# Business Process Modeling in Software Development*

Veronika Thurner
Department of Computer Science, Technical University of Munich
Arcisstr. 21, 80290 Munich, Germany
email: thurner@informatik.tu-muenchen.de

**Abstract**

This paper is structured into two parts, dealing with notational and methodological aspects, respectively. The first part introduces a description technique for business processes which provides both an intuitive, graphical notation and a textual syntax. The definition of semantics of this description technique is related to a mathematical system model. Using this mathematical system model as a common formal basis, the relation between business process model and other models employed in software development can be precisely defined. In the second part, a business process driven method of software development is outlined, as well as a systematic approach to business process reengineering. As software development and business process reengineering are closely related and often occur together in practice, these two approaches will be integrated into a method which systematically combines both techniques.

## 1 Motivation and State of the Art

In many software engineering methods, approaches to requirements engineering involve a detailed modeling of different aspects such as system structure, data or behavior. These models are an essential means of communication between system developers and expert users. Furthermore, they are the basis from which system design and implementation are derived in later stages of the development process.

As the quality of requirements specifications is a decisive factor for software quality and correction costs [Dav93b], much effort is usually spent on system modeling in the early stages of the software development process. However, the models developed quite often only aim at providing the system developer with a better understanding of the system to be developed, rather than producing a set of unambiguous, consistent and semantically integrated documents which support an (at least) half-automated derivation of subsequent results in the development process, such as design or implementation documents. Thus, the high effort spent on modeling is often not used effectively.

Therefore, in order to support system development in an optimal way, description techniques for models of specific system views must be intuitively understandable on the one hand. On the other hand, description techniques must be precise enough to ensure an unambiguous and consistent description of the system. Consequently, a precise definition of its semantics should be provided for *each* description technique that is employed for system modeling. Furthermore, a semantical basis common to *all* the description techniques involved

*This work was carried out within the Project SysLab, sponsored by Siemens-Nixdorf and the German Research Community (DFG) under the Leibniz program.

has to be defined, which allows the precise definition of the interdependencies between the different models and system views.

"Classical" software development methods, such as the structural approach SSADM [CCT90], or the object-oriented approaches OOA/OOD [Boo94], OMT [RBP+91] and OOSE [Jac92], provide powerful, usually graphical description techniques. However, these are not semantically well founded. As a consequence, misunderstandings due to different interpretation of notational elements occur. Furthermore, models are ambiguous, so that inconsistencies are hard to detect. Due to the insufficient definition of semantics, sophisticated services such as consistency checks cannot be automated, so that tool support usually is limited to graphical editing tools.

Recent software development methods attempt to provide some semantic foundation to their description techniques, quite often by using a metamodel, as was done e.g. for UML [BJR96]. Metamodels are usually documented as some sort of class diagram or ER diagram. They focus on statical system aspects but do not possess an interpretation that models dynamic aspects of system behavior appropriately. Thus metamodels are no sufficient definition of formal semantics for description techniques. The semantical adequacy of the metamodel provided for UML is discussed in more detail in [BHH+].

In the SysLab project, we use a mathematical system model ([RKB95], [GRK96]) which is based on stream processing functions [BDD+93] as a foundation for formalizing and integrating the various description techniques used in SysLab for specifying different system views ([Het96], [GKRB96], [GR96]).

The doctoral thesis corresponding to this paper will cover the definition of an adequate description technique for business process modeling, which is both intuitively understandable and semantically well defined. The definition of semantics and the relation to other description techniques is based on the mathematical system model of SysLab [GRK96].

Furthermore, methodological aspects of business process modeling in system development are treated. In the business reengineering community, the introduction of adequate information technology is seen as one of the key enablers of innovation and radical improvements within an enterprise (for example, confer [Dav93a]). Thus, business reengineering very rarely takes place without dealing with software development and customizing to a certain extent.

When software is developed for supporting human users in the execution of some defined tasks, usually an attempt is made to engage the specific advantages of information technology effectively, rather than mereley automating existing (often paper oriented) execution mechanisms. At the introduction of the new software system to the respective business organization, execution mechanisms and, more globally, business processes are adapted in order to use the new system support effectively. Thus, business process reengineering and software development are closely related and often occur hand in hand. However, although expertise on either one of the two domains is abundant in literature, both approaches have not yet been integrated in a satisfactory way.

