steps can be included in the display (alternatives, loops, end nodes, . . .).
- ❑ Data modifications: each data modification is being logged. For each step, the form versions current at those step can be displayed. Changes are traceable down to form field level.

The process history is not only relevant ex post (after process completion), but also during run time for handling the process; providing answers to questions like: "Who completed the previous step?", "By whom and when has this form field been changed?".

3.3.4 Search

Search functions are one of the core components of a workflow system. Each and every process instance, either currently active ones or already finished ones must be findable according to some criteria. We distinguish three search functions:

Short search

There is one input field provided for a search term; it will be searched for in the *subject* or *process id* fields. The search field is permanently visible in the navigation area, providing a quickly accessible basic search capability.

Standard search

Several search fields are provided. Search criteria are e.g. process start time interval, process type, participating person, process data. [2]

The result of short search and standard search actions is a list of process instances matching with the entered criteria. The following columns are displayed:

- ❑ Process instance id

[2]Why is there a rather complex search mask provided, when the usual interface for Internet search engines consists of a single field? The reason is the large amount of very similar data that will accumulate. There might be several thousands of orders for one specific article within the current year. Heuristics like click frequency are of no real use here.

- ❑ Process definition name
- ❑ Tasks and agents where the process is currently located
- ❑ Subject
- ❑ Process instance start time (and completion time, if applicable)

Extended Search (Report Designer)

The extended search module provides the capability to define search criteria on all the available data and to flexibly combine them. The form of the result can be adapted to a great extend. Data aggregations like multilevel counting, summing up and calculation of averages over several process instances are supported and are a main tool for analyzing process instances (cf. chapter 4).

The creation of such queries requires considerable knowledge about the principal structure of the data and is therefore neither intended nor suitable for the general user population. Usually, an administrator defines frequently needed searches. Those queries or *reports* can be stored. Permissions to execute the reports can be granted to users with certain roles. Those users can execute the reports (in a parametrized manner) quite easily without requiring any special in depth knowledge. A list of reports the user is permitted to execute can be accessed via the link *Stored queries* in the navigation area.

In all search functions, the permission system must be taken into account: not everyone is allowed to see each process or to access the process data. Section 3.6 elaborates on the permission system.

3.3.5 User Interface Customization

A crucial point of the user interface is its adaptability. This is in particular true for BPMS as illustrated by the following aspects:

- ❑ Presentation of process data: the worklist should not only display common and general data but also process specific and application specific information.
- ❑ Functions dependent on roles and process progress: the set of executable functions is dependent on the roles assigned to the user and on the step of the process.

- Higher ranking employees like heads of departments would be allowed to execute additional reports and may perform additional functions too, like e.g. maintaining some application specific master data.

The user interface must be adapted with respect to (1) the permissions and roles of the user and (2) dependent on the application, since different data is needed in different application domains. The adaption of user interfaces in @enterprise is discussed in section 3.9.1.

3.3.6 Mobile Access

Up to now, we have been discussing a web based interface that is suitable for common desktop environments. With the almost ubiquitous availability of smartphones (a mobile phone with at least an Internet browser but also other capabilities), this class of devices is becoming more and more relevant for user interface considerations. The most influential factors are:

- Universal device availability (literally billions of smartphones have been sold).
- Universal Internet access with acceptable speed via mobile nets or wifi hotspots.
- Universal device acceptance and usage for mobile work. A report of Optus [?] states that 6% of the employees do work from their home, 36% are working regularly from outside the office (at home or during travel), 39% access the company network via a mobile device.

Until quite recent times, the integration of mobile devices implied to deliver a carefully selected basic (almost rudimentary) subset of the whole functionality like reception of messages, sending of status information or information retrieval.

Currently, an almost complete variant of the capabilities available at the desktop must be provided on the mobile device, too. Prominent challenges are: security, ergonomics, integration of the application with common smartphone functions.

The security aspects will be dealt with in section 3.8.7; in the following, we will discuss the other two topics.

User Interface at Smartphones

There are two possible ways to implement a smartphone client: a web-application that utilizes the devices browser as user interface or an application (an "app") that is to be installed at the smartphone. Both methods have their specific advantages and pitfalls. Here, we will restrict the discussion to web based interfaces.

One main aspect of interface design is the adaption of the navigation and handling to the restrictions arising from available input methods and from small screen areas. In figure 3.11, the start page and the worklist of a mobile version of @enterprise is shown. We abstain from using frames, broad tables and toolbars with small icons. The single real restriction is in the area of document management and document processing, due to the lack of suitable applications on the client.

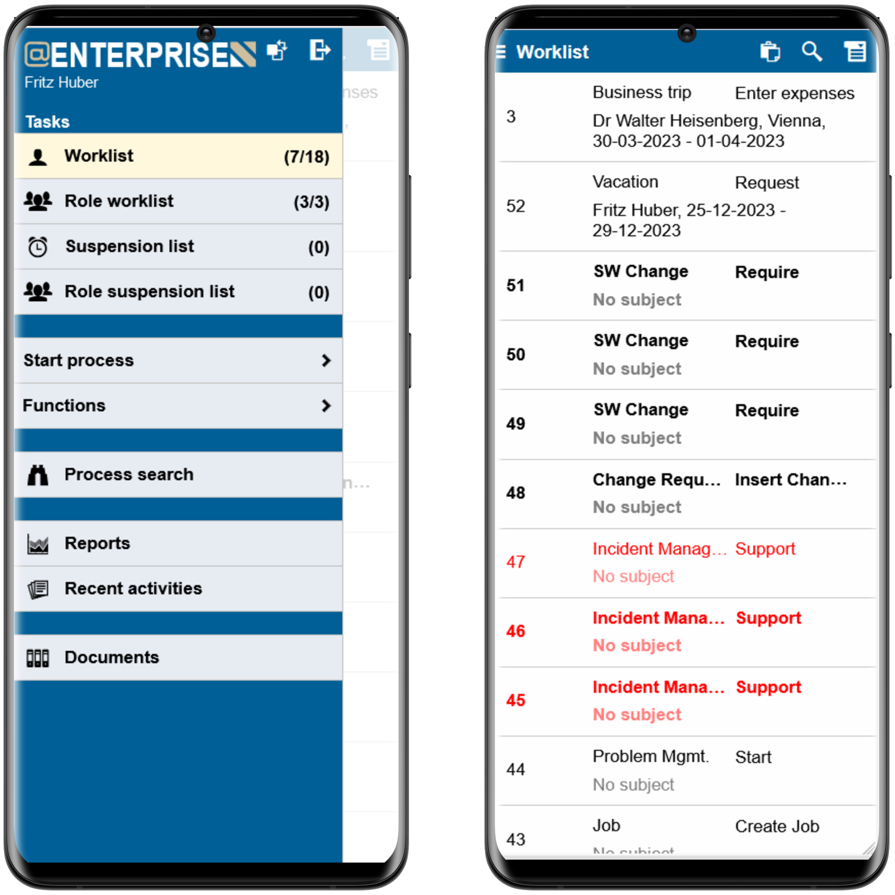
Integration of mobile functions

The forefather of the smartphone was mainly being used to make phone calls while being on the way. Communication functions do also play a central role in current smartphones (making calls, SMS (Short Message Service), MMS (Multimedia Messaging Service), e-mail services. In the collaboration of several users on a business process, the need to communicate with other process participants would arise rather frequently. Integration with smartphone communication services is easy. A profile page provides e-mail address and phone number for each user, they could be specially linked to the corresponding application. Profile pages could be directly accessible from e.g. the history view of a process or from a form.

Another feature of smartphones is the availability of location information via GPS (Global Positioning System), providing the physical position of the device.

Figure 3.12 shows an application for location based services via GPS integration. An employee of a utility maintenance company is on the road and gets a task into his worklist. The form contains the position of the problematic device, a click on button *Show on Map* switches to a map view of the position. Likewise, a click of *Set to Current Loc.* inserts the coordinates of the current position in a form when recording an issue.

Additional integration with smartphone capabilities like attachment of photos, videos or audio recordings to processes are equally feasible.



@ENTERPRISE
Fritz Huber

Tasks

Worklist	(7/18)
Role worklist	(3/3)
Suspension list	(0)
Role suspension list	(0)

Start process >
Functions >

Process search

Reports
Recent activities

Documents

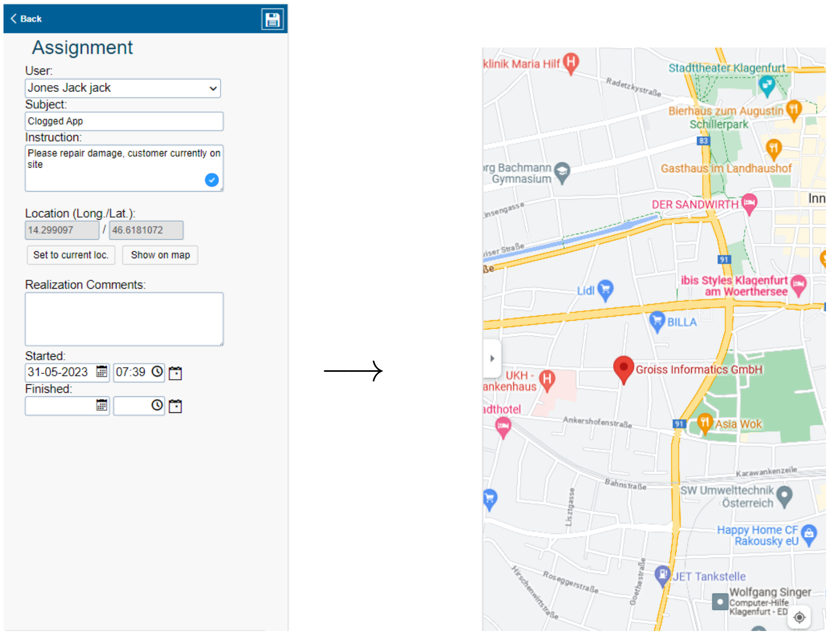
Worklist

3	Business trip Dr Walter Heisenberg, Vienna, 30-03-2023 - 01-04-2023	Enter expenses
52	Vacation Fritz Huber, 25-12-2023 - 29-12-2023	Request
51	SW Change No subject	Require
50	SW Change No subject	Require
49	SW Change No subject	Require
48	Change Requ... No subject	Insert Chan...
47	Incident Manag... No subject	Support
46	Incident Mana... No subject	Support
45	Incident Mana... No subject	Support
44	Problem Mgmt. No subject	Start
43	Job No subject	Create Job

Figure 3.11: @enterprise on a Smartphone

3.4 Social BPM

The previous section presented the integration of communication tools and the BPM application. Going a step further, we arrive at a general integration


Back

Assignment

User: Jones Jack jack

Subject: Clogged App

Instruction: Please repair damage, customer currently on site

Location (Long./Lat.): 14.299097 / 46.6181072

Set to current loc. Show on map

Realization Comments:

Started: 31-05-2023 07:39

Finished:

Figure 3.12: Form display and GPS integration

of collaboration and communication frameworks with BPM software. The resulting collective system is termed a *Social BPMS*.

Social software facilitates the communication between persons, the term subsumes a set of different systems like blogs, wikis, chats, forums, and social networks themselves. The central aspects are concerned with information from and about the other participants and provision of straightforward communication patterns.

Let us first deal with the information aspect from the focus of business process management. Collaboration in processes builds groups of people which communicate about and via those processes. Further groups are implied by the assignment to roles or organizational units. In a BPM the following information about users could be provided:

- ❑ User profile: via the organization tree browser or the process history, a profile about each user can be accessed. It contains basic contact data, current work items, location, ...
- ❑ Location in a Map: another view of locations could be provided in form of maps, especially suited for mobile work patterns.
- ❑ Availability and absence lists: can provide information about who is currently online.

The actual reason for the provisioning of such information is to enable communication between the participants. The following mechanisms could be provided:

- ❑ Conferencing: telephone conferences or audio/video conferences (Skype)
- ❑ Chat: short bursts of textual communication
- ❑ Wiki: creation and collaborative editing, extension and revision of (textual) information
- ❑ Blog: user created texts centered around a special topic, usually with sequential character. Normally not being revised but comments may be attached.
- ❑ Tagging: tags or keywords can be attached to documents and be used to quickly find and access the corresponding data. Tag clouds could be used for visualization of key word frequencies.

Not all of those communication patterns ("media") need to be integrated directly in the BPMS. The crucial point is that they can be seamlessly started and accessed directly from within the system (being "just a click away"). The forms of wikis, blogs and tagging are well suited for integration with the document management system of the BPMS.

The communication facilities can be used for collaborative work and problem solving during work on a process instance, but can also be used during the modeling phase. Especially in the case of a mobile working pattern, it enables more seamless interactions and can compensate some shortcomings of telecommuting [**?**].

3.5 Flexible Process Flow

The basic idea of workflow is the uniform execution of predefined processes, so that a set of business processes adheres to a prescribed flow or to prescribed rules. When there are rules, in practice there will be exceptional cases. It is not possible nor feasible to anticipate and model all exceptions. The arising complexity would not warrant the perceived completeness of the model. A much more desirable way of handling special cases would be the ability of the BPMS to provide appropriate flexibility functions for treatment of exceptions. In the following, we will present a set of typical exceptional situations, in figure 3.13 they are depicted.

3.5.1 Going Back

A common change in the process flow is to *go back* (cf. figure 3.13 a). The agent of a step recognizes, that for whatever reasons, he can not complete the step. He wants to send the process back to a previous step or a previous agent, e.g. allowing an applicant to clarify the reasoning for an application before it can get approved.

Since the normal execution of the process up to a point implies also the possibility for integration of system steps or the execution of post-conditions with arbitrary effects, a naive going back functionality could lead to undesired side effects. There must be the possibility to apply predefined compensation actions during going back to ensure a consistent process state. Going back several steps possibly means to execute a sequence of such reverting actions.

Parallel constructs do also need special attention during going back. When going back from a step within a parallel construct to a step before the parallel construct, the other parallel branches would have to be aborted. The abortion of steps – that is the elimination of steps from worklists – is a privileged action, a special permission *abort step* is needed to be able to go back when other parallel branches are still active.

Also going back into a parallel construct that has already been completed poses some complications. A main issue is how to treat other branches of the parallel construct; there is no general answer, @enterprise does not support going back into parallel constructs.

