Report on the Seminar

Process-oriented Information Systems

Editors

Prof. Dr. Mathias Weske
Dipl.-Wirt.Inform. Jens Hündling
Dipl.-Wirt.Inform. Hilmar Schuschel

Contact
Hasso Plattner Institute for Software Systems Engineering
at the University of Potsdam
Chair for Business Process Technology
P.O. Box 90 04 60
D-14440 Potsdam, Germany
Editorial

In the summer term 2003 the seminar Process-oriented Information Systems was held at the Hasso Plattner Institute (HPI) for Software Systems Engineering at the University of Potsdam. This collection of papers represents the topics in this seminar. Master students of Software Systems Engineering at HPI prepared the material presented in the seminar and summarized in this report. The basic idea of this seminar was to gather knowledge in the field of process-orientation, and especially processes executed within complex information systems. Furthermore, the students dealt with to conceptual as well as to practical aspects of process technology. The seminar was organized by the Business Process Technology group headed by Prof. Dr. Mathias Weske and was composed of ten topics. Each student handled one topic under supervision of an advisor. The topics where presented by the students at regular seminar meetings, in June and July 2003.

Recently, process-orientation is a major topic for all kinds of businesses and is fostered in different areas of business computing. Additionally, information technology is already used to perform a large portion of the routine work, and – in a variety of application scenarios – the necessity for human interaction is weakening. Since processes reflect the knowledge of how to conduct business within and between companies, efficient and effective information systems are needed to manage processes. This knowledge can be explicitly modelled in process-aware components of the information systems. Nevertheless, still a large amount of the process knowledge is hidden in applications, the IT infrastructure itself and well as in business rules that are not electronically available. Process knowledge and management is needed for economic success today, which in turn is more dependent than ever upon the efficient implementation of business processes, their optimization, and their integration with those of business partners.

This collection of paper starts with Marc Förster introducing a formal method for business process modelling called Pi-calculus. Formal process methods promise to enhance the integration of processes in different business areas significantly, at the same time allowing to strictly prove certain properties of a process, such as the absence of deadlocks or behavioural equivalence of processes with different structure. Next to the theoretical concepts of Pi-calculus, the paper shows by example that the calculus meets essential requirements for being the foundation of a formal business process modelling language. Catharina Gramlich continues with a paper on "Business Process Analysis", where she explains the necessities of efficiently analyzing processes with regard to errors and flaws. Insufficiently or even erroneously designed business processes induce additional costs. The paper gives an overview of the properties that are relevant for business process modeling. The soundness criterion for verifying the correctness of workflow nets is presented and the software tool Woflan for analyzing workflows based on soundness is introduced. "Flexibility In Workflow Management Systems" is concerned in the paper of Christian W. Günther. Especially in the Workflow domain it is crucial for successful businesses to have a mechanism allowing for change of business processes without halting the whole system. However, implementation of viable mechanisms for handling such change faces critical problems and open questions. The paper focuses on the particular problem of transferring running workflow instances to a newly changed process definition.

Process execution in a Service Oriented Architecture based on Web Services is presented next. In Hagen Overdick's "Implementation Concepts for Workflow Patterns", the need for explicitly modeling business processes in such an environment is explained, the requirements elaborated, and a process execution engine based on Workflow Patterns is presented and discussed. Thereby, the paper provides
a generic platform able to cope with the requirements of the Service Oriented Architecture and flexible enough to map arbitrary workflow languages.

The following paper "Prolegomena to Software Product Lines" by Sebastian H. Schenk introduces Product lines. Products sharing many features and differing only by some degree are developed together from a set of generic assets which are reused through all products and are capable to handle common as well as variable features. The sound principles of product lines have already been transferred and adapted to the development of software products. The paper explains the core concepts, the phases of the associated software development process, characterizes currently used methods and identifies benefits, risks and problems.