We present a business process oriented approach to software development, as well as a method for business process reengineering, which is based on the experience gained in business process modeling and reengineering projects in corporation with Siemens-Nixdorf AG. The systematic integration of both methods is currently under development.

# 2 Description Technique for Business Process Models

After an informal introduction of basic concepts, we present a graphical and a corresponding textual description technique intuitively by way of example.

## 2.1 Abstract Syntax

As key concepts of a business process we use

- process / acitivity,
- message, and
- event, which is defined as a tripel of message, sender and receiver activity.

Using the intermediate concepts of role (or logical actor) and logical channel (defined as a pair of sender and receiver activities), we relate the key concepts to other notions fundamental in system modeling, such as component, object, data type, physical channel, and physical actor. Helpful for structuring process diagrams, but not formalized here, are the concepts of task and phase.

### 2.1.1 Informal Definition of Process Concepts

An *activity* defines the creation of some resulting output out of incoming messages. A *process* is an activity that is further refined. We call an object that is passed from one activity to another a *message*. Each sending and receiving of a message is an *event*. It defines the causal relationship between a pair of activities due to the passing of a message, thus implying a partial order over the set of activities of a process.

Via the concept of a *role* or *logical actor*, we flexibly relate activities and physical actors, which may be human beings (or organizational units), hardware, software systems, or a combination thereof. Activities are associated with roles in n:1, roles with physical actors in m:n relationships. Messages are passed via *logical communication channels*, connecting the sending and receiving activity. When physical actors are assigned (via roles) to activities later on in the modeling process, the logical channels are mapped to physical channels connecting the system's physical actors. Each business process performs a specific *task* that helps to achieve some of the system's business goals. *Phases* structure a process according to milestones and significant intermediate results.

### 2.1.2 A Mathematical Notion of Processes

Formally, we define a process as a triple $p = (A, M, \varepsilon)$ where

- $A$ is a set of processes or activities,
- $M$ is a set of messages, and
- $\varepsilon \subseteq M \times A \times A$ is the event relation (which we require to be acyclic), defining the set of events that occur in process $p$.

From the event relation $\varepsilon$, we derive a partial ordering relation $\leq$ over the set of activities $A$ in a process $p$ according to

$$a_1 \leq a_2 \iff a_1 = a_2 \quad \vee \quad \exists_{m \in M, a \in A} \quad a_1 \leq a \ \wedge \ (m, a, a_2) \in \varepsilon$$

for any $a_1, a_2 \in A$. Reflexivity and transitivity are ensured by the definition of $\leq$. Antisymmetry follows from our restriction of process $p$ to be acyclic.

A set $R$ of roles is related to the set of activities $A$ via the relation $\varrho \subset A \times R$. We restrict $\varrho$ to hold $(a, r_1) \in \varrho \wedge (a, r_2) \in \varrho \implies r_1 = r_2$ for all $a \in A$ and $r_1, r_2 \in R$, so that each activity is associated with only a single role.

## 2.2 Concrete Graphical Syntax

As graphic representation of business processes, we use directed acyclic data flow nets, derived from data flow diagrams introduced in [DeM79]. The expressiveness of this notation is sufficiently powerful, since we restrict our business process model to the description of *exemplaric* system behavior (cf. section 4).

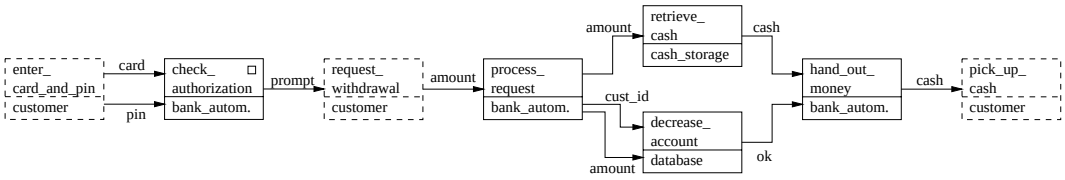



Figure 1: Process diagram for withdrawing money from a teller machine

Figures 1 and 2 show example process diagrams. Rectangular nodes represent activities or processes, where solid surrounding lines denote internal activities, dashed lines external ones. Decision activities symbolize their possible outcomes by a set of fields annotated with conditions (see verify_identification in Figure 2). A small square in a node's upper right corner indicates the activity's further refinement. Roles are denoted at the bottom of an activity symbol. Arrows annotated with a type symbolize data flow.