Going back could also be modeled explicitly by an appropriate loop construct. The following pattern can frequently be observed: in the first step of a sequence,

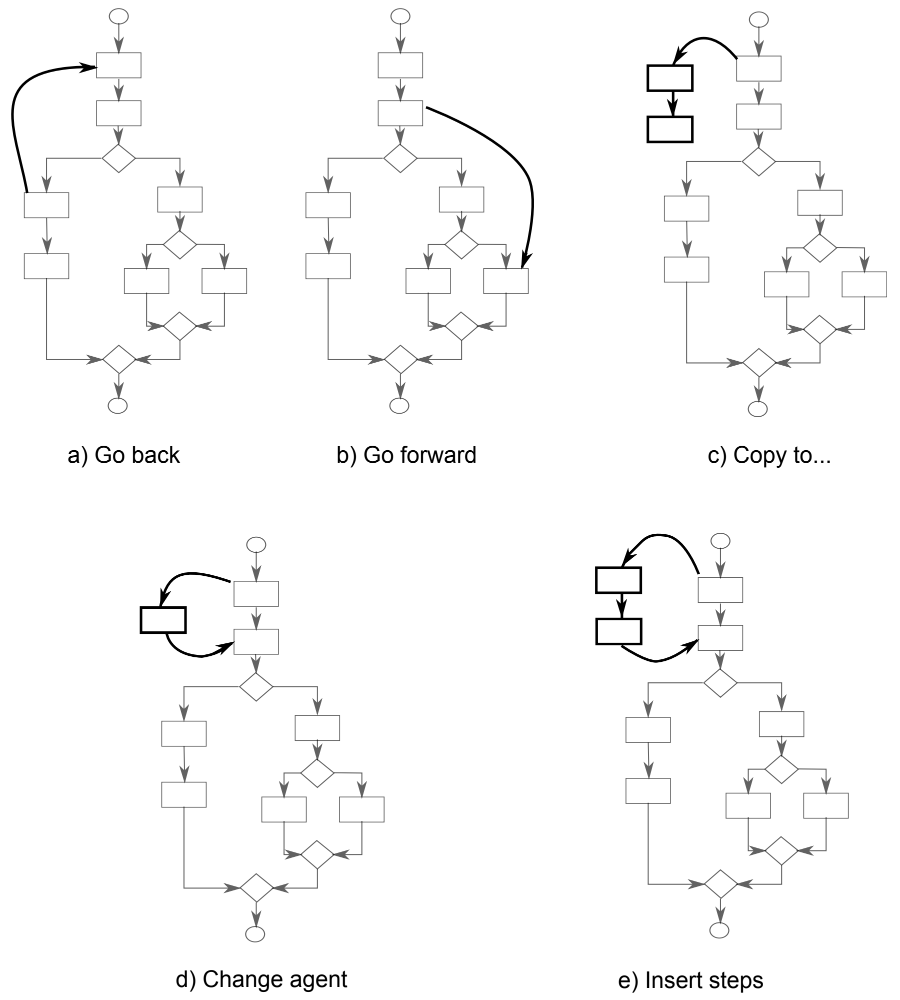


Figure 3.13: Deviations from Process Flow

a job is executed, a subsequent step is concerned with checking the result. A negative outcome would effect to recurrent execution of the first step. This would continue until the result is adequate.

Modeling this pattern with a loop makes it explicit. This enables to have a slightly different treatment for the correction step than for the initial step, e.g. to assign a shorter due date. Also standard reporting could explicitly differentiate between initial work and corrective actions. Even in a moderately complex process, the modeling and explicit addition of all possible ways to go back would lead to unduly complexity and rather cluttered and incomprehensible process definitions. Explicit modeling for going back is appropriate only when there is special treatment of the correctional step or when it is an integral or central process part.

3.5.2 Going Forward

Going forward (cf. figure 3.13 b) is even more problematic than going back: the process designer would need to define which steps could be skipped and which steps are mandatory. In @enterprise this functionality is not provided.

It must be mentioned, that going back like presented above is always meant to be to return to a previous step in the process structure and not according to the process history. Going back in the process history could mean to "go forward" in the process definition (e.g. when there are loops in the definition or when the "going back" function has been used earlier).

3.5.3 Copy to...

Sending a copy of the process data (see figure 3.13 c) is also an ad-hoc extension of the process structure. Another agent would receive a work item in her worklist with a read only view of the process data for reference or consulting purposes. This action is universally applicable.

3.5.4 Change Agent

When an item is routed to an agent A and this user is not in the position to work on the item (has no time, is not really competent, ...), he wants to hand off the item to another user. When the new agent completes the work item, the prescribed process path will be followed again (see figure 3.13 d).

This is a widely used feature, but one has to bear in mind, that in some processes there are firm restrictions about who is entitled to work on which items (e.g. an application must always be approved by a manager). Unconstrained agent changes would violate such process restrictions.

In @enterprise, this function can be forbidden for individual tasks. (this is a property of the task definition).

This function could also be implemented via explicitly extending the process model. When an agent change is likely to take place at a certain step in the process, a loop could be introduced where the agent assignment gets newly determined for each loop execution by reading it from a form field. The loop would be terminated via an explicit condition upon a second form field (see figure 2.28).

3.5.5 Insertion of Steps

Insertion of steps is a further common modification of the process flow (cf. figure 3.13 e). Those new steps are inserted after the current one and before the next one, as a kind of *detour*.

To maintain the integrity of the process, two issues must be considered:

Permissibility of Tasks

Only tasks that are related to the process may be inserted in it. That are tasks, that are already present in the ordinary process flow, but also additional tasks that were explicitly added to the process definition at build time by the designer as *ad-hoc tasks*. For all those tasks, the agent definitions and the form field visibilities are predetermined. But it still could be a problem that the order and position of those ad-hoc tasks are defined at run time. E.g. an additional *edit* step after an already finished *approve* step would lead to unapproved changes.

@enterprise allows to check for such conditions in the preprocessing. Insertion of ad-hoc steps can be used in unstructured or semistructured uncritical processes or could be restricted to tasks that can be used universally in the process (e.g. because their rather restricted form field visibilities).

Permissibility of Control Structures

Usually the insertion of steps is carried out by end users. Normally they can not be expected to have appropriate knowledge to correctly express conditionals

	PROJECT	PROCESS
Description	Project plan: individually created on general principles	Process Definition: common prescribed schema, individual adaptions
Control	Comparison of target and real progress, maintained by the project manager	Data from the process history and from forms
Forwarding	manual, informal communication	automatically by an engine

Figure 3.14: Project Management versus Process Management

or similar process components. Insertion of complex control structures should be avoided or forbidden in ad-hoc runtime modeling.

In @enterprise, the function "Send to" can be used to insert steps, it supports individual steps, sequences and parallel constructs.

3.5.6 Run-time Process Definition

Starting with a minimal process definition consisting of just a single task, the aforementioned mechanisms allow us to construct almost arbitrary complex processes at run time. This is essential when the individual process instances differ significantly from each other so that individual control flows are provided. There is no prescribed process definition, but rather an instance specific *plan* is created at process start time. Such processes are often called projects and the systems for their management are *project* management systems (and not *process* management systems).

Let us briefly elaborate on the differences between project management and process management (cf. figure 3.14):

When a BPMS allows to define ad-hoc processes and provides functionality for insertion of steps at run time, also projects could be executed and managed within a BPMS, thereby retaining the needed flexibility and also gaining the advantages of progress monitoring. In figure 3.15 a sketch of such a general project process is given.

There, the task *Manage* is being executed parallel to the individual project tasks in the *parfor* construct. The project manager (the agent of the *Manage* task) would gradually enter new project tasks, assign appropriate agents and would

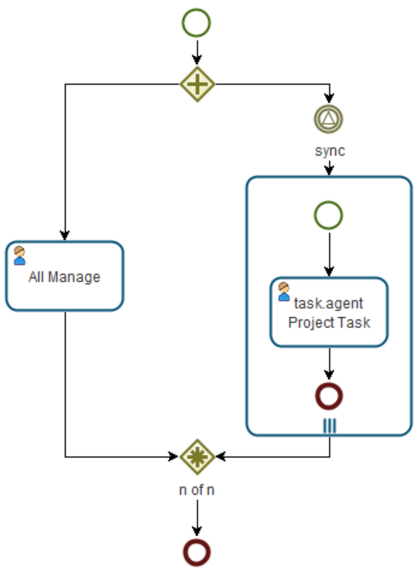


Figure 3.15: Project-Process

start the new tasks via a dedicated function. The tasks would be executed in parallel, but additional steps could be attached in sequence to each of them. This lines out that the execution of complex projects could be supported adequately and in a very flexible manner.

The described process is contained in the @enterprise demo application *Project management*. It can be downloaded from the web site of Groiss Informatics.

3.6 Permissions

A crucial component of a BPMS is the permission system. This is the subsystem that checks which user has access to which data in what form. Taking into account the sometimes very personal nature of the data used in the BPMS, it is quite clear, that there must be restrictions for access to e.g.:

❑ Processes that deal with payments or salaries or with termination of employment

- ❑ Search results, especially condensed ones that can allow to infer employee efficiency

The code of conduct according to such data can very much depend on the context. What is perfectly permissible in one country or company might be shunned upon or even be forbidden by legislation in another country or company.

In this section, the elements of the permission system will be introduced.

The principal means of permission granting or revocation is achieved via role assignment. This is known as *Role Based Access Control* (RBAC) [?]. Unfortunately, the power of this concept is not sufficient for our purposes, we need additional rules for a more precise description and definition of permissions. Such systems are known as *Attribute Based Access Control* (ABAC) [?].

A central differentiation when deciding about permissions is the separation of process independent permissions from process related permissions. A process related permission would be granted to a user when she participates in a process (e.g. has worked on one of its items in the past). We begin our discussion with the process independent permissions.

3.6.1 Proces-Independent Permissions

The schema depicted in figure 3.16 is the foundation of the administration of permissions. An agent is granted the permission to execute a right on a target object.

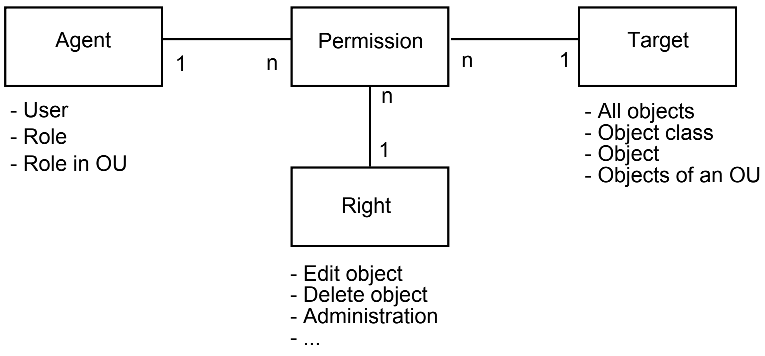


Figure 3.16: Permission Schema

- Agent: this can be a user, a role or a role within an organizational unit. The permission would apply to all users which were assigned this role (respectively who were assigned this role in the corresponding organizational unit).
- Right: this signifies the intended operation (e.g. view an object, edit an object, execute an object). It might be a right already predefined by the system or could be an application specific one.
- Target: targets can be individual objects, the permission would apply to this single object; targets could be an object class, the permission would apply to all instances of the class (e.g. to all reports). Universal applicability of a permission would be to define no target at all.

  There are two other forms of target set definition, based on the relationship of a target to an organizational unit. The permission scope *objects of an organizational unit* denotes either the objects of a specific organizational unit (the objects created by or owned by it), or the objects of the organizational unit in which the role was assigned to the user.

Examples for permissions are:

- The role *Sys* is permitted to execute all functions (right *execute* on object class *function*).
- The role *Manager* in the organizational unit *Controlling* may *view* all *processes*.
- Persons with role *clerk* may *view* such processes that where started in the department where the role has be assigned ("started in their department").

Negative Permissions:

Negative permissions (that is the explicit revocation of a permission by attaching a "false" attribute to it) can be used to concisely formulate exceptional cases. E.g. Mr. Heller may view all objects of department "Hexagon", except the object "Catch-22".

Rules: To avoid the explicit statement of who is permitted to operate on an object for each single (subordinate) object, we can define object class specific rules. For example, the edit permissions for the role assignments of a user are implied by having edit permissions on the user. The check for edit permission on the object "role assignment" is transformed to a check about permission to edit the user object of the role assignment. In @enterprise, such rules are formulated in Java.

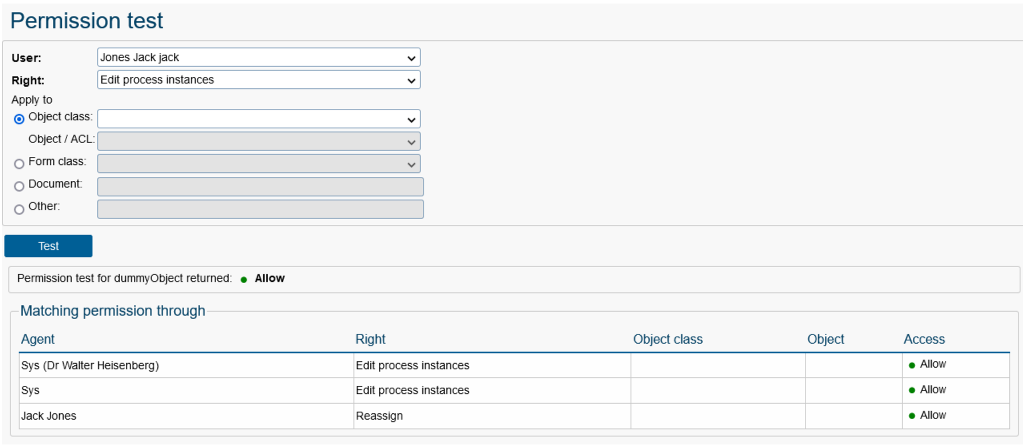
A prominent rule is the *Owner Rule*: when an object has an owner, the owner is automatically entitled to use the rights *edit*, *view* and *delete* on this object.

Permission Lists: allow easier maintenance of permissions according to the following two aspects:

1. Individual permissions can be grouped to *permission lists*. Such lists can be attached to several objects. Changes to the permission list would apply to all objects with this list.

2. For each object class, one permission list can be designated as the *standard permission list*. To each newly created object, this special list will be attached.

The administrative function "Permission Test" can be used to check whether a certain user has a particular permission (see figure 3.17. It can also provide insight, why the permission is being granted (on the basis of which entries in the permission system).


Permission test

User: Jones Jack jack
Right: Edit process instances
Apply to
[✓ Object class:]
Object / ACL:
Form class:
Document:
Other:

Permission test for dummyObject returned: ● Allow

Matching permission through

Agent	Right	Object class	Object	Access
Sys (Dr Walter Heisenberg)	Edit process instances			● Allow
Sys	Edit process instances			● Allow
Jack Jones	Reassign			● Allow

Figure 3.17: Permission Test

3.6.2 Process Related Permissions

Let us review the actions that can be applied to process instances and the related data:

1. Find process instances, view process history
2. view current process form data
3. view the process documents and their content
4. edit forms
5. edit documents, attach new documents, delete documents
6. standard functions in the worklist: give back, put into suspension, finish, go back, put into user folder
7. change agent of a process step
8. execute application specific functions on work items
9. abort and reactivate process instances
10. archive or delete process instances

Some of those actions are essential for a participant to be able to perform the task. Those actions should be available as default, without the need to explicitly assign permissions. The permission system contains a rule, that items 1 to 7 from the list above are permitted for the agent.

For functions of the application (item 8), explicit permissions are needed. Certain functions could be made available to only to some of the users.

A further rule of the permission system is concerned with previous process participants (those users, that worked on some activity of the process instance in the past). Those agents are permitted to find the process instance and to view the process history (item 1).

Some actions may be useful for non-participating users (persons that did not work on tasks of the process instance): finding processes (1), view some details (2,3), change current agents (7), abort, reactivate and archive the process (8,9,10). For those purposes, individual permissions for special rights must be assigned to the users. In @enterprise, the following rights for process instances are predefined and imply the corresponding actions:

- ❑ *view process instances*
- ❑ *change agent*
- ❑ *edit process instances*: abort and reactivate processes, change agent

- *configuration*: process archiving

Only in exceptional cases, permissions would be assigned for particular process instances. What makes sense in practice is to target all instances of a certain process definition or all instances started within a certain organizational unit. Accordingly, the four rights of the list above can have process definitions and organizational units as targets in a permission.

In the permission system, a rule would transform the question "Has user U the right *change agent* on process instance P?" into "Has user U the right *change agent* or the right *edit process instances* either on the organizational unit in which P was started or on the process definition of P?".

3.7 Substitutions

The definition of substitutions is needed to deal with absences of employees. For the period of absence of a person U, another person S would take over the work of U. User S is a substitute of user U. For S, in order to be able to work on the items assigned to U, there must be permissions set up for S. User S must get (at least part of) the permissions of U for the period of substitution.