Peter Aschenbrenner in "Interactive Groupware Systems" classifies interactive cooperative systems in context with Computer Supported Cooperative Work (CSCW) and Groupware. The centralized and replicated architecture variants of these systems are discussed with special regard to the requirement of interactivity. For the replicated architecture variant synchronization strategies are introduced and exemplified by the Network Text Editor."A new Approach to Business Process Support: Case Handling" by Jakob Magiera explains why workflow management systems and the workflow approach do not provide the required flexibility needed especially in information-intensive business processes. The paper points to the weaknesses of traditional workflow management and summarizes how the approach of case handling as implemented in the FLOWer software can amend them.

The usage of well known planning algorithms from the AI domain for modeling business processes is elaborated in the paper "Using AI Planning Algorithms To Support Business Process Modeling" by Harald Meyer. Since the development of business process models is very labor-intensive, planning algorithms can reduce the costs by automatting the ordering of activities in the process. The main focus lies on the selection of an appropriate algorithm.

Antje Rogotzki's paper "Workflow in ERP Systems" surveys the concepts and usage of workflow realization in ERP systems. The considerations are based on a SAP R/3. After a brief introduction to the concepts of the R/3 architecture, SAP business workflow is explained. The second part concentrates on workflow modeling in R/3 and describes how control-flow and data-flow are defined; an example workflow and its implementation in R/3 illustrates their usage.

This collection ends with Anne Rozinat's paper on "Process Mining". The traditional approach of introducing workflow management to support an existing business process requires a time-consuming design phase for modelling this process before it can be realised and enacted. Process mining can be used to support design or redesign of workflow models by extracting information from log data of real executions within an information system. The paper presents a technique for process mining using log-based ordering relations to construct the workflow mode and demonstrates a mining tool.

After each session and in the end of the seminar open topics were discussed and it was pointed to potential problems. Furthermore, potential impacts and interdependences of the different approaches and aspects of the presented topics were analyzed. For instance, process mining and process analysis seems to be closely related and aim at similar targets but have very different starting points and assumptions. Another example is the lack of flexibility of classical workflow management, which is explained in several of the presented papers, and it was discussed how it is assessed and how flexibility is enhanced by the different approaches.

This collection of seminar papers is the second in an ongoing series and follows the Bachelor Seminar Web Services for B2B Integration [1] from the Summer Term 2002. Provided with a set of research papers and technical white papers, students prepared oral presentations as well as written presentations. Presentations had conceptual as well as practical parts, the latter of which included hands-on experience in the relevant software systems. The seminar provided a nice forum for discussions. It has to
be noted that both the language of the written and oral presentations was English. All participants did very well in coping with this challenge. The members of the Business Process Technology group wish to thank the students for their interest and their commitment, which made this teaching event particularly fruitful for both advisors, and students.

M.W., J.H., H.S.

Potsdam, September 2003

References:

<table>
<thead>
<tr>
<th>Title</th>
<th>Author</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Theory of Business Process Modelling: The Pi-Calculus</td>
<td>Marc Förster</td>
<td>1</td>
</tr>
<tr>
<td>Business Process Analysis</td>
<td>Catharina Gramlich</td>
<td>18</td>
</tr>
<tr>
<td>Flexibility In Workflow Management Systems</td>
<td>Christian W. Günther</td>
<td>34</td>
</tr>
<tr>
<td>Implementation Concepts for Workflow Patterns</td>
<td>Hagen Overdick</td>
<td>56</td>
</tr>
<tr>
<td>Prolegomena to Software Product Lines</td>
<td>Sebastian H. Schenk</td>
<td>70</td>
</tr>
<tr>
<td>Interactive Groupware Systems</td>
<td>Peter Aschenbrenner</td>
<td>100</td>
</tr>
<tr>
<td>A New Approach to Business Process Support: Case Handling</td>
<td>Jakob Magiera</td>
<td>111</td>
</tr>
<tr>
<td>Workflow in ERP Systems</td>
<td>Antje Rogotzki</td>
<td>140</td>
</tr>
<tr>
<td>Process Mining</td>
<td>Anne Rozinat</td>
<td>156</td>
</tr>
</tbody>
</table>
Theory of Business Process Modelling:
The Pi-Calculus