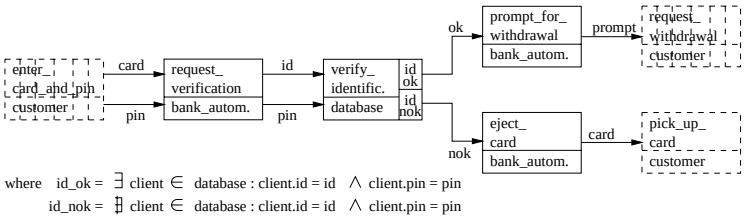



Figure 2: Refinement of process check_authorization, and its context

Figure 2 shows the refinement of process check_authorization. The neighboring nodes of the refined process are copied in order to visualize the process context on the refinement level. They are distinguished by their dashed background.

In the classical notion of data flow nets, an activity $a_2$ starts processing only after receiving *all* its input from all its direct predecessor activities $a_{1_1}, \ldots, a_{1_i}$. We refer to this interpretation of message passing as *end-to-start relationship*.

For practical purposes, this interpretation proves to be too restrictive to be suitable. Therefore, we suggest the more general notion of *middle-to-middle relationship* or *dependency*, where the relation of some $a_1, a_2 \in A$ with $\exists_{m \in M} \quad (m, a_1, a_2) \in \varepsilon$ indicates that at some point of time in the execution of $a_1$, message $m$ is sent from $a_1$ to $a_2$. This interpretation does not enforce an ordering relationship between *all* the outputs and inputs of two activities.

## 2.3 Concrete Textual Syntax

Our textual description of business processes is based on *black box*, *glass box* and *refinement view*. Black box and refinement view together are equally expressive as the graphical process description, leaving the choice of representation to the user according to his or her purposes. We complement both by the glass box view, which documents aspects of an activity's internal realization, such as pre- and postconditions, or any data that was created as output in some other process, and is to be read or modified by the activity.

We based the definition of our textual syntax for processes and activities on some ideas that were introduced first in [CAB+94] as operation schemes. However, we extend their concepts and use a formalized notation rather than natural language. In the following, the textual notation is illustrated on part of the example process introduced above. A definition of the concrete textual syntax in EBNF-notation is given in Appendix A.

**process model** behavior_of_teller_machine = {withdraw_money, retrieve_account_information }

**process** withdraw_money = {
enter_card_and_pin, check_authorization, request_withdrawal, process_request,
retrieve_cash, decrease_account, hand_out_money, pick_up_cash
}

**black box** check_authorization = {
**type** **internal**
**role** bank_automaton
**input** (card, enter_card_and_pin, check_authorization),
(pin, enter_card_and_pin, check_authorization)
**output** (prompt, check_authorization, request_withdrawal)
}

**glass box** check_authorization = {
**reads** customer_record **from** customer_database
**where** customer_record.id = **input**.card.customer_id
**changes** customer_record.nr_requests **from** customer_database
**where** customer_record.id = **input**.card.customer_id
**to** customer_record.nr_requests = customer_record.nr_requests + 1
**pre**
**post** **input**.pin = customer_record.pin
**where** customer_record.id = **input**.card.customer_id
}

**refinement** check_authorization = {
**subactivities** request_verification, verify_identification, prompt_for_withdrawal, eject_card,
pick_up_card
**subinput** (card, enter_card_and_pin, check_authorization)
= (card, enter_card_and_pin, request_verification),
(pin, enter_card_and_pin, check_authorization)
= (pin, enter_card_and_pin, request_verification)
**suboutput** (prompt, check_authorization, request_withdrawal)
= (prompt, prompt_for_withdrawal, request_withdrawal)
}

**black box** verify_identification = {
**type** **internal**
**role** database
**out_switch** id_ok := $\exists$ client $\in$ database : client.id = id $\wedge$ client.pin = pin;
id_nok := $\nexists$ client $\in$ database : client.id = id $\wedge$ client.pin = pin
**input** (id, request_verification, verify_identification),
(pin, request_verification, verify_identification)
**output** id_ok: (ok, verify_identification, prompt_for_withdrawal);
id_nok: (nok, verify_identification, eject_card)
}

...