Since a person has to fulfill several functions (has been assigned to several roles), substitutions must me definable on the basis of this roles. Some tasks of an absent department manager might be delegated to an assistant, other task categories require another department manager or even a superior as proper substitute. Substitutions for individual role assignments are called *role substitutions*.

For a substitute it must also be possible to act on tasks which were directly assigned to the substituted user. This is called *personal substitution*. Personal substitutions can optionally include all role substitutions, thereby minimizing maintenance.

It should be possible to define different substitutes for different periods of absence, e.g. for a two weeks leave, designate user S1 as substitute for the first week and user S2 as substitute for the second week.

Let us briefly summarize the possible functionality concerning substitutions:

- several personal substitutes, each with an optionally attached period of validity, including or excluding role substitutions;

- several role substitutes, each with an optional period of validity.

3.8 Interfaces and Application Integration

The execution of business processes is always paired with data manipulation operations. The data are linked to other data of the company (or the organization.). Since the BPMS is not the sole system that processes data, some form of data exchange has to take place.

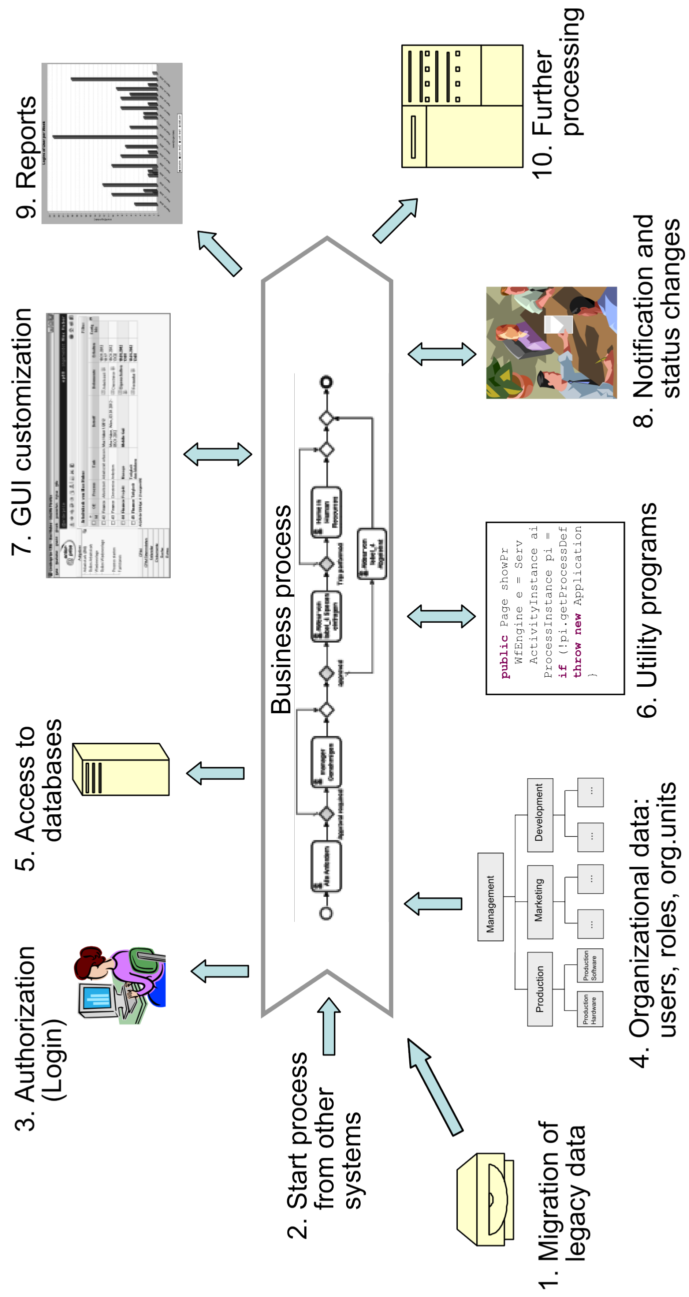
Let us look at an example concerned with applications for leave. The process itself will be rather simple, the process data will usually be one single form.

Is there even the need for integration in the case of such an almost trivial process? Typically, the answer is yes, and not to a small extend:

- The process is started via a web form which is integrated in an internal web site or a portal. This allows users to start processes without the need to explicitly log into the system and with minimal navigation effort.
- When the leave has been approved, some bookkeeping must be done against the holiday allowance in the human resource system.
- People directly involved (like team members, supervisors, substitutes) should get an informational e-mail.
- The periods of leaves should be aggregated and displayed department-wide (also considering other absences like business trips, sickness, etc.).
- A personal calender entry should be generated.
- As assistance for data entry, the amount of working days of the leave will be calculated automatically.

This rather basic example nevertheless allows to get an idea of the amount and depth of integration to ensure effective and comfortable usage of the process. Each BPM project is also a software development project and in particular an integration project.

In figure 3.18, the integration landscape is depicted. The BPMS (and the business processes) are in the center, surrounded by integration functions and components.



```
public Page showPr
  WfEngine e = Serv
    ActivityInstance ai
  ProcessInstance pi =
  if (!pi.getProcessDef
  throw new Application
}
```

Figure 3.18: Application Integration

1. Migration of legacy data: Often, the necessity to import instance data from a predecessor system arises.

2. External process start: Processes are started in an external system like e.g. triggered by a support request in form of an e-mail or by an incident message in an alarm system.

3. Authorization: The simple case of login to the workflow system is a system internal list of accounts and passwords. More comfort can be provided by checking the password against some centrally administered directory server. Additional convenience is achievable when the browser client already knows the credentials of the user and uses it to provide seamless login (Single Sign On, SSO).

4. Organizational data: The administration of users, roles, organizational departments and role assignments can also take place externally to the BPMS.

5. Database access: process relevant data may be in external systems (like e.g. an inventory system). Seamless access and provision of external data is crucial to achieve the optimization potential of the process. Manual transfer is painstaking and error prone, it should be avoided by all means.

6. Calling utility programs: the central metaphor for working on a process provided by the BPMS is the editing of forms in the browser. For effective operation, mechanisms like input assistance, in-place calculations, context sensitive selections and help texts are needed. But data is frequently also processed and worked on with external applications. Integration efforts must ensure direct access to those applications and seamless data transfer.

7. User Interface Customization: Ergonomic arrangements and dynamically provisioning of exactly those functions and information that the users require in the current context (process step).

8. Notifications: Users not working with the BPMS on a regular base must nevertheless be informed about new work items in a timely and appropriate manner (e-mail, SMS, paging).

9. Reports: for report creation, the integrated component of the BPMS can be used, it could also be necessary to provide instance data (run time data) for a third party reporting system.

10. Result processing: The instance data resulting from process execution are transmitted to other systems (e.g. a permit registration system for a building permit application process).

In the following sections, we will discuss these integration aspects. We will structure the presentation not along the items presented above but rather according to their technical implementation.

3.8.1 Organizational Data

The most significant standard for the administration of organizational data is LDAP (Lightweight Directory Access Protocol) [?], often in the form of AD (Active Directory) provided by Microsoft.

It is obviously advantageous to import the users and the organizational structure from such a directory server. In @enterprise an extensible interface for directory import is provided. One can define the access credentials and the search path for a directory server. The mapping between the data in the LDAP server and the @enterprise master data is accomplished via an easily adaptable API. A standard implementation for the import of users from Active Directory servers is provided.

Usually, the external directory servers do not provide all the information needed in the organizational data model of the BPMS, especially role assignments and permissions must often be administered in the BPMS directly. Another option is to extend the directory server schema and to provide the missing data there.

3.8.2 Authorization

The native authorization mechanism in @enterprise is via user id and password. The plain HTTP protocol sends the password unencrypted, but for extranet usage, encryption is strongly advisable. There is a special authorization class `SSLAuth`, that supports encryption of the password. Proper configuration of SSL is a prerequisite.

Passwords do pose some security risk, even when they are transmitted in encrypted form. A sticky note on the monitor frame or on the underside of the keyboard is often revealing.

When the Single-Sign-On package for windows is installed in @enterprise, it is possible to get along without a special @enterprise password.

Passwords could be completely abandoned using smart card technology. The authorization must be adapted to the API of the particular smart card framework. If the smart card client installs a browser certificate, no @enterprise client side code is required. An example is provided in the demo class `ClientCertDemoAuth`.

In a Single Sign On scenario, importing basic user data from an external system in inevitable.

3.8.3 Services

As mentioned before, one of the central aspects of a BPMS is the integration of different and diverse applications. The BPMS is used to coordinate (*orchestrate*) the calls to the other systems and to appropriately transform data.

The concept of construction of business processes on the basis of individual applications as building blocks (the *services*) is known as *Service Oriented Architecture* (SOA).

One of the main goals of service orientation is reuse. The call should take place in a platform independent manner. Web services are the most prominent set of standards in this area. They use standard protocols like HTTP(S) and XML based data formats.

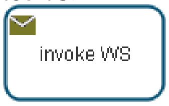
In a service oriented architecture, the BPM system itself can be seen as a service or set of services. Business processes could be started and BPM data could by accessed by other applications using web services.

The services provided are defined in the application specific master data of @enterprise. The description of the web services takes place via WSDL (Web Service Description Language).

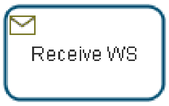
Web-Services in the Context of Processes

The process model of @enterprise provides specialized node types for utilization of web services. Each node of such a type references a registered web service. There are three types nodes for interaction with web services:

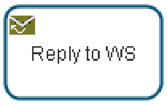
- **Invoke**: A web service is being called, the result is processed synchronously. In the BPMN this is represented as:



- **Receive**: The process waits for an incoming message. After the message has been received (the internal web service has been invoked by an external application), the process continues. If the first node of a process flow graph is a *Receive* node, then process start can be accomplished via web service calls.



- **Reply**: The *Reply* node is always a successor of a *Receive* node (but is optional itself) and can be used to send a result back to the service caller. Between receive and reply, system steps may be used to process the data received.



The data transfer between the web service formats and @enterprise forms can be done declaratively. Form fields are mapped to XML elements in the documents being sent and received via web services.

Calling functions of the BPM-Systems

The Wf-XML standard of the Workflow Management Coalition [?] defines a set of web services as standard means to access basic functionality of BPM systems. The provided web services of the standard are:

- Get a list of process definitions
- Start a process
- Query the state and data of process instances
- Change the state of activity instances

Additional web-services could be implemented and installed on the BPMS server.

3.8.4 Data Import and Export

Accessing external (*process relevant*) data is a common aspect of handling workflow tasks. E.g. when filling out an order form, data about the customer, articles or prices might be accessed.

The data could be stored either directly in the BPM system or an access to an external system must be done in one of the following forms:

- Periodic copying the data into the BPMS. In @enterprise a file import interface is provided. On this base, a frequent import of files exported from third party data bases can be done.

  As an alternative to file based data transport, the data could also be provided via a web-service.
- direct data access: For instance, a selection could be implemented such that the data is read from the external database and just the selected item is transferred to the BPMS.

To provide the BPM systems data to other systems, there are also several possible concepts:

- Access to the database: the external application could directly access the BPMS internal database tables in a read only manner. Appropriate schema adaption and access restriction could be implemented by means of SQL *views*.
- The reporting component can be used to create a report that is periodically being executed by a timer. The results could be made available as a file.
- Application coupling via web services: the BPM system could offer a web service to execute a report or to retrieve the data.

The decision which form to choose is dependent on the the permissible degree of coupling between the systems, on the general architecture of the partner system and on how current the data must be.

3.8.5 API-Programming in the BPMS

The workflow engine provides an application programming interface (API) to allow the implementation of functionality in a programming language. At numerous places in the process definition and the subordinate objects of the workflow applications, extension points (hooks) are provided to automatically call those functions (e.g. complex calculations, condition evaluations, agent assignments, preprocessing in tasks).

The following list provides an overview of of the locations where intervention via API is possible:

1. Process flow
   (a) Conditions: alternatives, loops, choices, generalized parallelism
   (b) Preprocessing: action before a task is put into the worklist
   (c) Postcondition: action at work item completion
   (d) System step: automated workflow step with program call
   (e) Agent assignment
2. Form handling:
   (a) Customization of form editing (input assistance, additional data display, ...)
   (b) Actions before insertion, update, deletion
3. Extension of the GUI via application functions
4. Customization of table display (worklist, DMS, ...)
5. Timer triggered actions: one shot actions or recurring ones
6. Actions at system start or shutdown

A common characteristic of all the hooks is that the API programs often have very little direct influence on the process flow. For the developers of API functions, this means that she has to design and implement a method or class that is being called by the system at the defined extension points. The result of the call is again interpreted by the BPMS. This concept is called *Inversion of Control*, see [**?**]. The control is in the hands of the BPM framework. The main advantage for application developers is that they are able to concentrate on core functionality and to largely ignore technical environmental aspects.

The @enterprise API is founded on this design principle. For the extensions above, one can implement a single method (e.g. for conditions, preprocessing, systems steps) or an interface consisting of several methods must be implemented (e.g. for customization of tables). Adapter classes with default implementations are provided for those interfaces, so that just the methods that must be adapted need to be implemented.

The following small method should illustrate the basic concept: in a preprocessing method, the due date of an activity instance should be read from a form field. The Java method to implement this is:

```
public void setDuedate() {
   WfEngine wfe = ServiceLocator.getWfEngine();
   ActivityInstance ai = wfe.getContext();
   ProcessInstance pi = ai.getParent();
   DMSForm f = wfe.getForm(pi, "item");
   Date dt = f.getField("duedate");
   if (dt != null)
      wfe.setDuedate(ai, dt);
}
```

The method `setDuedate` first gets its context – the current activity instance `ai`, the corresponding process instance `pi`, the form `f` and the value `dt` of the form field `duedate`.

The engine function `setDuedate` is used to set the due date for the activity instance.

The context (current process, form instances) is usually passed to the API functions directly via parameters, or can be accessed from initial navigation roots by special calls like `wfEngine.getContext()`.

The API programs can execute arbitrary actions, e.g. access databases or other external systems, send e-mails etc. A transaction boundary is provided by the BPMS, changes do get committed automatically, errors lead to rollbacks and do not lead to inconsistent states.

A detailed elaboration of the @enterprise API can be found in the Application Development Guide and the API documentation. Both can be accessed via the Groiss Informatics web site (*https://www.groiss.com*).

3.8.6 Application Integration at the Client

Client side application integration is needed and sensible, when the application to be integrated has an own or proprietary user interface, e.g. the creation and processing of documents with office software suites. A pure web client is often rather restricted, for better integration of local applications and hardware (e.g.

document scanners), additional client installed software is needed. @enterprise offers a Java based client for such client side integration purposes.

The integration aspects for mobile clients have already been discussed in section 3.3.6.

3.8.7 Security Aspects

In a BPM applications several diverse persons, departments and systems work together; this imposes additional complexity for the architecture of overall application security.

External access

A central question concerning security is the location of the access points to the system. Are the accesses from a tightly controlled dedicated local network, are they from within a controlled intranet of the organization or are they originating from the Internet?

When Internet access is possible, the system must be adequately secured. There will be a non-zero probability of attack. A mere authorization with user id and password will hardly be sufficient.

Access to system administration should be restricted to dedicated clients or networks; in @enterprise, an *URLChecker* can be configured, see the @enterprise Installation Manual [**?**], section *Access Control*.

Authorization

The applications which will access the BPMS must also be subject to authorization like real users, see section 3.8.2.

Security in API Calls

The integration with external systems deserves additional thought.