Marc Förster
Hasso Plattner Institute for Software Systems Engineering
Prof.-Dr.-Helmert-Straße 2-3, 14440 Potsdam, Germany
marc@deckfour.com

Abstract. Globalization has produced a degree of competition, and also collaboration, in many sectors of economy going beyond what we have known in the past. Economic success today is more dependent than ever upon the efficient implementation of business processes, their optimization, and their integration with those of business partners. Formal methods in business process modelling promise to facilitate these tasks significantly, at the same time allowing to strictly prove certain properties of a process, such as the absence of deadlocks or behavioural equivalence of processes with different structure. This paper presents the Pi-calculus as a variant of process formalisms, explains its theoretical concepts, and shows by example that it meets essential requirements for being the foundation of a formal business process modelling language.

1 Introduction

Products and services are realized through processes enacted by a single person, a company, or a network of collaborating businesses. In such collaboration people participate as well as information systems and in this perspective, economy today differs not only in quantity but also in quality from the past. “The connectivity of the Internet and the myriad internal networks found throughout most organizations have put the resources and the information needed to accomplish virtually any task no further than a mouse click away.”¹ In order to survive, companies have to deal with the increasing speed of change in business relationships, shorter time-to-market, and shorter product lifecycles. The development of efficient business processes by applying Business Process Management (BPM) techniques is therefore becoming more important than before in ensuring economic success. Since many companies offer their services via Internet BPM also needs standards for specification and integration of Web services, e.g. Web Services Description Language (WSDL), or Universal Description, Discovery, and Integration (UDDI). Formal approaches for modelling business processes could serve as a universal foundation for BPM, so they have been researched intensively.

Many different formal and semi-formal process modelling approaches exist today. This paper presents a formal one, called Pi-calculus, developed by Robin Milner of

¹ [DG01 : 4].
First, the present state of business process modelling will be discussed, as well as possible future developments in this area and the purpose of formal methods in process modelling. Besides, requirements for process modeling techniques are formulated. After this, syntax and semantics of the Pi-calculus are demonstrated along with the explanation of some of its essential concepts. In section four the calculus will be used to model example business processes, and is thereby shown to fulfill the aforementioned requirements. Eventually, the implications for BPM will be discussed and open issues mentioned.

2 Business Process Modelling

There exist a lot of definitions of the business process concept, a sizeable fraction of them stating that business processes are mainly or solely there to create customer value. In this paper a more abstract point of view is assumed, because it can be argued that many activities of businesses are not customer-focused at all, but instead have other important objectives like saving taxes or creating shareholder value. Therefore the following definition seems general enough to encompass most, if not all, business processes:

>A business process is a set of consecutive and/or simultaneous operations that contributes to the realisation of a business objective or of a well-defined business activity.

Note the concurrency property (“simultaneous operations”) present in this definition, which represents an essential aspect of business processes and of their modelling. Although the modelling of business processes has been around for quite a while, the application of formalisms in this area is partly still in the state of research and development. This section wants to give an overview of present shortcomings in business process modelling, point out possible future developments and the role formalisms will have there, and deduce some requirements that a formal business process model has to meet.

2.1 Current Deficits

Business process modelling as it is today resembles data storage before the advent of Relational Database Management Systems (RDBMS). There is no standardized way in which businesses define their processes or workflows, respectively – in general, semi-formal methods are used. Whereas data is decoupled from applications and kept in databases with standardized interfaces, e.g. SQL, business processes are deeply hidden in a company’s various software systems, from comprehensive Enterprise

---

Cp. [RM99]. The explanation of Pi-calculus presented in this paper is based on Milner’s book.