# 3 Towards a Formalization of the Description Technique

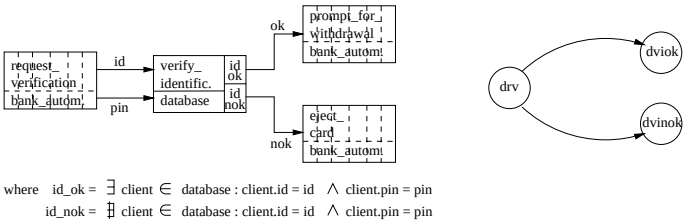
The semantics of our description technique for business processes will be defined with respect to a mathematical system model, forming the basis for relating business process descriptions to other concepts of system modeling.

## 3.1 Mathematical System Model

As described in more detail in [GRK96] and [RKB95], in SysLab a system is seen as a set of hierarchically structured, communicating components. A component is understood in the object-oriented sense as a unit of data values (states) and functionality with a published interface and providing some service. Communication is realized via a communication medium that transmits messages among the ports of components.

In the mathematical system model, component behavior is given as a black-box description. For basic components (i.e. components that are not refined any further), behavior may in addition be defined in a state-oriented way. As component types, we distinguish process, role and data components.

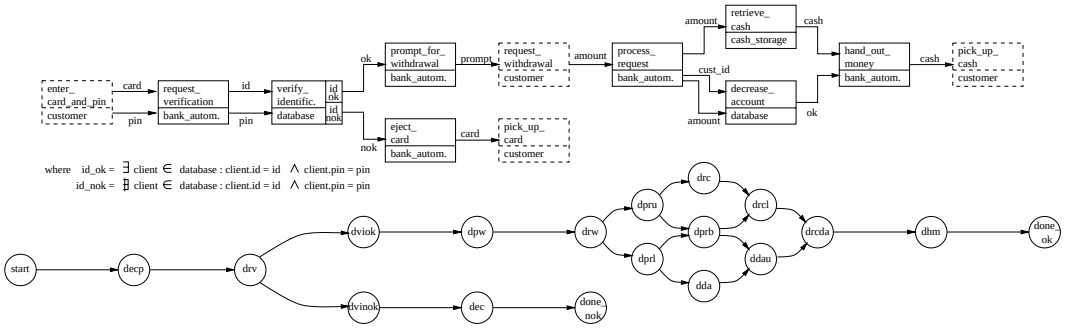
## 3.2 Process and State Transition Diagrams




| from_ state | precondition | input | output | postcondition | to_ state |
|---|---|---|---|---|---|
| drv | ∃ client ∈ database : client.id = id ∧ client.pin = pin | id, pin | ok | | dviok |
| drv | ∄ client ∈ database : client.id = id ∧ client.pin = pin | id, pin | nok | | dvinok |

Figure 3: Automaton for activity component verify_identification

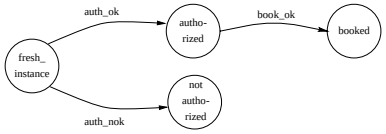
Depending on the stage of development of our system's process model, we focus on different aspects as components. First, each process or activity itself is seen as a component. Thus a single component is associated with each activity, providing the service rendered by this activity and encapsulating the activity's local data. Figure 3 shows the state oriented view of a single activity.



| from_ state | precondition | input | output | postcondition | to_ state |
|---|---|---|---|---|---|
| start | | | card, pin | | decp |
| decp | | card, pin | id, pin | id = card.id | drv |
| drv | ∃ client ∈ database : client.id = id ∧ client.pin = pin | id, pin | ok | | dviok |
| dviok | | ok | prompt | | dpw |
| dpw | | prompt | amount | | drw |
| drw | | amount | pr:amount | | dpru |
| drw | | amount | cash_id, pd:amount | | dprl |
| dpru | | pr:amount | cash | cash_counter' = cash_counter - amount | drc |
| dpru | | | cust_id, pd:amount | | dprb |
| dprl | | | pr:amount | | dprb |
| dprl | | cust_id, pd:amount | ok | client.deposit' = client.deposit - amount | dda |
| drc | | | cust_id, pd:amount | | drcl |
| dprb | | pr:amount | cash | cash_counter' = cash_counter - amount | drcl |
| dprb | | cust_id, pd:amount | ok | client.deposit' = client.deposit - amount | ddau |
| ... | ... | ... | ... | ... | ... |