Accesses from the BPMS to other systems are a rather simple case. The other system has to provide some form of technical access. If the external system has security or privacy relevant data (like the user data of a directory server), then this reliance of trust to an external system may become problematic. There may also be the need to confirm the identity and authenticity of the external system (e.g. with certificates).

When external systems are calling the BPMS, the access must be properly contained. Web services should be specifically restricted to a well defined set of operations. Also for web services authorization could be required; certificates could be put to use in this case, too.

System Environment

While this topic is not specific for BPM applications, we would like to point out that the security of the system environment is a major component and building block for the security of the overall system. Some issues are:

- securing access to the data base
- securing network access, physical access to the network components
- securing the servers, physical access, basic system accounts, etc.

3.8.8 Further Aspects of Application Integration

Besides the functional requirements and the security requirements, there are a lot of further aspects for the integration of applications:

Transactions: Proper error handling: some escalation concept and mechanism must be devised to at least inform the process owners or support personnel about errors. Procedures for diagnosis and remediation of such errors must be developed. This is of utmost importance when a change in the BPMS triggers a change in an external system and the assumption is that both systems are in perfect synchrony.

Service unavailability: Such interruptions will occur in practice and must be accounted for. The solutions could include escalation mechanisms or temporal queuing and replay capabilities.

Logging: Traceability of accesses from external application to the BPMS. Such operations should be logged in sufficient level of detail to allow for issue diagnosis. Depending on the particular requirements, data changes should be logged, sometimes a simple flat file journal might be sufficient, sometimes a sophisticated persistent logging in the database might be needed.

3.9 Customization of the @enterprise User Interface

A BPMS is a largely domain neutral framework which is suitable for rather diverse kinds of processes and which provides general functions for process execution. To allow for efficient process handling in a particular application or domain, the user interface will have to be adapted. It will provide a subset of the general functions, removing the unneeded capabilities and enrich this set with additional functions and views specifically suited and tailored for the application. We will briefly present the adaptability features of the @enterprise user interface.

3.9.1 GUI Configurations

A *GUI configuration* object in @enterprise allows to create a customized appearance and set of functions for specific roles. Sometimes, just a singular configuration is sufficient, but when there are different user groups with rather distinct needs, several GUI configurations must be provided. For each GUI configuration, one can state for which roles or users this configuration is applicable. A priority can be used to disambiguate the selection in the case of multiple assignments.

Example:

Agent	configuration	Priority
Role `all`	`standard`	10
Role `manager`	`extended`	20

Everybody who has been assigned the role `all` will get the `standard` configuration. Users in the role `manager` will be getting the configuration `extended`, irrespective of potential assignments of role `all`.

A configuration defines the GUI, a set of navigation links that lead to the individual functions. Dependent on the type of the link, several properties can be modified:

Text: a text in the navigation.

Link: arbitrary link or a link to a basic function of @enterprise.

Worklist: set of columns, type of worklist (personal, role, suspension), and toolbar functions.

Structured Worklist: root for user defined folders; modifications like for worklists.

Process start: link to the list of startable processes or for the start mask of a particular process.

Functions: link to the list of task independent functions.

Reports: list of all reports or execution of a particular report.

DMS: root folders of the document management system. Columns and toolbar functions can be adapted.

Form Table: table of a form type, the columns of the table and the toolbar functions can be adapted.

In figure. 3.19, an example of such a adapted navigation is presented. Modifications with respect to the default configuration are: the removal of the role suspension worklist, the addition of the function "Time recording" and the introduction of a search field.

3.9.2 Adaption of Styles

The preferences for colors and fonts are rather individual, @enterprise provides a style configurator to allow the setting of such preferences by the end users themselves.

On the other hand, centrally administered styles may be needed to achieve a proper branding or conformance with a corporate style. For this purposes, no graphical interface is provided, a style sheet must be edited.

3.9.3 Internationalization

BPMS are being used in large multinational organizations and corporations, where the users can be expected to speak different native languages.

One way of dealing with such a situation would be to develop the whole user interface in just one preferred language (the corporate language). Often this is not desirable for reasons of employees with lacking fluency in this language or for reasons of a specific diversity policy. Then the application has to be multilingual.

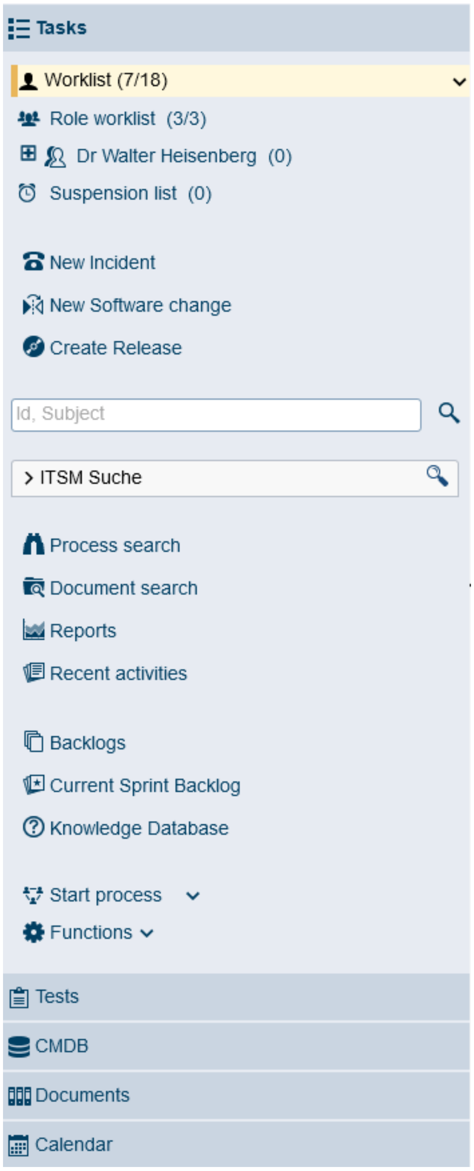
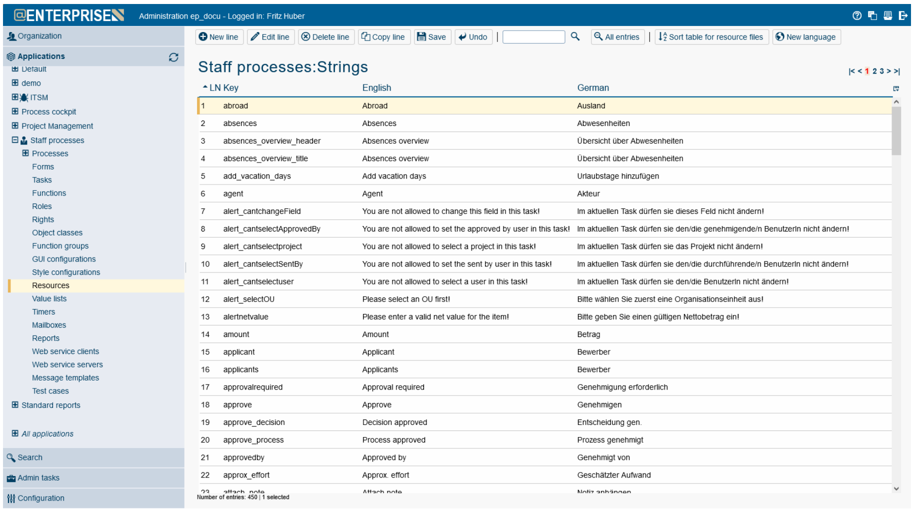


Figure 3.19: Adapted Navigation

All names and texts which will appear in the GUI, all descriptions and help texts in forms and all process components must be translated to all desired languages.

@enterprise offers some support for this aspect. Already in the modeling phase the names of the process elements can be entered in a multilingual way. For the names of processes, tasks, forms etc., in the mask we specify not a name but the *key* for the name. In additional fields, the translations for the languages to be used can be entered. A navigation link "Resources" provides access to an editor for the maintenance of the keys and their language specific values, cf. figure 3.20.



Staff processes:Strings

LN	Key	English	German
1	abroad	Abroad	Ausland
2	absences	Absences	Abwesenheiten
3	absences_overview_header	Absences overview	Übersicht über Abwesenheiten
4	absences_overview_title	Absences overview	Übersicht über Abwesenheiten
5	add_vacation_days	Add vacation days	Urlaubstage hinzufügen
6	agent	Agent	Akteur
7	alert_cantchangeField	You are not allowed to change this field in this task!	Im aktuellen Task dürfen sie dieses Feld nicht ändern!
8	alert_cantselectApprovedBy	You are not allowed to set the approved by user in this task!	Im aktuellen Task dürfen sie den/die genehmigende/n BenutzerIn nicht ändern!
9	alert_cantselectproject	You are not allowed to select a project in this task!	Im aktuellen Task dürfen sie das Projekt nicht ändern!
10	alert_cantselectSentBy	You are not allowed to set the sent by user in this task!	Im aktuellen Task dürfen sie den/die durchführende/n BenutzerIn nicht ändern!
11	alert_cantselectuser	You are not allowed to select a user in this task!	Im aktuellen Task dürfen sie den/die BenutzerIn nicht ändern!
12	alert_selectOU	Please select an OU first!	Bitte wählen Sie zuerst eine Organisationseinheit aus!
13	alertnetvalue	Please enter a valid net value for the item!	Bitte geben Sie einen gültigen Nettobetrag ein!
14	amount	Amount	Betrag
15	applicant	Applicant	Bewerber
16	applicants	Applicants	Bewerber
17	approvalrequired	Approval required	Genehmigung erforderlich
18	approve	Approve	Genehmigen
19	approve_decision	Decision approved	Entscheidung gen.
20	approve_process	Process approved	Prozess genehmigt
21	approvedby	Approved by	Genehmigt von
22	approx_effort	Approx. effort	Geschätzter Aufwand
23	attach_note	Attach note	Notiz anhängen

Number of entries: 450 | 1 selected

Figure 3.20: Editor for Internationalization

3.10 Playing through a Process

In the modeling chapter we were concerned with the definition of processes. This chapter has been dealing with run time aspects. We are now in the position to actually run process instances. We will start it on the system where the modeling took place. Deployment to a production system is being treated in section 5.6.

In order to be able to execute a process it must be sufficiently defined. For each interactive step there must be an agent description and a task id provided. Each condition must contain a syntactically correct expression and each call in a system step must be technically sound. When those prerequisites are fulfilled, the process definition can be set to active and instances can be created.

All roles that are being used in the process must be assigned to at least one user. The action for process start is found in the navigation below the worklists, see figure 3.7. After start of the process, the work item of the newly created process will appear in the personal worklist.

3.11 Elements of a BPM Application

Summarily, the elements of BPM applications are:

- ❑ Process definitions: executable process specifications
- ❑ Tasks: interactive steps of a process with preprocessing method, post-condition, etc.
- ❑ Form types: describe an entity for persistent data storage, the representation in the GUI and the types of the fields
- ❑ Functions: procedures that support the process execution in some manner
- ❑ Roles: to define participants in the processes
- ❑ Permissions: to restrict access to data and functions
- ❑ Resources: for internationalization purposes, contain translations for labels
- ❑ Web-Service Servers and Clients: definitions of called and offered web services for application integration
- ❑ Reports: data analysis and condensation
- ❑ Timer: for recurrent, automatic action execution

Besides those elements (all of which are stored in the data base), the implementation will consist of program code and further artifacts like HTML pages, icons and help texts.

Chapter 4

Monitoring and Optimization

As stated in the introductory chapter, the process management cycle also comprises the monitoring of the current activities and the optimization of the processes. Starting point for the optimization efforts are the process definitions and the run time data of the process instances.

4.1 Run Time Data Analysis

During execution of the processes, the BPMS generates a wealth of data that can be put to use for process analysis and optimization.

From the data of the process instances and the activity instances, information like the following can be gained:

- What is the content of the worklist of a particular agent?
- How long are the worklists of the employees in my department?
- How many processes of a particular type are currently active?

Such data base queries or *reports* can be executed regularly or even periodically, to get some insight about the current state of the system. Using a set of such standardized reports, essential conclusions about performance can be drawn. There could also be specialized reports for particular process types, taking into account the specific process structure and data.

In the following sections the mechanisms for dealing with the run time data are described: the reporting component for data extraction, the dashboard for an overview of the most essential information and the process cockpit as an overall view.

4.1.1 Reporting

A reporting component is an integral part of a BPMS, it enables the creation of reports over all the information in the data base. Further examples for reports (besides the above mentioned run time centric ones) are:

- ❑ List of unsettled invoices and overall amount of outstanding payments (if the processing of invoices is done within a process)
- ❑ Number of open incidents per product.
- ❑ Monthly trend of issues reported and resolved for a product.

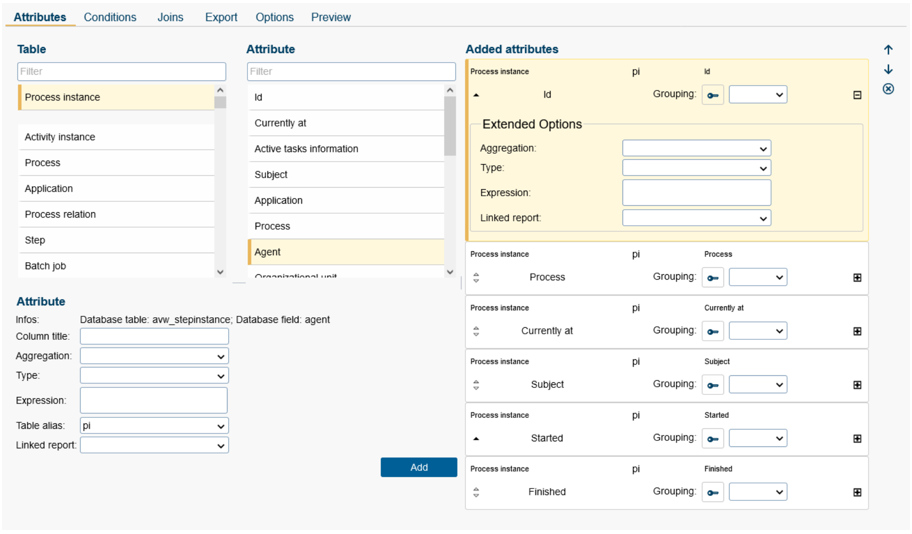
The ability to combine data from the process execution run time and from application data is a crucial one.

Report design is a complex task, knowledge of the data and their interrelations is needed. Furthermore, the report designer is entitled to access all the data. The number of people with this level of access should be minimized. After such reports have been designed, execution permission for them can be granted to members of roles. The execution of the reports will be done in real time on the live data. Starting the report triggers extraction of data from the data base, calculation of the results and presentation of them.

Figures 4.1 and 4.2 show some masks for report design in @enterprise.

The elements of a report can be defined step by step. Those elements are:

- ❑ The display attributes: which data is presented in the result.
- ❑ Conditions: specifying restrictions to select the proper set of results.
- ❑ Parameters for conditions: conditions can be parametrized. The designer creates the principal abstract condition, the end user states the concrete value for each execution of the report. E.g. the report designer would create a condition that restricts the report results to processes started in a certain (but unspecified) year, which the end user would fill in.