3 Adapted from a newsgroup posting of Jacques Littre of swift.com.
4 The notions of process, business process, and workflow are used as interchangeable throughout this paper.
Resource Planning (ERP) tools to plain document management systems. This can make migration of well-tried processes to new applications a difficult and expensive task. Most software vendors offer their own way of implementing workflows in their products (SAP R/3 uses ABAP with Open SQL on a low level and graphical modelling on a more abstract level), so there is no agreed-upon way of extracting a process out of one application and insert them into another. Even if the translation has been successful, it is generally hard to make sure that the new process formulations have exactly the same properties as the old ones.

2.2 The Future of Business Process Modelling

The so-called “Third Wave” of BPM\(^5\) addresses these deficits and tries to overcome them by aiming at fully formalized process models. In analogy with the evolution of relational DBMS based upon relation algebra, the vision is to have Business Process Management Systems (BPMS) based on process algebra that could deliver process descriptions to various applications and integrate them. This would decouple business processes from applications for the first time and make the easy exchange of one process for another possible. When companies decide to replace one of their software systems by a new one they would no longer have to go through the error-prone proceedings of translating existing workflow definitions to new, and possibly poorly compatible, application or database interfaces.

The value chain of products or services usually spans several companies. Hence, there is also a need to be able to integrate the processes of two or more participants and to check for their compatibility. Having formalized process models would make it possible to do this in a coherent and automatized way. Eventually, if these models were executable directly, one would arrive at a *programming language* for business processes.

2.3 Formal Methods

Since the 1960s functional programming has been associated with an algebraic system called *Lambda-calculus*, devised by Church\(^6\) thirty years earlier. This calculus reduces computation to pure function definition and application and thus forms a kind of universal core language from which functional languages, e.g. LISP, have been derived. “Its importance arises from the fact that it can be viewed simultaneously as a simple programming language in which computations can be described and as a mathematical object about which rigorous statements can be proved.”\(^7\)

In the case of business processes, among the properties to be proven by applying a process modeling language are the absence of deadlocks, or process equivalence. A formal process description method general enough to express all kinds of business actions and interactions could then indeed serve as a common basis for the envisioned

---

\(^5\) Cp. [SF03]


\(^7\) [BCP95 : 1].
BPMS, ensuring interoperability while at the same time making simulation and centralized application control possible.

Until now for processes there exists no mathematical concept that could be seen as canonical in the same way as the Lambda-calculus. Research has been conducted on the application of formal methods to process modelling for a long time, resulting in a great variety of process calculi. They differ not only syntactically but also in their semantics, mainly according to different concepts of action observability and communication, i.e. synchronous or asynchronous, shared memory or message passing. Nevertheless, Pi-calculus has become a buzzword in the BPM community and today is widely understood to be a strong candidate for a universal business process modelling language.

2.4 Requirements for a Business Process Modelling Language

A typical business process is enacted by a network of communicating entities exchanging information, with their actions taking place concurrently (the entities can be humans and also software systems). The behaviour of concurrent systems is inherently non-deterministic since it cannot be predicted in which order atomic actions of the participating processes occur. This non-determinism may be unwanted, but a process modelling language must be able to capture it so that it can be avoided. More often than not a business process is subject to constraints, be it a fixed budget, deadlines, or the availability of resources, so it should also be possible to express such constraints. Companies today usually use products and services of other companies to deliver their products, or services, to customers, and such collaboration makes it necessary to integrate processes of the involved parties. At the same time processes can be important assets of a company that it does not want to reveal to the public or to competitors. To be useful, a business process modelling language must therefore be able to discriminate between internal, unobservable actions and those that can and may be observed from the outside. Normally, a workflow would be published as a black box, where only the behaviour at its interfaces with the environment is specified. The process model should then make it possible to verify that the internal behaviour of the service provider satisfies the specification. In short, a formal business process modelling language must deal with

- communication (through data flow),
- concurrency (and thus non-determinism),
- constraints,
- differentiation between internal and external behaviour,
- process equivalence.