Figure 4: Automaton with process control states

Then, these basic components are composed to a state oriented view of the corresponding business process as a whole. Figure 4 shows part of an example, where channel identifiers pr = (process_request, retrieve_cash) and pd = (process_request, decrease_account) are introduced to resolve ambiguities in the output of message "amount" of activity process_request. From this process description, the automata for describing complete system behavior are derived.




| name | precondition | input | output | postcondition |
|---|---|---|---|---|
| auth_ok | ∃ client ∈ database : client.id = id ∧ client.pin = pin | id, pin | ok | ∃ client ∈ database : client.id = id ∧ client.pin = pin |
| auth_nok | ∄ client ∈ database : client.id = id ∧ client.pin = pin | id, pin | nok | ∄ client ∈ database : client.id = id ∧ client.pin = pin |
| book_ok | ∃ client ∈ database : client.id = id | id, amount | ok | ∃ client ∈ database : client.id = id ∧ client.deposit' = client.deposit - amount |

Figure 5: Example automaton for component database

Furthermore, those basic activity components belonging to the same process instance and the same role are composed to yield a component corresponding to a role's behavior in a specific process instance. Figure 5 shows the behavior of component database in our example process.

## 3.3 Data and Sorts in the Process Model

In the process model, data occur associated with input and output channels of components, or as data states of basic components. In both cases, data are usually typed. Therefore, we associate a sort as described in [Het96] with every channel of the process model (and thus with the message transmitted by it), as well as with the data states of basic components.

# 4 Methodological Aspects

The introduction of new software to an existing business organization should not only automate existing practices, but rather bring about real improvements in processing and efficiency. Therefore, system development and business process reengineering have to be dealt with hand in hand in a systematic way. Integrated approaches require the combination of a solid know-how both in software development and in economics and business process reengineering.

In the following, we introduce a business process driven approach to software development, as well as a systematic method for business reengineering. The integration of these techniques is subject of further research.

## 4.1 Business Processes in Software Development

In the SysLab methodology of system development, we employ business process modeling for identifying and documenting *typical* system behavior in a global, task oriented way. In parallel, we document those data objects that are relevant and necessary for specifying the process model. Focussing on *typical* exemplaric system behavior rather than a generic specification of *complete* system behavior helps to keep the process model simple while still expressing the most important aspects.

Note that at this stage, we do not yet consider *how* these activities will be carried out, as a too early focus on established execution mechanisms endangers the potential for improvements and change that is connected with a systematic integration of business process reengineering and system development.

Based on the business process model, we derive interface definitions for the efficient, process oriented support of specific working places. First, we extract those activities of the process whose roles are associated with the working place under consideration. From this subset, the activities directly communicating with the software system define the order, context and functionality of the system services that will be executed from this specific working place. In addition, the information that is input to (or output of, respectively) a system service, is already documented in the process model.

To ensure that the static structure of the system supports the behavior of the system in an optimal way, we sketch a first design of our system's component structure only *after* gaining a well founded understanding of the *typical* system behavior in its global context by developing a first draft of the process model. As modeling progresses, process model and component structure are developed and refined iteratively in an integrated way.

When the system's component structure turns stable in principle, we progress from the modeling of *typical* to the more complex and technical modeling of *complete* system behavior. As the description technique used for complete behavior modeling should especially accomodate the needs of the system developer with regard to expressiveness for generic complete behavior description, formality and tool support, in SysLab we turn to state transition diagrams [GKRB96] for specifying the complete behavior of system components. As both description techniques for system behavior are semantically integrated with respect to the mathematical system model, the information specified in the business process model is a start-up for the state transition diagrams.

The addition of error handling is an important step in the completion of a system's behavior description. Here, we use the previously defined business processes for schematically detecting possible errors in system behavior. Prime candidates for detecting errors are the events in a process. For example, a necessary output event is not issued by an activity within the required time interval. Or, an illegal output is issued that has to be dealt with by some exception handling routine to be defined.