Attribute

Infos: Database table: avw_stepinstance; Database field: agent
Column title:
Aggregation:
Type:
Expression:
Table alias: pi
Linked report:

Process instance	pi	Id	
Id		Grouping:	
Process instance	pi	Process	
Process		Grouping:	
Process instance	pi	Currently at	
Currently at		Grouping:	
Process instance	pi	Subject	
Subject		Grouping:	
Process instance	pi	Started	
Started		Grouping:	
Process instance	pi	Finished	
Finished		Grouping:	

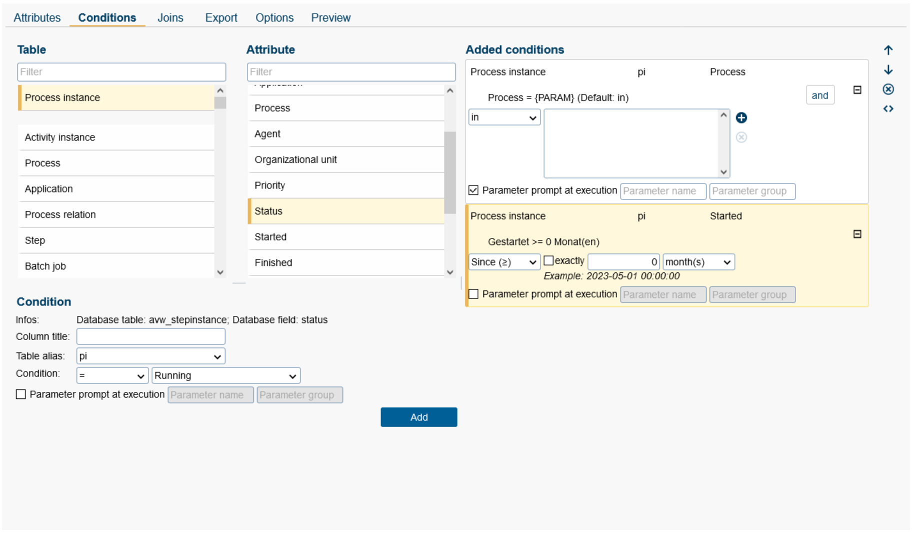
Extended Options: Aggregation:, Type:, Expression:, Linked report:

Figure 4.1: Report Designer - Attributes

- ❑ Implicit parameters: are used to limit the result set depending on the execution context. The context contains the executing user, her home organizational department and the execution timestamp. So, a report for the processes of the current month in the department of the report user can be created.

- ❑ Aggregations: one of the central features of reports is data condensation: counting of non-numerical indicators like number of processes or documents, etc.; summing up, calculations of averages, minimal and maximal values of numerical data. The aggregations can take place on several levels, e.g. the first level counts the processes per month, in the second level the months with the most and least process instances are determined.

- ❑ Presentation: the simplest form of a result is a table, usually rendered directly in the web browser. For printing or archiving, PDF (Portable Document Format) is normally used. For further processing in other systems, the CSV format (Comma Separated Values) might be used. The MS-Excel format is also directly supported. Business graphics, like bar charts, line charts and pie charts are provided for better visualization.



Table

Filter

- Process instance
- Activity instance
- Process
- Application
- Process relation
- Step
- Batch job

Attribute

Filter

- Process
- Agent
- Organizational unit
- Priority
- Status
- Started
- Finished

Added conditions

Process instance pi Process

Process = {PARAM} (Default: in)

in

☑ Parameter prompt at execution Parameter name Parameter group

Process instance pi Started

Gestartet >= 0 Monat(en)

Since (≥) ☐ exactly 0 month(s)

Example: 2023-05-01 00:00:00

☐ Parameter prompt at execution Parameter name Parameter group

Condition

Infos:	Database table: avw_stepinstance; Database field: status
Column title:	
Table alias:	pi
Condition:	= Running

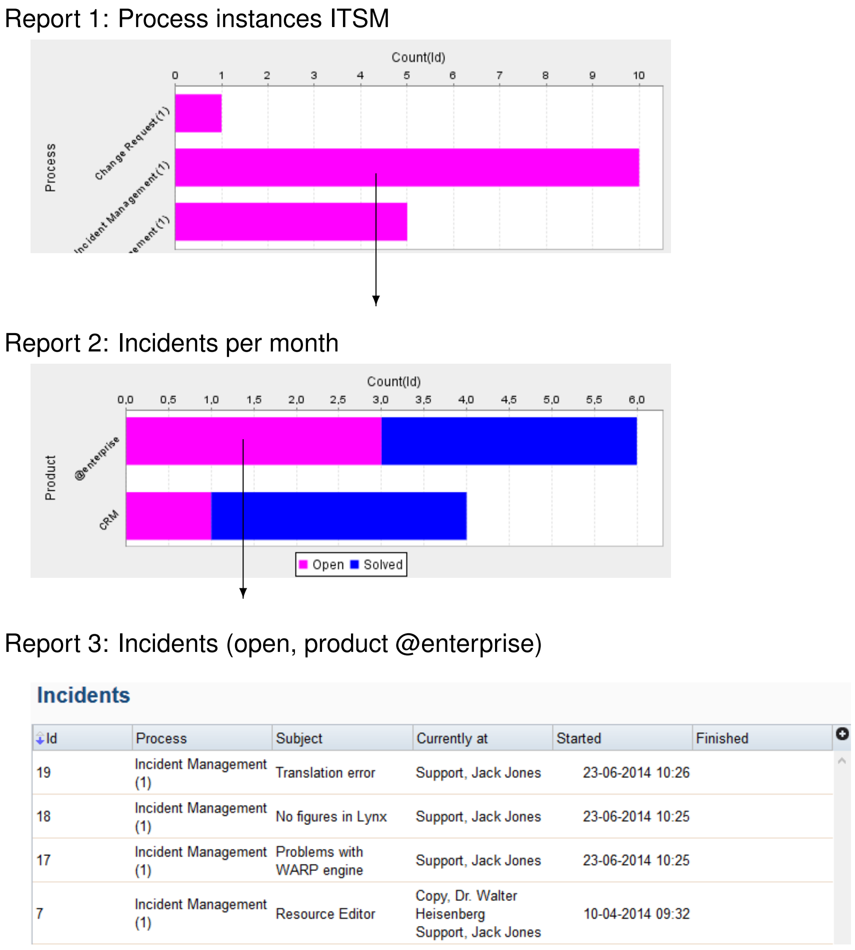
☐ Parameter prompt at execution Parameter name Parameter group

Figure 4.2: Report Designer - Conditions

- ❏ Links between reports: greater levels of detail could be reached via links and stepwise drill down capabilities.

An example for linked reports is depicted in figure 4.3. A general report shows the number of process instances per process type. Clicking on a bar could lead to a second report that contains the instances of the selected process definition in the current month, grouped by products and with the categories *open* and *solved*. To see the open or solved issues of a certain product, a click on the corresponding bar area could get the detailed list of the corresponding process instances. Those provide navigation links to their form data, documents and histories.

In the user interface of the BPMS, a link for the list of executable reports will be placed in a suitable location, but reports can be also be placed in other locations as well.



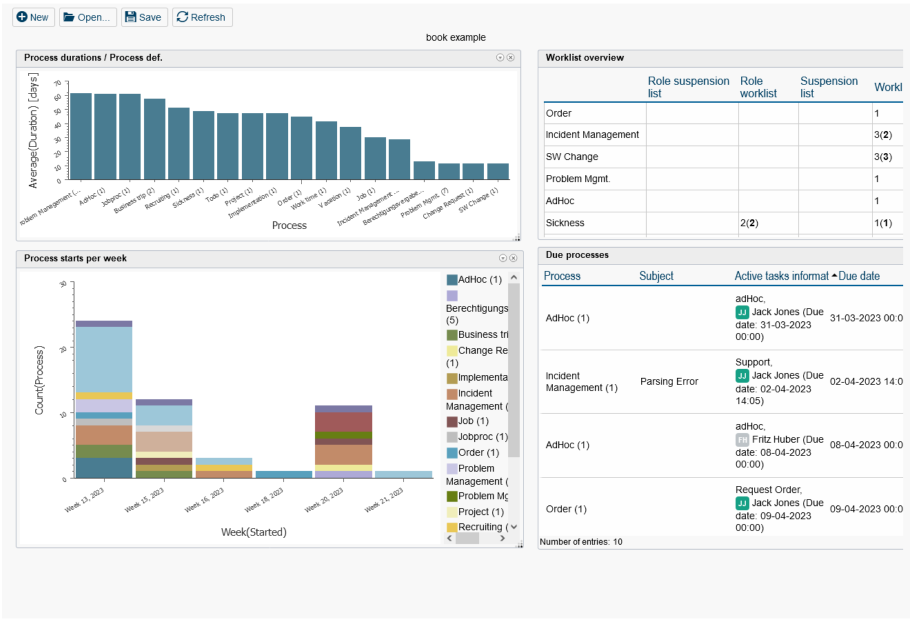
Incidents

Id	Process	Subject	Currently at	Started	Finished
19	Incident Management (1)	Translation error	Support, Jack Jones	23-06-2014 10:26	
18	Incident Management (1)	No figures in Lynx	Support, Jack Jones	23-06-2014 10:25	
17	Incident Management (1)	Problems with WARP engine	Support, Jack Jones	23-06-2014 10:25	
7	Incident Management (1)	Resource Editor	Copy, Dr. Walter Heisenberg Support, Jack Jones	10-04-2014 09:32	

Figure 4.3: Drill-down Reports

4.1.2 Dashboard

The dashboard gives a personalized overview about the system state. Each user can individually compose and arrange the most crucial information for her on this page. This can be reports, a worklist overview (with numbers of new/due/all entries), appointments, current system users, etc. Figure 4.4 shows an example of such a dashboard in @enterprise.



Worklist overview

	Role suspension list	Role worklist	Suspension list	Workl
Order				1
Incident Management				3(**2**)
SW Change				3(**3**)
Problem Mgmt.				1
AdHoc				1
Sickness		2(**2**)		1(**1**)

Due processes

Process	Subject	Active tasks informat	Due date
AdHoc (1)		adHoc, JJ Jack Jones (Due date: 31-03-2023 00:00)	31-03-2023 00:0
Incident Management (1)	Parsing Error	Support, JJ Jack Jones (Due date: 02-04-2023 14:05)	02-04-2023 14:0
AdHoc (1)		adHoc, FH Fritz Huber (Due date: 08-04-2023 00:00)	08-04-2023 00:0
Order (1)		Request Order, JJ Jack Jones (Due date: 09-04-2023 00:00)	09-04-2023 00:0

Number of entries: 10

Figure 4.4: Dashboard

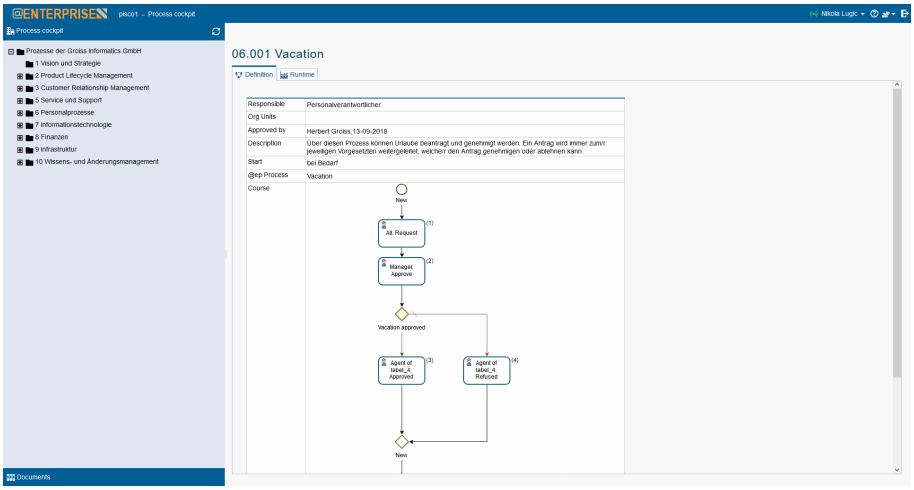
The dashboard can also be used as the default start page. It would then be displayed immediately after logging in.

4.1.3 Process Cockpit

While reports are well suited for obtaining various detail information and the dashboard is fine for gaining a quick overview, the need for a comprehensive view of all processes calls for yet another form of presentation. In chapter 1 we introduced the process handbook as a collection of all documented processes in the whole organization.

The *Process Cockpit* is an extended form of this handbook. Not only process definitions but also current run time information of the process instances is readily accessible.

Figure 4.5 shows the start page of a process cockpit, where a hierarchical representation of all the processes of a corporation can be found. The structure



Prozesse der Groiss Informatics GmbH
- 1 Vision und Strategie
- 2 Product Lifecycle Management
- 3 Customer Relationship Management
- 5 Service und Support
- 6 Personalprozesse
- 7 Informationstechnologie
- 8 Finanzen
- 9 Infrastruktur
- 10 Wissens- und Änderungsmanagement

06.001 Vacation

Responsible	Personalverantwortlicher
Org.Units	
Approved by	Herbert Groiss,13-09-2018
Description	Über diesen Prozess können Urlaube beantragt und genehmigt werden. Ein Antrag wird immer zum/r jeweiligen Vorgesetzten weitergeleitet, welche/r den Antrag genehmigen oder ablehnen kann.
Start	bei Bedarf
@ep Process	Vacation
Course	

Figure 4.5: Process Cockpit, Start Page

is configurable, in the example, it follows the Process Classification Framework of the APQC (American Productivity & Quality Center) [?]. For each process, a varying degree of detail is available, depending on the completeness of the implementation. There are:

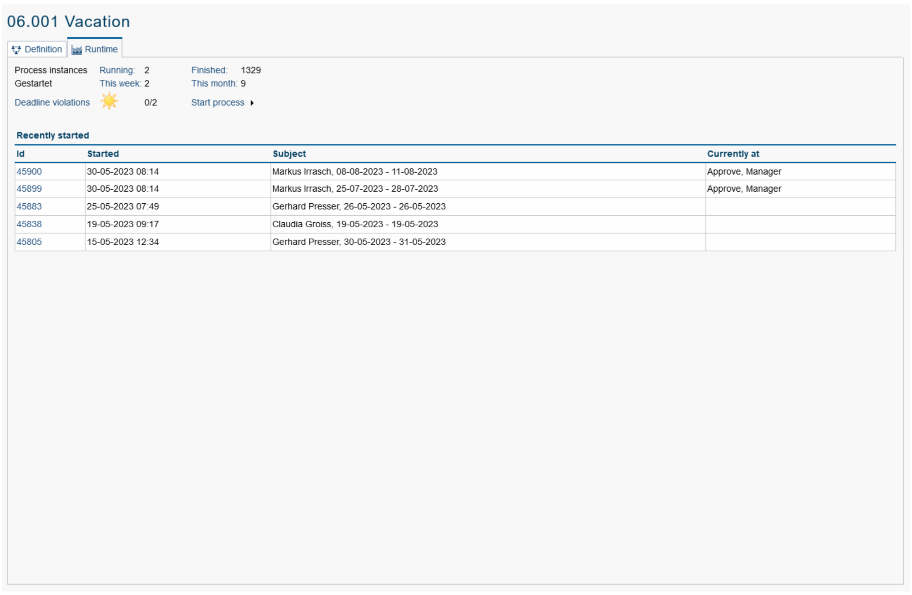
1. processes with just a textual description; there is no corresponding workflow process;
2. processes which are indeed described via a process definition, but that are not executed within the BPMS
3. and processes that are properly defined and also executed within the BPMS.

Accordingly, the process cockpit contains the following information:

- ❑ documents that are somewhat connected to the process, textual process descriptions, execution guidelines and help texts. This type of information is available for all kinds of processes.
- ❑ a representation of the process description for the processes in groups 2 and 3.

❑ for processes in group 3, also run time information is accessible.

The process cockpit view of one process type is shown in figure 4.6. The information about the process definition is grouped in two tabs. The view is configurable per process definition. One can select the reports do be executed immediately, the reports to be accessible via links, additional functions and additional arbitrary links.