Although other requirements can be thought of, these are certainly among the most important. In the following shall be examined if the Pi-calculus meets them.

---

8 E.g. Fusion Calculus (Parrow), CSP (Hoare), CCS/Pi-calculus (Milner).
3 The Pi-Calculus

While Turing machines and the Lambda-calculus represent computational models of behaviour, the Pi-calculus captures interactional behaviour. Its communication concept does, for the sake of simplicity, not differentiate between processes, channels, and data. Instead, it focuses on “names”, which can be identified with pointers, references, or variables in a computer program. Consider a small example involving a client, a server, and a printer, which is depicted in Fig. 1.

![Diagram of communicating processes](image)

**Fig. 1. Example of communicating processes.** The server holds a link to the printer, which it passes, via channel $b$, to the client, so that the client can access the printer and send data to it.

Now consider the algebraic equations

$$S = b\langle a \rangle S'$$  \hspace{1cm} (1)

$$C = b(c)\overline{c}(d)C'$$  \hspace{1cm} (2)

$$P = a(e)P'$$  \hspace{1cm} (3)

**Printing Process** $= S \parallel C \parallel P$.  \hspace{1cm} (4)

These are equations in Pi-calculus notation. In (1) the server process is defined as putting out the name $a$ over the channel $b$, and then behaving like $S'$. In (2) the client process is defined as receiving a name via the channel $b$ and binding it to its internal variable $c$. Thereafter it sends out the name $d$ over the channel $c$ and goes on behaving like $C'$. In (3) the printer behaviour is defined: it receives a name via channel $a$ and then will behave as $P'$. At last, the printing process (4) is defined to be the concurrent composition of the server, client, and printer processes. According to Pi-calculus semantics the dynamics of interaction in the printing process is as follows:

$$S \parallel C \parallel P = \overline{b}(a)S' \parallel b(c)\overline{c}(d)C' \parallel a(e)P'$$  \hspace{1cm} (5)

reduces to ( $\longrightarrow$ )
\[ \overset{6}{\longrightarrow} S' | \tilde{c}(d) . C' | a(e) . P' . \]

The client has received the name \( a \) over the channel\(^9 \) \( b \) and stores it in the variable \( c \). By this, \( c \) in \( \tilde{c}(d) \) is replaced by its new value \( a \), which represents access to the printer. Finally, the client can send its data, \( d \), over channel \( a \) to the printer, and the communication comes to an end after the corresponding reduction

\[ \overset{7}{\longrightarrow} S' | C' | P' . \]

In Pi-calculus, sub-expressions of a process of the form \( \tilde{x}\{y\} \) (output) or \( x(y) \) (input) are called action prefixes. An input and an output action prefix with the same channel name can, as we have seen, interact when they are part of a concurrent composition of different processes, and thereby the involved process expressions are reduced. This communication concept may seem simplistic at first glance but is capable of modeling an important requirement, namely non-determinism.

### 3.1 Non-Determinism of Reduction

Process expressions beginning with an input or output action prefix may be part of several reductions at once. Consider the example

\[ P = \tilde{x}\{y\} . \text{NIL} | \tilde{x}\{z\} . \text{NIL} | x(w) . \tilde{w}\{v\} . \text{NIL} . \]

In \( P \), the third sub-process can either interact with the first, yielding

\[ P \overset{9}{\longrightarrow} \tilde{x}\{z\} . \text{NIL} | \tilde{y}\{v\} . \text{NIL} \]

or with the second sub-process, yielding

\[ P \overset{10}{\longrightarrow} \tilde{x}\{y\} . \text{NIL} | \tilde{z}\{v\} . \text{NIL} . \]

This non-confluence of reduction models the fact that concurrent processes can give different results depending on the order of internal events, and thus fulfills an important requirement.