State transition diagrams describe complete system behavior on a level of detail which is already close to implementation. In fact, with an adequate tool support, code for prototyping can be generated from state transition diagrams.

At different stages in the software development process, the business process model can be used to check the intermediate results of the development process on quality and adequacy. For example, the simulated execution of a business process is employed to verify the adequacy of an interface prototype. Similarly, a business process specified during requirements engineering as desired system behavior is used as a testcase in the integration test of the software system.

## 4.2 Overview over the Business Reengineering Method

As in requirements engineering, the integration and involvement of expert users is a key factor for the success of a business reengineering project. The business reengineering method sketched here is derived from a corporation with Siemens-Nixdorf AG. It is described in more detail in [Thu97]. In our approach, we embed business process reengineering in a global reengineering methodology to ensure an integrated realization of both quick, continuous improvements and more radical, complex changes in process and organization structure. Both is necessary: early improvement results keep employees motivated for the change effort, and radical changes enable innovation [Dav93a].

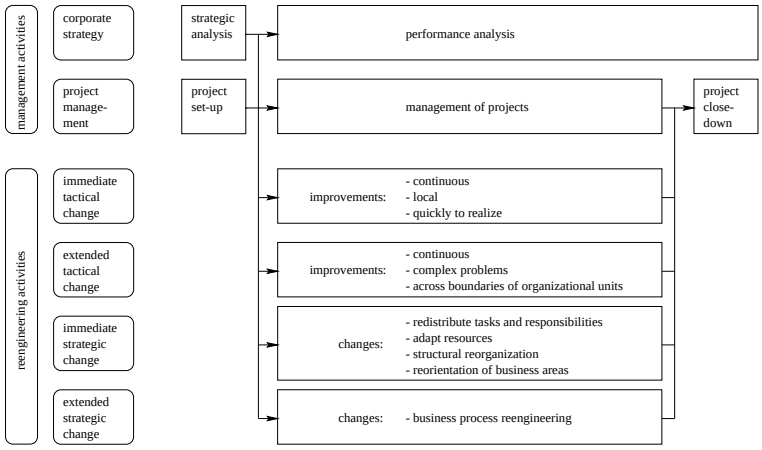



Figure 6: Embedding BPR in our methodology for business reengineering

To achieve this, we structure our reengineering activities into four subtask domains, symbolized in Figure 6 by rectangles with rounded corners in columns at the left. Each task domain is put into practice by one or more processes.

*Tactical improvements* concentrate on detailed business activities and deal with local technological and operational changes, without impact on the business

organization as a whole. By *immediate tactical changes*, we categorize small scale improvements, tackling any problem where both its cause and a possible solution are somewhat obvious, and the solution is easy to implement. In contrast to this, *extended tactical changes* cover long-term improvements which require a higher degree of error recovering and problem solving.

As *strategic improvements* are closely related with business objectives and a business's strategy of achieving them, they tackle global organizational and process problems. *Immediate strategic changes* are a treatment for problems arising from flaws in the organization of an enterprise, e.g. due to the existing distribution of competence, responsibility and tasks. Finally, the task domain of *extended strategic changes* is the most complex of the four, as it incorporates and triggers all other reengineering tasks, the introduction of information technology, and other changes that extend over a rather long period of time.

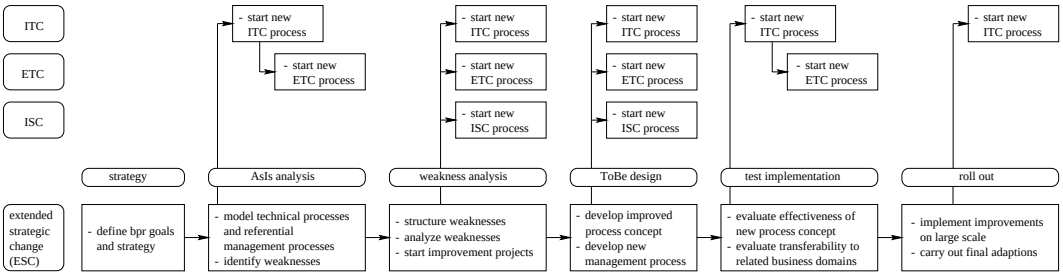



Figure 7: Overview over methodology for business process reengineering

Figure 7 gives an overview over our methodology for business process reengineering, detailing the BPR process of Figure 6. Phases structuring the process chain are depicted by rounded rectangles, aligned horizontally in a row.