Process instances Gestartet	Running: 2 This week: 2	Finished: 1329 This month: 9	
Deadline violations	0/2	Start process	

Recently started

Id	Started	Subject	Currently at
45900	30-05-2023 08:14	Markus Irrasch, 08-08-2023 - 11-08-2023	Approve, Manager
45899	30-05-2023 08:14	Markus Irrasch, 25-07-2023 - 28-07-2023	Approve, Manager
45883	25-05-2023 07:49	Gerhard Presser, 26-05-2023 - 26-05-2023	
45838	19-05-2023 09:17	Claudia Groiss, 19-05-2023 - 19-05-2023	
45805	15-05-2023 12:34	Gerhard Presser, 30-05-2023 - 31-05-2023	

Figure 4.6: Process Cockpit, View for one Process Definition

The displayed information is subject to the permission system. Process owners are allowed to see all detail information for the process instances, mere process participants can view those instances they worked on, non-participating users may view general reports e.g. with some aggregated key performance indicators.

4.2 Process Optimization

This section discusses one of the main goals of business process management and the deployment of BPMS, namely to increase the efficiency of business processes. While this section is of utmost interest to many readers, it is neither by chance, nor by bad planning that we are dealing with optimization rather late. Process definition, handling within a BPMS and analysis of run time data are prerequisites for process optimization.

On the basis of the detailed information from reporting, dashboard and process cockpit components, the process optimization can start. The quantitative data collected in the BPMS are certainly just one facet of the picture and are not sufficient for a general optimization. Other essential information would e.g. be customer satisfaction or product quality. Such qualitative data must be collected, too.

There are several different directions to pursue process optimization:

- Speed: the goal is to achieve short process turnaround times. The focus can be on average time or can be to avoid extremal values as much as possible.
- Quality: an optimal result shall be achieved. Errors and corrective iterations should be avoided by all means (e.g. when building a product).
- Resource Utilization: the resources should be utilized in an optimal way to minimize process cost.

The goals are at least partially in conflict. It is not possible to enhance a process with respect to all the criteria, better quality may imply longer throughput times. Less probable timeouts would usually mean less resource utilization. It is the task of the management to postulate the proper goal mix for the particular situation.

According to the focus of this book, we do not elaborate on general process optimization, but on the optimization closely related to the BPMS. We distinguish three aspects:

- better resource utilization,
- optimized process flow,
- optimization of the activities.

Further (untreated) facets would be organizational aspects, activity handling outside of the BPMS, goal definition and goal balancing.

The optimizations can be performed at three levels (cf. [?]), like depicted in figure 4.7:

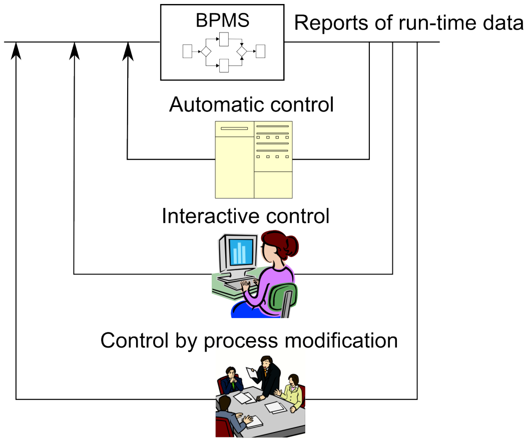


Figure 4.7: Optimization Cycles

- ❑ Within the BPMS: The system assigns tasks and orders them without manual intervention.
- ❑ Operative Management: The processes are being observed and some manual intervention takes place.
- ❑ Strategic Management: No intervention at process instance level, rather the process definitions, the tasks and activities or the resource allocation are targeted.

The levels are different with respect to their time horizon, the effort to implement them and their sustainable effectiveness.

4.2.1 Optimization within the BPMS

When bottlenecks arise during the execution of business processes, some mechanisms in the BPM system can contribute to their resolution.

Since the resources provided can not be augmented by the BPMS, just reassignment of work is possible. From each of the running processes, two indicators are considered:

- ❑ process priority
- ❑ probability of exceeding a deadline

Both numbers can be combined (e.g. by simple multiplication) to compute a processes urgency.

The second criterion for reassignment of activities is the load of the resources. The load factor could be approximated on the basis of the number items in the worklists, balanced with the individual (projected) effort needed.

A scheduling component could implement the following functions:

- ❑ scheduling of new activity instances: they are assigned to the agents with the least load (the *take* function is executed automatically on the behalf of the respective users).
- ❑ rescheduling of existing activity instances: items in personal worklists could be reassigned to other users (provided that the work on those items has not yet started).
- ❑ Deadline calculation: determination of process deadlines could be dependent on current workloads and resource utilization.

Assuming that activities in the worklists have priorities or levels of urgency attached to them, the question arises whether to allow users to decide themselves about the next task they are to carry out. Such freedom would thwart the efforts of automatic scheduling, but is nevertheless the approach that gets usually implemented. A system that would force the users to work on exactly the item that the system prescribes would lead to acceptance problems. Normally, user satisfaction, efficiency and effectiveness is enhanced by some degree of freedom and self determination of the work.

4.2.2 Optimization by Operative Management

The operative management can act upon the information provided via the process cockpit and the performance reports. Possible actions on this layer are:

- ❑ Manual assignment: explicit assignment or reallocation of work items to particular users.
- ❑ Increasing capacities: bottlenecks could be eliminated by resource reallocation. Lightly loaded users could temporarily be assigned to roles of heavy loaded users. Working hours could be increased.
- ❑ Deadline adjustment: The current process instances are analyzed. Maybe deadlines can be extended or priorities can be reconsidered without breaching any contracts or customer agreements.
- ❑ Deadline calculation for new processes: Customers can be informed about expected delays as soon as possible.

Usually the process owner is responsible for evaluating, deciding and executing such manual interventions.

4.2.3 Optimization by Strategic Management

In the outermost feedback loop in figure 4.7, there are two possible approaches to enhance the performance:

- ❑ Requisition of additional resources: when all potentials for internal resource shifting are exploited and the processes have to be carried out in the current manner, additional resources must be acquired. This could be on a seasonal basis and would require some planning and lead time.
- ❑ Process optimization: finally, the most essential and high-yielding optimization; the enhancement of the processes. On one hand, the process structure can be enhanced, on the other hand, the individual steps could be streamlined.

Resource control is not discussed here, in the following section we will concentrate on process optimization as one of the main tasks of corporate management.

4.2.4 Changing the Process Structure

An overview of the principal variants of changes to the process structure is given in figure 4.8.

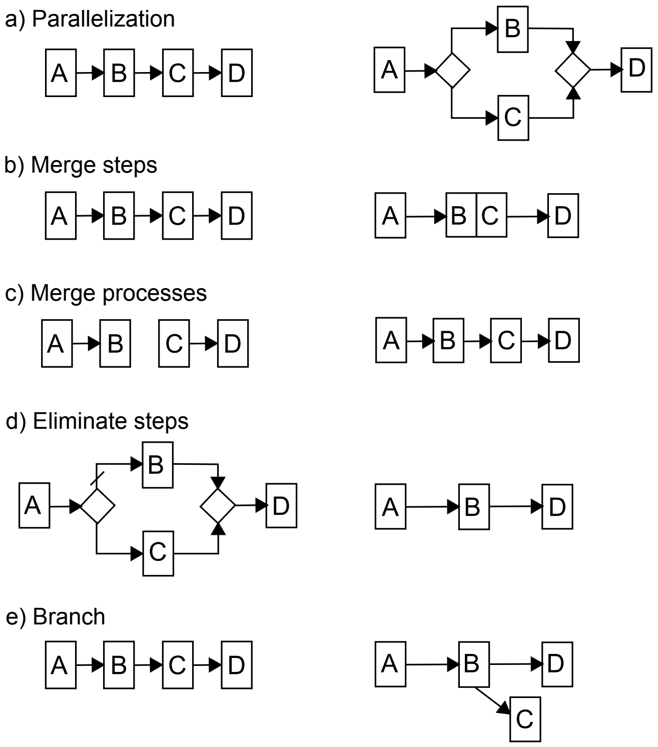


Figure 4.8: Optimization via Structural Changes

a) Parallelization

If two steps *B* and *C* are executed sequentially, but the results of *B* are not needed in *C*, the steps *B* and *C* can be executed in parallel.

b) Merge steps

When one and the same agent usually works on two consecutive steps, then those steps should be merged into one. Usually this means considerable savings as some actions are to be executed just once (finding the line in the worklist, opening the details, reading and understanding the case, closing the case, finishing the step and selecting the appropriate successor path or agent)

c) Merge Processes

Often the results of one process provide the initial input of a successor process.

A simple example is the application for a business trip. When the trip is approved, an expense report is to be produced later on. Those are parts of a larger overall process, that should be modeled and executed as one workflow. In the period between the approval and the expense processing, the process can be placed in the suspension worklist, the system will automatically recall the item when the trip finishes.

Process merging can also be accomplished on a larger scale, crossing organizational or corporate boundaries have a large potential for savings and improvements. Detailed and precise agreements with the partners on an technical as well as on an organizational level are needed. This area of business to business (B2B) communication is a very extensive one and is not in the scope of this book.

d) Elimination of Steps

From the run time data, one can find out if some steps are executed rarely or not even once. Such steps should be scrutinized from a business perspective as candidates for elimination or possibly merging with other steps, leading to simpler processes.

e) Branching

Many administrative processes contain steps for approval. Sometimes, such steps could be substituted by parallel informational steps. Instead of waiting with process execution for approval by a superior, the process will continue immediately, the supervisor will just be informed in a parallel step. In case of usually high approval rates and negligible negative consequences of unwarranted (temporal) process continuations this might be preferable.

A real world example might illustrate this [**?**]: Public officers in the province of Styria (Austria) that that want to undertake a business trip to a foreign country must get approval by the provincial government(!). The rate of denial of one to two applications per year let the provinces board of audit come to conclude that this does not warrant the effort and substantial delay in processing. The auditors suggested to change the process from an delaying approval step to a parallel information step. The suggestion was not implemented.

Let us now consider the optimization of individual tasks.

4.2.5 Optimization of Tasks

The optimization of the execution of an elementary task takes place principally in the areas of form processing and application integration:

a) Form Processing

The processing and handling of forms poses the following optimization potentials:

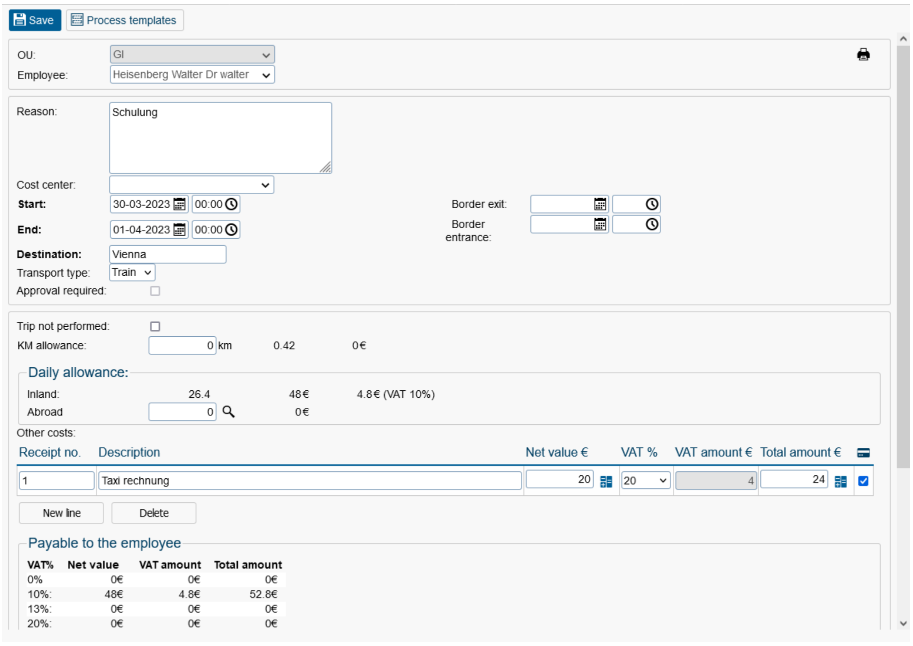
1. Avoid unneeded data: which of the form fields are changeable, but are never mandatory in a step? Which fields are not being used in subsequent systems or in the final product? Both kind of fields are candidates for elimination.
2. Automatic import of data from other systems or other process instances.
3. Automatic and up front filling in of sensible default values, like the own name in a form or the number "1" as amount in an order item form.
4. Selection instead of input: selection of a value is usually faster, simpler and more accurate than to input a value in a free form manner. Context information like other field values can be used to restrict the permissible selection to the minimum number needed.

In figure 4.9, a form is presented that has just 4 free form text fields out of 46 fields in total. The other fields are imported from other systems or the values can be selected with value lists.

b) Structuring of Unstructured Data

The processing of unstructured data (like in documents) does take significantly longer time than the handling of structured data (like in forms). The quality of structured data is better, field values can be checked, optional fields can be indicated, mandatory ones can be enforced. Specific help texts can be placed at field level (this is why public agencies choose to have an abundance of forms). Creation of documents with unstructured data should be avoided whenever possible. Time end effort goes into:

- ❑ calling the tools (e.g. office applications)
- ❑ collection of the data (often from process forms)



OU:	GI		
Employee:	Heisenberg Walter Dr walter		
Reason:	Schulung		
Cost center:			
Start:	30-03-2023 00:00	Border exit:	
End:	01-04-2023 00:00	Border entrance:	
Destination:	Vienna		
Transport type:	Train		
Approval required:	☐		
Trip not performed:	☐		
KM allowance:	0 km	0.42	0€

Daily allowance:

Inland:	26.4	48€	4.8€ (VAT 10%)
Abroad	0	0€	

Other costs:

Receipt no.	Description	Net value €	VAT %	VAT amount €	Total amount €	
1	Taxi rechnung	20	20	4	24	✓

Payable to the employee

VAT%	Net value	VAT amount	Total amount
0%	0€	0€	0€
10%:	48€	4.8€	52.8€
13%:	0€	0€	0€
20%:	0€	0€	0€

Figure 4.9: Optimized Form

- ❑ formatting the documents: years of work time are wasted with the formatting and layout of text processing documents.
- ❑ integration of the document into the workflow (depending on the BPMS)

Often, manual text processing occurs at the end of a process, when the result is a letter or official notification. Much more efficient would be the automatic generation of such documents on the basis of structured process data.

The question, if the user should be able to edit such letters after their generation is discussable. If the documents are in a standard form, we advise against it to avoid that effort and time that is wasted time for mere formatting or style.

c) Partial or Full Automation of Steps

BPMS offer a high potential for optimization by (at least partial) step automation. In section 3.8, the application integration area has been discussed. Candidates for automation are especially:

- ❑ data transfer between IT systems
- ❑ information transfer to persons outside the boundaries of the BPMS (e.g. sending an e-mail)
- ❑ checking of conditions, e.g:
  - order of an Internet connection: What kinds of connection are available at the street address?
  - application for a trade license: Does the applicant have the needed competence and prerequisites?
  - opening of an account: What is the customers credit rating?

  Several of such checks could be carried out completely automatic or at least all relevant data could be collected in a process form and presented to the user in order to enable efficient decision making.

What remains in the process are the productive steps, where *new* data is entered, decisions are made and externally (not computer-related) performed actions are being documented.

4.2.6 Implementing the Optimization

The changes must be designed and developed and lead to new versions of processes, tasks, forms and integration code. Those new versions can be technically activated at an predetermined date.

The evaluation of the changes again leads us to the analysis of the run time data (as having been dealt with at the beginning of this chapter). Execution times of processes and tasks can now be compared. Since a new version usually is a conglomerate of several changes, the effect of each individual alteration is not directly quantifiable.