---

\(^{9}\) The pair \( \tilde{b} \), in the server process, and \( b \), in the client process, can be seen as the channel’s sending and receiving end, respectively.

\(^{10}\) Of course there may be further reductions possible between \( S' \), \( C' \), and \( P' \), depending on their definitions, but we are not interested in them here.

\(^{11}\) \( \text{NIL} \) denotes the inert process, a process that does nothing. It may be omitted in expressions.
3.2 Process Equivalence

State automata have long been used to model system, or process, behaviour. Automata theory even offers a notion of equivalence, namely language equivalence. This means that automata accepting the same language, i.e. the same input sequences, are regarded as equivalent. It can also be shown that every non-deterministic automaton can be converted to a deterministic one that accepts the same language. Such a conversion is convenient as long as “stand-alone” automata are concerned, but at the same time does not take into account the consequences of inherent non-determinism in concurrent, interacting systems. Let us look at an example, a drink vending machine represented by the state automaton of Fig. 2.

![Fig. 2. State automaton model of a simple drink vending machine.](image)

When the machine is in the start state A, a user can insert a 2-cents coin. The machine will advance to state B, and a button can be pressed to make the machine output a cup of tea and return to the start state. Alternatively, another 2 cents can be inserted and the machine goes to state C where the “coffee” button can be pressed to get a cup of coffee.

The language $\text{Lang}$ of this automaton, if we assume that every state is accepting, is

$$\text{Lang} = \{ (2 \text{ cents}), (2 \text{ cents, tea button}), (2 \text{ cents, coffee button}), \ldots \}$$

and so on. Now, look at a slightly different model:

![Fig. 3. Non-deterministic state automaton representing a drink vending machine.](image)

Although according to automata theory this machine is equivalent to the machine modeled in Fig. 2 it behaves differently when interaction with the environment is taken into account.

---

12 [RM99 : 14-15].
When someone inserts the first coin into this machine it can either go to state E, where another coin must be inserted to get a cup of coffee, or to state F, where one can only press the “tea” button. Having tried this vending machine many users would certainly not regard it to be equivalent to the first one, although both accept the same language. Hence, as soon as interactive behaviour is considered a revised concept of equivalence becomes necessary, more precisely, one in which the two machines would be seen as inequivalent.

### 3.3 Simulation

An automaton $P$ is said to *simulate* another automaton $Q$ if every state transition that $Q$ performs can be matched by a corresponding transition of $P$. This must be true for both in- and outputs: Beginning at the start states, $q_0$ and $p_0$, (1) if $Q$ outputs a value $x$ and advances to the next state then $P$ must also be able to output $x$ and reach the next state. (2) If $Q$ accepts an input $y$ and goes to another state, $P$ must also input $y$ and make the transition to the next state. Being now in the successor states, conditions (1) and (2) must again hold for $P$ and $Q$. Going back to the vending machine, let us call the deterministic machine $P$ and the non-deterministic one $Q$. Does $P$ simulate $Q$?

First, $Q$ accepts the input of 2 cents and makes the transition from state D to E. $P$, in state A, can also accept 2 cents and go to state B. Next, $Q$’s “tea” button is depressed, it puts out a cup of tea and is in the start state again. $P$’s “tea” button can then also be pressed to make it put out tea and return to the start state. Now the machines could again perform steps (12) and (13), but coming from state D there is another transition $Q$ can make, which also has to be matched by $P$:

\[ Q: D \xrightarrow{2 \text{ cents}} E \quad P: A \xrightarrow{2 \text{ cents}} B \]  
\[ Q: E \xrightarrow{\text{tea button}} D \xrightarrow{\text{tea}} A \quad P: B \xrightarrow{\text{tea button}} D \xrightarrow{\text{tea}} A \]

It has been shown that all transition paths of $Q$ can be matched by $P$ and so $P$ can be said to simulate $Q$. But $Q$ does not simulate $P$, as the counter-example of Fig. 4 demonstrates.
Theory of Business Process Modelling:
The Pi-Calculus

Fig. 4. Vending machine $Q$ does not simulate machine $P$. When $Q$ gets to state E and $P$ performs the transition to C by accepting the input of 2 cents, $Q$ cannot match this transition since being in state E it does not accept the same input.