Starting from the major goals of the business system, we identify major processes as a general context and define the boundary of that part of the business system to be analyzed in more detail. We structure our process model according to task domains and phases, as well as by hierarchical refinement. For modeling, we employ a hybrid aproach, integrating both top-down and bottom-up steps to ensure adequate granularity and consistency.

All along with process modeling, any weaknesses in the existing system that are touched in interviews or discussions are collected and documented. They are supplemented by weaknesses specific to processes, which are identified by structured process evaluation and evaluation workshops, both of which are based on the process model previously developed. In a next step, these weaknesses are structured into related groups. Complex weaknesses are then analyzed to separate mere symptoms from problem causes, using creative problem solving techniques derived from [Van88], to render possible solutions that tackle problems at their roots, rather than treating mere symptoms.

According to our methodology, design and testing of changes and improvements are closely linked and integrated in an iterative process, which allows even roughly sketched concepts to be evaluated, before they have to be worked out in detail. For all improvement measurements, it is evaluated wether the expected chance of its success and its estimated gain is worth the cost of the improvement project. Measuring the progress of improvement projects supports early discovery of problems in improvement concepts, and thus is the basis for decisions on the further realization of improvement projects.

# 5 Outlook on Current and Future Work

To reduce redundancy in a process model, a new version of the description technique will introduce a class concept for activities. There, activities that occur multiply in one or several processes are grouped into an *activity class*, in which the features common to all instantiations of this activity are predefined. A process will then consist of a set of uniquely identifyable *activity instances*, which are based on a set of defined activity classes.

Furthermore, for the description technique for business processes, the formal foundation with respect to the mathematical system model will be worked out in greater detail, extending also on the relationship between business process diagrams and state transition diagrams already sketched in section 3.

The integration of detailed, quantifyable management mechanisms and the technical aspects of business reengineering and software development into a more plannable, integrated business reengineering and software development process is subject of further research, which will be embedded in the interdisciplinary project FORSOFT.

# A Definition of Concrete Textual Syntax

For defining the concrete textual syntax of the description technique for business processes, we use the Extended Backus-Naur Form EBNF as defined in [BFG+93]. The nonterminals ⟨activity-id⟩, ⟨process-model-id⟩, ⟨data-id⟩, ⟨switch-id⟩, ⟨role⟩, ⟨sort⟩, ⟨expression⟩, ⟨predicate-expression⟩ and ⟨component⟩ are not specified in a more detailed way in this paper.

⟨process_model⟩ ::= **process model** ⟨process-model-id⟩ = **{**
  {⟨activity-id⟩ //,}*
**}**

⟨process⟩ ::= **process** ⟨activity-id⟩ = **{**
  {⟨activity-id⟩ //, }*
**}**

⟨black_box⟩ ::= **black box** ⟨activity-id⟩ = **{**
  **type** **internal** | **external**
  **role** {⟨role⟩}
  {**out_switch** {⟨switch-id⟩ := ⟨predicate-expression⟩ //,}* }
  **input** {⟨event⟩ //,}*
  **output** {⟨event⟩ //,}* | {⟨switch-id⟩: {⟨event⟩ //,}* //;}*
**}**

⟨glass_box⟩ ::= **glass box** ⟨activity-id⟩ = **{**
  **reads** {⟨data-id⟩ **from** ⟨component⟩ **where** ⟨predicate-expression⟩ //,}*
  **changes** {⟨data-id⟩ **from** ⟨component⟩ **where** ⟨predicate-expression⟩
  **to** ⟨expression⟩ //,}*
  **pre** ⟨predicate-expression⟩
  **post** ⟨predicate-expression⟩ | {⟨switch-id⟩: ⟨predicate-expression⟩ //;}*
**}**

⟨refinement⟩ ::= **refinement** ⟨activity-id⟩ = **{**
  **subactivities** {⟨activity-id⟩ //,}*
  **subinput** {⟨event⟩ = {⟨event⟩ //,}* //;}*
  **suboutput** {⟨event⟩ = {⟨event⟩ //,}* //;}*
**}**

⟨event⟩ ::= (⟨sort⟩, ⟨activity-id⟩, ⟨activity-id⟩)

## Acknowledgements

Thanks go to Prof. Dr. Manfred Broy for the possibility of working on this doctoral thesis, and to him and all my colleagues for many interesting discussions and cooperative efforts. Furthermore, I thank Dr. Günther Klementz and his department at Siemens-Nixdorf for continuous support. Especially, I am highly indebted to Michael Adler and Andre DeLeeuw for my integration into their business reengineering team, and for many fruitful discussions.