At least equally important to the analysis of run time data is the feedback that users and customers of the process will provide. Which of the changes are viewed as positive which ones are cumbersome or disadvantageous? The software ergonomics (human computer interaction) field has seasoned techniques to gain such answers like questionnaires, observations and video analysis. Since system performance is largely dependent on the level of user and customer satisfaction, the opinion of those people will have a heavy weight in determining the quality of a change.

Chapter 5

Workflow Projects

Workflow applications are a special kind of software, so the development process used in workflow projects has much in common with general software development cycles. Nevertheless, some specific considerations apply for workflow systems development.

Mathias Weske et.al. have introduced in [?] a reference model for the conduction of workflow projects. A slightly adapted form of this model provides the basic structure for the following treatment. The principal phases of the workflow development process are coarsely depicted in figure 5.1.

We do not explicitly show iterations or feedback loops in this picture. Going back from a later phase to some earlier one in order to detail or revise previous work is permissible and sometimes highly desirable.

5.1 Survey Phase

The first phase is a survey, starting with the identification of the problems that the workflow application is intended to tackle and solve.

The following checklist can be used for identification of such problems

1. application name and synopsis
2. process description (textual form)
3. participants (organizational units, groups, persons)

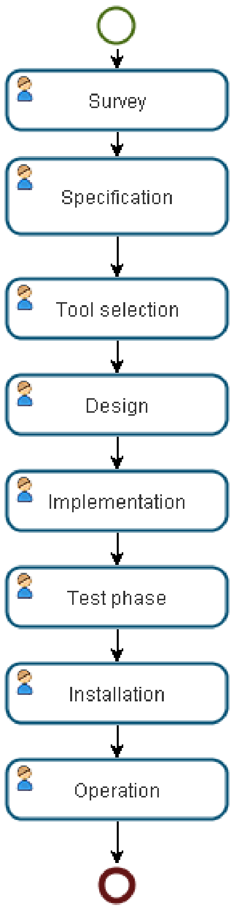


Figure 5.1: Development Process

4. interfaces to other applications
5. interfaces to applications or persons outside the scope of the organization
6. intended benefits and focus of the project:
    - ❑ reduced cycle times
    - ❑ automation of individual steps
    - ❑ better traceability and analysis capabilities
    - ❑ integration of legacy application
    - ❑ conformance with legislative rules and industry standards

   - ❑ increased quality (less errors, more complete, ...)
   - ❑ other benefits (elaborate)
7. intended deployment date
8. cost estimation
9. benefit estimation

In a kick-off meeting, such a checklist provides some guidance for the survey team to arrive at a principal decision about the projects feasibility. Clearly, some of the topics like costs and benefits can be handled only in the form of coarse estimations, there might arise the need for further and more detailed information gathering activities, but the following three questions should be clarified as result of the kick-off efforts:

1. Is workflow technology adequate for the solution of the problem?
2. Do the (projected) benefits warrant the (projected) costs?
3. Are the (projected) resources available?

Aspects of the first question have been already dealt with in chapter 1 (see page 12). The intention of the next two questions should be quite clear. If all three questions can (at least preliminarily) be answered positively, a project team can be appointed and the requirements gathering phase can start.

5.2 Requirements Specification Phase

The ultimate goal of this phase is the creation of a requirements specification. The core of this specification would describe in detail, how the processes in the scope of the project are intended to be executed in the future. But also several other important aspects of the IT system to be developed must be dealt with.

The crucial items that should be contained in such a requirements specification (specifically tailored to include workflow aspects) are:

1. General Description
   1.1. Goals and Scope
   1.2. Participants
2. Processes

- 2.1. Process 1
  - 2.1.1. Process Flow
  - 2.1.2. Way or Mechanism of Process Start
  - 2.1.3. Escalations and Exception Handling
- 3. Tasks
  - 3.1. Task 1
    - 3.1.1. Agents
    - 3.1.2. Preconditions and Postconditions
    - 3.1.3. Compensations
    - 3.1.4. Escalations and Exception Handling
- 4. System Steps
  - 4.1. System Step 1
    - 4.1.1. Function
    - 4.1.2. Interfaces
    - 4.1.3. Exception Handling
- 5. Data and Forms
  - 5.1. Overall data schema
  - 5.2. Form 1
    - 5.2.1. Fields (names, data types and domains)
    - 5.2.2. Graphical Layout
    - 5.2.3. Support Functions
    - 5.2.4. Form Field Visibilities per Process Step
    - 5.2.5. Relationship to Process or DMS
  - 5.3. Lists of Values
    - 5.3.1. List of Values 1
      - 5.3.1.1. Usage
      - 5.3.1.2. Value Sets
- 6. Functions
  - 6.1. Function 1
    - 6.1.1. Core Function
    - 6.1.2. Permission Aspects
    - 6.1.3. Placement in the GUI
- 7. Timers
  - 7.1. Timer 1
    - 7.1.1. Function
    - 7.1.2. Interval
    - 7.1.3. Interfaces
    - 7.1.4. Exception Handling

8. GUI
    8.1. General Layout and Navigation Concepts
    8.2. Specific Issues for each User Group
    8.3. Permission Aspects
    8.4. Adaptions for the Worklist Handling
    8.5. Adaptions in the DMS
    8.6. Desktop Integration and Tools
9. Reports
    9.1. Report 1
        9.1.1. Logical Content
        9.1.2. Layout and Presentation
        9.1.3. Permissions
        9.1.4. Parameters and Timer Integration
10. System Integration
    10.1. System 1
        10.1.1. General Description of System Nature
        10.1.2. Communication Mechanism
        10.1.3. Protocol and Data
        10.1.4. Error Handling
11. Milestones and Deadlines

The presented organization of the requirements specification document is based on the proposal of the IEEE [**?**]. We will now discuss some of the items. The section "General Description" contains the essential parts of the problem identification, presents the project synopsis and scope and clarifies the intended benefits for the participants and stakeholders.

5.2.1 Process Description

The process descriptions are the central items in the requirements document. Depending on the degree or formalization and documentation in the organization, more ore less detailed, precise and complete description of the as-is processes may be available. Nevertheless, it is of crucial importance to apply a "reality check" to those data. Interviewing the participants is an integral and essential task to gain insight in the procedures and manners of operation in practice.

One viable approach to understanding the individual steps and the process as a whole would be to "follow the flow", that is to accompany some real process instances in their way from participant to next participant. The following questions should be answered for each step:

- ❑ Who is involved, why this specific person?
- ❑ What is done and accomplished in this step?
- ❑ Why is the step needed?
- ❑ Which data is needed, which data is newly attached to the process?
- ❑ What tools are used?
- ❑ What is hard in this step, what are special cases?
- ❑ Are there alternative ways of doing things?
- ❑ How do the results influence further steps?
- ❑ What is next, where does the process continue?
- ❑ What are the time constraints?

The result would be a process description that is verified against the practice and in accordance with (representatives of) all process participants.

Other input for process descriptions are legislation, common rules and industry standards that might influence the process flow. Examples are needs for documentation and safe-keeping, traceability and audits, four-eyes principles, etc.

Additional care must be devoted to special and exceptional cases. How to deal with incomplete process data, or the need to sometimes "rush through" a particular process instance. There might be the need to explicitly model some of the anticipated important special cases as process variants or extra paths of the common process model for the standard case.

Process modeling could even in this early phase take place with @enterprise. Generally, the benefits of tool supported modeling here are rather complete data with seamless transition to a later phase, while the potential disadvantages would be to be caught too early in the details of technology and to be confined within its boundaries and thereby being distracted from the business issues and people. The low tech approach with e.g. flip charts, sticky notes, story boards and similar tools might be preferable.

Besides the process flow, it has to be determined how a process instance gets started. Will it be manually, will there be some external triggering event, or must it be started at certain points in time or at regular intervals?

For the tasks (the steps of the process that are to be carried out by humans), we need the following details:

- ❑ Agent: who is performing this step?
- ❑ Preconditions and post-conditions: automatic operations to be carried out during start of a task or during its completion.
- ❑ Compensations: Operations to be carried out when a task must be rolled back.
- ❑ Escalations: Treatment of exceptions and timeouts (business related issues).

For automatic system steps, the details of their function and data transformation and exchange are to be documented. Exception handling is also to be taken into account for automatic steps, but from the more technical perspective of possible faults and errors.

5.2.2 Process Data

Besides the process flow, process data is the next important aspect to deal with. Often there are forms or reports that contain the data. There might be paper based forms and there might be IT-based forms and screen masks which can be built upon. Sometimes, the complete data schema may be available for third party systems, or the data schema might be based on or related to an industry standard.

In the case of existing forms (paper based and electronic ones), it is of vital importance to use real instances with data (forms, that were filled out). So, the amount and nature of the data, its syntax and semantics can be much better understood. Exception handling by ad-hoc techniques like marginalia, additional documents stapled on, sticky notes and other creative approaches could be observed. On should strive to capture also this aspects directly in the form structures.

Besides the forms directly used in the process, additional data structures must be considered:

- ❑ Process relevant data
- ❑ List of values

For all those data, the following details are needed:

- ❑ the individual fields, their data types and domains,
- ❑ the permissions for viewing and changing, down to field level and with reference to a context or process state,
- ❑ nature of data administration: are the data originating from another system, are they references, must they be synchronized; is there manual administration of data within the BPMS?

5.2.3 Functions

While a BPMS provides a comprehensive set of standard functions for process execution, nevertheless further functions will have to be implemented for an application. The goal and operation of those functions must be described, their integration in the GUI must be detailed upon. We need to state, in what context they are applicable and what permissions are necessary to execute them.

5.2.4 Timer

Timer triggered events need to be elaborated with respect to their timing (points of time vs. regular intervals), and according to the actions that should be carried out on what specific data or processes.

5.2.5 GUI

A crucial aspect of the requirements of IT systems is the user interface.

In BPM projects, this item is slightly less relevant, since many functions are already prescribed by the BPM system. Most of the application specific GUI issues are dealt with in the areas of form layout and function design.

What remains to be treated is the adaption of the out of the box GUI provided by the BPMS. It must be decided, if there is the need for individually tailored GUI configurations for the different user groups (like presented in section 3.9.1). It might be sufficient, to provide one integrated GUI that just fades out certain components and functions, depending on the users role assignment and permissions.

Additional areas of adaption are functionality and layout of the worklist client and the functions and folder layouts in the document management system.

5.2.6 Reports

Description of reports comprises: what are the parameters, what should be included in the result, what is the format and layout of the report, who is allowed to execute the report, how is the report linked to other reports?

5.2.7 Integration

The last area of requirements specification is the integration with other IT systems. In section 3.18, a set of possible integration scenarios was presented. The requirements must state those external systems and the kind of integration: initial data import, periodic synchronization, real time access needs, error handling, etc.

5.2.8 Project Deadlines

It is not feasible to provide a detailed project plan in the requirements phase, but crucial deadlines must be stated and a rough idea of the intended milestones should be provided, e.g. "must be productive at Jan. 1st, 2012 " or "field test during summer"'.

5.3 Product Selection

In the project plan given by Weske et. al. in [?], the product selection takes place after the design phase. This ensures that product selection does happen before all the relevant details are available. We would like to precede design with product selection, since then, in the design and prototyping phases, the selected tool can already be used.

The criteria for product selection would be dependent on the requirements and the general profile of the application, nevertheless we will present an outline for a general criteria catalog, which can be the starting point for a project specific one [**?**].

The first level structure of the catalog closely corresponds to the chapter structure of this publication: modeling, execution, integration, reporting and analysis, general; the last item just contains general points for acquisition of IT systems.

1 Modeling
1.1. Process Definition
- Process definition language used to describe the processes. - If BPMN, which language constructs are supported? - Options for and kind of process versioning. - Process documentation: is there a - printable one - provided? - Are all details contained in the documentation? Is is suitable for end users? - Is a graphical process designer available? - Manner of definition for conditions, expressions and constraints? - Web-based process definition via browser possible? - Are there any additional components needed for this? - Are the process definition component and the run time component integrated or separated? - Availability and power of an API for process control?
1.2. Data Definition
- Is a form designer component provided? - Which data types are supported? - What input checks are provided, how are they defined; are they extensible? - How can dependencies upon form fields be defined? - How are 1:n and n:m relations being mapped? - Which layouts are available (free-form, table, two columns, multiple columns)? - Can forms be fine tuned outside of the form designer?

1.3. Organization Definition
- Which entities are in the organizational model? Is the schema available? - What substitution options are provided? Are there role substitutions, multiple substitutions, substitutions for time intervals? - Is there a browsable display for the organizational structure? - Are there hierarchical structures available? For organizational units? For roles (in the sense of role inclusion)?, For role assignments (hierarchical role scope)?
2 Execution
2.1 What degree of run time flexibility is supported?
- Going back - Delegation - Copy to - Ad-hoc extensions, which controls structures can be inserted?
2.2 Escalations and Exception Handling
- Which process states and what events can trigger escalation actions? - What escalation actions are provided? Are they extensible? - How are errors during calls of external applications being handled?
2.3 User Interface
- Is there a browser based client? Which browsers are supported? - Native browser support or are plugins needed? - How can the GUI be adapted? Is it configurable; is an API available? - Can the columns of the worklists be adapted and freely defined? - Can the worklist be structured (via tagging or via folders)?

- Can forms be displayed differently in different views? How?
- Are there several and differently adaptable GUI layouts for diverse kinds of users (roles)?
- How is branding / corporate identity accomplished?
- How is internationalization of processes, forms and other process components done?
- What natural languages are supported / available in the GUI?
- What is the participants view of the process history?
- Can a process be aborted and reactivated later on?
- How are changes being logged? Can changes be traced down to single field level?
- Is a web based administrative interface provided?

- Is there an interface for mobile clients?
- Is it a Web application or a mobile app (which mobile operating systems are supported)?
- How is the mobile interface adaptable and extensible?
- What restrictions apply for mobile GUIs?

- Is there a context sensitive online help system?
- Which manuals are available? Is the depth and extend of documentation adequate?

2.4 Security

- Which authorization mechanisms are available (username/password, certificates, smartcards)?
- Is single-sign-on supported? In which environments?
- Is role based access control supported for the permission assignment?
- Can permissions be assigned with respect to organizational units?
- Can permissions be assigned on individual processes, documents or folders?
- Can the permission system be used to define restricted system powers e.g. for operators, user administrators, process owners?

2.5 Document Management
- Can arbitrary documents be attached to processes? - Is there also a process independent document filing area for documents? - Is access to the documents also governed by the permission system? - Can the documents be enriched with arbitrary meta data? - How is scanning equipment being integrated? - What versioning schemes are provided? - How are documents being archived? - Can documents be stored in encrypted form? - Are electronic signatures of documents supported?
2.6 Reporting and Analysis
- Is an integrated reporting component provided? - Can run time process data be compacted and summarized? - Is there an adequate user interface for the report designers available? - Can reports also contain application data (form data)? - What output formats (HTML, PDF, Excel, CSV, charts) are provided? - Can reports be linked in a drill-down manner?
3 Integration
3.1 Enterprise Application Integration
- How can third party applications be integrated? - Which forms of integration of messaging (e-mail, social media) is provided? - Is there a notification mechanism for events? - How can organizational data be synchronized and kept current?
3.2 Architecture

- Scalability: how is adequate performance being provided under peak load (clustering, load balancing, etc.)? - Is a sizing guideline for hardware and infrastructure dimensioning available? - Which data base management systems are supported? - What software requirements are posed on the platform? - Which standards are supported?
3.3 API
- Is there an API? - How is it documented, are examples available? - What language bindings are provided? - Is it possible to use the BPMS as an embedded engine?
4 General
4.1 License and Costs
- What license models are offered (named users, concurrent users, pay per use, enterprise-wide license)? - Is a test environment included in the license cost? - What support contract models are available? At what cost? - What service level agreements are provided? - How about commercial stability of the vendor? - Product maturity; since when is it on the market? - Are there reference installations? - What is the installed base (installations and seat counts)? - Can resources be provided for consulting and implementation? Are such services available on the market? At what cost?
4.2 Operations
- How can the basic software be upgraded? - How can applications be upgraded? - Is there an import and export facility for processes and other artefacts?

- Is a scripting language provided?
- Availability of trainings and courses for end users, process designers, administrators and developers? Concrete contents, durations, cost?

Besides those common selection criteria, normally there are additional criteria specifically applicable in the context of a project or an organization. It must also be noted, that the catalog does not weigh the items. Individual weights must be assigned to the selection criteria to document their relative importance for the specific selection project, in particular a differentiation between essential "must-haves" and optional "nice-to-haves" can help to quickly assess principal product suitability.

5.4 Design and Implementation

We will treat the next two phases - design and implementation - in a combined way. Since they can be overlapping and are not necessarily be carried out in a strictly sequential manner. One can follow a very agile and prototype driven approach with clear advantages with respect to development time, product quality and user participation and satisfaction.

In the design phase, the individual components of the system are specified, their intended behavior is described precisely and completely. In the implementation phase, the components are created on the basis of this specification.

The components of a workflow application are partly the process definitions and their constituting artefacts, partly programs written in a programming language.

The components of the first kind are largely finished when the definition is finished, just for the program code, an explicit implementation phase is needed. The differentiation is often somewhat blurred, let us look at some examples:

- ❑ Process description in BPMN
- ❑ Definition of a condition via XPath
- ❑ Definition of a post-condition in Java

The main difference between the three examples is the generality of the language being used. BPMN is applicable for the definition of business processes, XPath is a navigation language for XML documents and Java is a general purpose programming language. The creation of software based on models in domain specific languages is called *model driven software development* [?]. This approach tries to avoid specific implementation steps to a large extent.

During the creation of a workflow application, for each component there is a specification being made followed by an optional implementation phase for the component.

There are two results of this phase; there is a design document - structured like the requirements specification - that provides all needed details for the implementation and there is also an implementation of the processes, which can be tested right on in the next phase.

5.5 Test Phase

The test phase is needed to decide upon the applications fitness for use and to ultimately achieve this quality. Error identification, documentation, tracking and correction will take place. Usually, the test is divided into two parts, the *lab simulation* and the *field test*.

5.5.1 Lab Simulation

The lab simulation can be started when significant parts of the implementation are completed.

The tests are driven by a *test plan*; in the case of a workflow application the process flow and structure can provide an orientation for the structure of the test plan and for the individual tests.

An adequate predefined organizational structure containing the relevant users, departments, roles, role assignments and permissions build the base of all tests.

The test is carried out by following the control flow of the process, while impersonating the participants. Conditional execution paths and loops must be considered, several variants of the process must be executed, until (theoretically) all possible execution paths were covered by the tests.

In every process step, conformance to specification is to be checked:

- Functions: which functions are provided - are they behaving correctly?
- Forms: correctness of field visibilities, is all needed information available, and visible; are the proper fields editable, are mandatory fields really mandatory?
- Check wrong inputs for all fields. Are adequate error messages provided for all invalid inputs?
- Conformance with documentation.

When errors are observed, they get properly described, documented and reported to the development team. Corrections are implemented and a new version of the application is being made available on the test platform.

The test phase results in a *test protocol*, which gives insight about the kind of the test cases that were executed, about the errors fixed and the errors still remaining in the applications. After lab simulation, a field test takes place.

5.5.2 Field Test

In this phase, the application and the processes are being tested by selected "real" users.

Before the field tests, the users must be trained appropriately. The function of the application is presented and the users do walk through process instances in a manner like they would in a production situation.

The field test is not primarily intended to uncover programming errors or glitches of a purely technical nature, the emphasis is on correctness of the application and on ergonomic matters; main issues would be:

- Missing functionality: walking through the cases reveals functions that would be essential or helpful in the day to day use of the application.
- Wrong functionality: some function is not carried out appropriately. The error may be rooted in the requirements specification, the design or in the implementation.
- Poor usability: A tester working completely through a single business case during the lab test is not likely to encounter certain kinds usability problems. Issues and annoyances and potential optimizations in ergonomic aspects of software often reveal themselves only when using

them over and over again in a realistic work pattern. Field tests are better suited to identify such problem.

The field test is also done in a feedback cycle, where the identification and reporting of problems would lead to additional development effort until a new version of the application will be installed on the test platform.

After positive completion of the test phase and with a detailed test protocol as documented result, the application can be transferred to the production environment.

5.6 Installation

Implementation of a workflow project takes place on one or more development systems. From there, the artefacts are integrated and transferred to a test system. This is the working platform for the test phase. After this, the application is deployed on the production system. The test environment should not be dismantled, but rather be still available for the support personnel to be able to reproduce problem situations emerging in the production system.

Another system would be a training platform. It should be separate from the production system (to allow interaction without consequences), and also separate from the test system (since on the test system, a future version of the application could already have been installed).

Usually, the installation in the production system will not be conducted by the developers. The software and an *installation instruction* will be delivered to operations people with a technical background but who cannot be expected to be deeply acquainted with the area of workflow. The installation documentation should try to avoid workflow specific terms and the procedure should be largely automatic and scripted, abstaining from interventions via the administrative GUI.

The installation of an application is carried out in two parts: the installation of the BPM system itself, followed by the installation of the application. For the first part, one can build on vendor provided procedures and installation routines. The deployment of the application can be done via a sequence of manual steps via the administrative interface, or scripts could created and used. We will discuss both variants in the context of @enterprise.

GUI based application deployment

A function *Install Application* is provided to initiate the deployment of an application in the system. The application would have been packed in an archive (a *zip* file format), which contains the program code, an initial configuration of the application and an export file containing the needed master data objects (process definitions and components, ...). The function unpacks the archive to a target directory, adapts the path definitions in the system and imports the master data objects.

There may be the need to adapt the parameters of the initial configuration, this can be also done via the administrative GUI.

Administrative Shell

When it is not possible or desirable to manually install the application, be it because of policy regulations of the platform provider, or for reasons of traceability, scripted installation is possible.

@enterprise provides an administrative command line interface (a script console) where administrative actions can be carried out without browser access. Sequences of commands can be combined to scripts. In particular, for application deployment such a script can be created. Remaining actions for the administrator are:

- ❏ unpack files to the target directory
- ❏ adapt parameters in the install script
- ❏ run the install script

This requires minimal manual intervention, is documented, traceable and repeatable. The script is formulated in Groovy (*http://groovy.codehaus.org/*), a powerful and popular Java-based scripting language.

A chapter of the @enterprise administration guide is devoted to the details of the administration shell [**?**].

5.7 Application Upgrade

After initial installation, the need to install updates will arise more or less frequently. Code changes will have to be put in place and database objects (process definitions and components) will have to be adapted.

The approach is not that different from an initial installation. For the database objects, the XML based import mechanism (of a file that has been exported from a development system) can be used again, the changed code is copied to the application class path directory.

The XML import differentiates between new objects to be freshly created and existing objects, to which the appropriate modification must be applied. For instance, a process definition could have been extended with several additional steps.

In some upgrade cases, further actions need to be performed like data base operations, or changes to permissions assigned to roles. Those actions could be implemented as a Java method which is registered as a special application upgrade method. Then, those actions could be initiated via the administration GUI. Or again, a script for execution in the administrative shell could be created.

The actions needed to be performed by an administrator (besides general precautions like backup procedures):

- ❑ inform the users about the planned upgrade and downtime
- ❑ deactivate user logins
- ❑ stop the server
- ❑ unpack the files to the application directory
- ❑ start the server
- ❑ execute the script with the upgrade actions,
- ❑ activate user logins

Again the scripting provides for a repeatable and documented upgrade procedure (a protocol file can be generated via the `-log` parameter).

With the deployment of the system, the initial development process is completed. Figure 5.7 gives an overview of the documents generated in the individual phases.

We will conclude the chapter with some discussions about tasks during the operation of a workflow application.

Document	Project step
Problem statement	Survey
Requirements Specification	Requirements Gathering
System specification	Design
User documentation	Implementation
Administration guide	Implementation
Installation instruction	Implementation
Test plan	Implementation
Test protocol	Test phase

Figure 5.2: Documents created in the development process

5.8 Operation

The tasks during the operational productive life of the application can be classified into the following categories

- ❑ technical administration: maintaining the infrastructure
- ❑ organizational administration: maintenance of the organizational structure
- ❑ process administration: check the process flows, process analysis and monitoring
- ❑ issue, problem and change management: planning, structuring and executing changes

The assignment and distribution of the tasks or their bundling will depend on the size and degree of specialization within the organization or may be partially delegated to an external service provider.

5.8.1 Technical Administration

The technical administration is concerned with the uninterrupted operation of the technical infrastructure where the BPMS is installed, like database capacity, storage capacity, processing power and operating platform.

A technical administrator would take care of the server infrastructure, she would not need detailed knowledge about the BPMS, except from the following aspects:

1. update procedures
2. providing log information for bug identification and handling
3. appraisal of resource usage and proper system operation
4. execution and monitoring of backup tasks

The updates are made on the basis of the prescriptions of the developers, see above. The second and third items are supported by administrative functions available in the @enterprise administrative GUI (server monitor, server control and configuration). In the server monitor, basic items like memory consumption and state of the database connections can be checked. The technical administrator would use a special *sysadm* account, which will also function even in the case of missing connectivity to the data base.

Tasks in the third and forth group of items must be performed on a periodic base. Monitoring the resource consumption would take into account the areas of load on the data base system, the CPUs, the file systems, the network and number of licensed seats and ensuring that no bottleneck will be encountered.

5.8.2 Organizational Administration

When there is no automatic synchronization mechanism with an external user administration system like a directory server, the maintenance of the organizational data must be done manually. New user accounts would have to be created, accounts for retireing users must be deactivated. Role assignments would have to be adapted. Permissions will have to be granted and revoked. Even in the case of an automatic synchronization with a directory service, some of the actions would usually be BPM system specific and to be carried out manually nevertheless (e.g. role assignments).

Reorganizations are a further issue. Whole organizational substructures could be pruned or otherwise changed. Since running process instances have an organizational context, it must be ensured that existing processes would continue to run in an orderly manner. Newly created process instances must comply and follow the new organizational structures.

In the case of (unexpected) absences of employees, it might be necessary to reassign tasks from the worklist of one employee to other employees or to assign some substitutes.

5.8.3 Process Administration

Basic system performance is the area of responsibility of technical administrators. But the business processes must also be assessed according to their business performance and efficiency. Questions to be answered are:

- Process performance: How long do the process executions take? Is there a trend developing? Compare as-is performance with prescriptions and assumptions.
- Bottleneck identification: are there bottlenecks in the area of individual users or roles or in certain process steps?
- Exception handling: Are there overdue processes and steps? Are there unduly long running processes or tasks, are there processes waiting long times for external events (e.g. completion of batch steps), are there processes without active participants?

Answers could be gained with the help of the administrative functions and with specific reports. On the basis of this data, appropriate correctional measures can be devised and executed.

The function of responsibility for process performance could be assigned for the whole BPMS or could be individually assigned for particular process definitions to special persons, the so called *process owners*.

The process owners are responsible for creating detailed reports for their own purposes, as well as for providing condensed information for upper layer management. Usually, those reports will be developed, enhanced and fine tuned during the operational phase based on the insight of the operation itself.

5.8.4 Error and change management

During the operation of the applications there will be errors uncovered and other deficiencies be identified. The processes themselves are exposed to some "aging", since in a living organization, there will be more or less continuous change. Those changes can be motivated either internally by process monitoring and optimizations or changes in infrastructure. They could also be imposed externally by changes in standards or laws. New data must be transported, some may become obsolete. New interfaces to systems or changes in platform software versions also would lead to maintenance efforts concerning the applications in the BPMS. A structured approach to changes is needed to maintain system integrity in spite of of those fluctuations. Change requests and needs for change must be collected, assessed and properly and orderly executed.

In @enterprise, an IT service management application (*ITSM*) can be installed and be operated simultaneously to the core applications. With the ITSM application, incidents (suggestions, wishes, perceived errors) can be reported, collected and assigned to the support and maintenance personnel. The incidents are assessed and classified as being problems (dealing with errors) or changes (dealing with features).

The tasks of the release management process are the bundling of (resolutions of) problems and change requests and to provide new consistent releases.

The overall change management process very much depends on the organization, it comprises at least the following activities: bundling of changes, sign-off for implementation, implementation of changes, test planning and execution for changes and deployment.

5.8.5 BPM in the Cloud

Cloud computing is one of the newer buzzwords and is gaining increasing popularity as a means to delegate and to outsource the operations of systems and applications to a service provider in order to reduce costs or to gain flexibility.

The application runs on the platforms in a data processing center of the service provider. The users are not concerned about the location of the application, it is positioned somewhere in a virtual "cloud". Several variants of cloud computing have been established (see [**?**]):

1. Infrastructure as a service, IaaS: the service provider allocates the computing platform, that is hardware, operating system, data base services, network access and sees to the technical administration that is concerned with the availability of the components. The BPMS is solely in the responsibility of the customer organization.

2. Platform as a Service, PaaS: in addition to the infrastructure, the BPMS is also administered by the service provider, the customer is doing process definition and maintenance and is responsible for interfaces to other applications. Technical administration is largely in the hands of the

service provider.

3. Software as a Service, SaaS: A complete BPM application is being made available by the service provider (e.g. CRM, Human Resources, ITSM). Customer specific data can be incorporated along predefined interfaces.

4. Process as a Service: this is the most radical form of outsourcing, the execution of whole processes or of some process parts would be solely in the hands of the service provider.

Let us give two examples for process outsourcing:

Invoice processing with the tasks of scanning and tagging, and payment can be done by personnel of the service provider, just the most critical step of checking the invoice and signing off its payment is performed in house.

In service management, complaint recording, user assistance or replacements of faulty components could be delegated to a service provider.

The expected advantages through deployment of BPM technology in the Cloud are a consequence of the outsourcing of peripheral (non-core) tasks in terms of the enterprise goals to specialists. The operation of a data center, of a data base management system, the updating of diverse software components is better placed in the hands of an IT specialist than in the hands of e.g. an industrial production plant. But there are also risks to be considered, especially in the areas of security, confidentiality of proprietary information and a deep level of dependency upon the service provider. Proper design of outsourcing contracts and the supporting measures is a heavily discussed topic.

About the Authors

Herbert Groiss has been deeply involved with the theoretical as well as the practical aspects of business process management for more than 15 years. He is founder and CEO of Groiss Informatics GmbH, a company which develops the business process management system @enterprise and implements solutions based on this foundation. Before entering the private sector, Herbert was a scientist at Technical University Vienna as well as at Klagenfurt University, were he worked in the areas of artificial intelligence, data base systems and workflow systems. He holds a Ph.D. in computer science from Technical University Vienna.

Michael Dobrovnik has been occupied in the fields of workflow management and business process management systems for about 15 years in the roles of developer, architect, consultant and head of development. Before joining Groiss Informatics, he was scientist at Klagenfurt University, working in the areas of database systems and information systems. He holds a Ph.D. in applied computer science from Klagenfurt University.

The authors can be contacted via e-mail at *herbert@groiss.com* and *michi@groiss.com*.