## References

[BDD+93] M. Broy, F. Dederichs, C. Dendorfer, M. Fuchs, T.F. Gritzner and R. Weber. The Design of Distributed Systems – An Introduction to FOCUS. Technical Report TUM-I9202-2, SFB-Bericht Nr. 342/2-2/92 A, Technische Universität München, München, January 1993.

[BFG+93] M. Broy, C. Facchi, R. Grosu, R. Hettler, H. Hußmann, D. Nazareth, F. Regensburger, O. Slotosch and K. Stolen. The requirement and design specification language SPECTRUM – An informal introduction, Part II. Technical Report TUM-I9312, Technische Universität München, München, May 1993.

[BHH+] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe and V. Thurner. Towards a Formalization of the Unified Modelling Language. to appear at ECOOP'97, Jyvskyla, Finland, June 1997.

[BJR96] G. Booch, I. Jacobson and J. Rumbaugh. *Unified Method Language – UML Semantics.* Rational Software Corporation, Santa Clara, CA., 1.0 edition, January 1996.

[Boo94] G. Booch. *Object-Oriented Analysis and Design.* Benjamin Cummings, 1994.

[CAB+94] D. Coleman, P. Arnold, S. Bodoff, C. Dollin, H. Gilchrist, F. Hayes and P. Jeremeaes. *Object-Oriented Development – The Fusion Method.* Object-Oriented Series. Prentice-Hall International, Inc., Englewood Cliffs, New Jersey, 1994.

[CCT90] CCTA, NCC Blackwell. *SSADM Reference Manual*, 4th edition, 1990.

[Dav93a] T.H. Davenport. *Process Innovation – Reengineering Work through Information Technology.* Harvard Business School Press, Boston, Massachusetts, 1993.

[Dav93b] A.M. Davis. *Software Requirements – Objects, Functions, and States.* Prentice-Hall International, Inc., Englewood Cliffs, New Jersey, 1993.

[DeM79] T. DeMarco. *Structured Analysis and System Specification.* Prentice-Hall International, Inc., Englewood Cliffs, New Jersey, 1979.

[GKRB96] R. Grosu, C. Klein, B. Rumpe and M. Broy. State Transition Diagrams. Technical Report TUM-I9630, Technische Universität München, München, June 1996.

[GR96] R. Grosu and B. Rumpe. Cuncurrent Timed Port Automata. Technical Report TUM-I9533, Technische Universität München, München, October 1996.

[GRK96] R. Grosu, B. Rumpe and C. Klein. Enhancing the SYSLAB System Model with State. Technical Report TUM-I9631, Technische Universität München, July 1996.

[Het96] R. Hettler. Description Techniques for Data in the SYSLAB Method. Technical Report TUM-I9632, Technische Universität München, München, October 1996.

[Jac92] I. Jacobson. *Object-Oriented Software Engineering – A Use Case Driven Approach.* Addison-Wesley Publishing Company, Reading, Mass., 1992.

[RBP+91] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen. *Object-Oriented Modelling and Design.* Prentice-Hall International, Inc., London, 1991.

[RKB95] B. Rumpe, C. Klein and M. Broy. Ein strombasiertes mathematisches Modell verteilter informationsverarbeitender Systeme - Syslab Systemmodell -. Technical Report TUM-I9510, Technische Universität München, March 1995.

[Thu97] V. Thurner. "Making it Their Idea" – A Case Study: Customer Participation and Commitment in BPR. to appear at the SCI'97 focus symposium on BPR, Caracas, July 1997.

[Van88] A.B. VanGundy. *Techniques of Structured Problem Solving.* Van Nostrand Reinhold, New York, 2 edition, 1988.