Having developed a criterion by which the two vending machines can be distinguished, this concept can be made formal and extended into an equivalence relation. In order to get more general, the equivalence relation is based on testing for the equivalence of states in a Labelled Transition System (LTS) instead of the comparison of distinct state automata.\textsuperscript{13} Fig. 4 shows the LTS of machine $P$ and $Q$. The result in LTS terminology: “State D does not simulate state A.” If $\text{Act}$ is the set of

\textsuperscript{13} An LTS can be seen as an automaton without a start state and with any state being accepting, so any state can be assumed as start state, resulting in a different automaton. The statement “Automaton $A$ with start state $a_0$ simulates Automaton $B$ with start state $b_0$” is then equivalent to the statement “$a_0$ simulates $b_0$” in the LTS consisting of the combined states and transitions of $A$ and $B$. 

Machine $P$  

Machine $Q$  

Labelled Transition System
input actions that is accepted by an LTS unified with the set of output actions it can perform,\(^\text{14}\) its definition is as follows:

A Labeled Transition System over \(\mathcal{A}\) is a pair \((Q, T)\) consisting of a set \(Q\) of states, and a relation \(T \subseteq (Q \times \mathcal{A} \times Q)\), which is called transition relation. If \(\alpha \in \mathcal{A}\) and \((q, \alpha, q') \in T\) we write \(q \xrightarrow{\alpha} q'\) and call \(q\) the source and \(q'\) the target of the transition.

The simulation concept introduced before is then formally defined as follows:

Let \((Q, T)\) be an LTS, let \(S\) be a binary relation over \(Q\). Then \(S\) is a strong simulation over \((Q, T)\) if, whenever \(p S q\),

\[
p \xrightarrow{\alpha} p' \Rightarrow \exists q' \in Q \text{ such that } q \xrightarrow{\alpha} q' \wedge p' S q'.
\]

We say that "\(q\) strongly simulates \(p\)" if there exists a strong simulation \(S\) such that \(p S q\). Applied to the vending machines, the following relation \(S_{Q,P}\) shows that \(A\) (the start state of machine \(P\)) strongly simulates \(D\) (the start state of machine \(Q\)), because it contains the pair of start states and every pair that results from it by applying the possible state transitions to both members. \(S_{Q,P}\) thus satisfies condition (18):

\[
S_{Q,P} = \{ (D, A), (E, B), (F, B), (G, C) \}
\]

After these preliminaries it is time to introduce the equivalence relation we have aimed at, called strong bisimulation:

A strong bisimulation over the LTS \((Q, T)\) is a binary relation \(S\) over \(Q\),\(^\text{15}\) such that both \(S\) and \(S^{-1}\) are strong simulations.

By this we have arrived at a formal criterion for stating behavioural equivalence of processes with respect to interaction. Since in the LTS corresponding to the vending machines \(P\) and \(Q\) state \(A\) strongly simulates state \(D\), but not vice versa, \(A\) and \(D\) (and hence the behaviour of \(P\) and \(Q\) possessing the start states \(A\) and \(D\), respectively) are not strongly bisimilar and thereby shown to be inequivalent. More formally, this is because the relation \(S_{Q,P}^{-1} = \{ (A, D), (B, E), (B, F), (C, G) \}\) is no simulation. Considering the second and fourth pair, when \(P\) goes from state \(B\) to \(C\), \(Q\) cannot make the transition from \(E\) to \(G\), as would be required.

### 3.4 Observation Equivalence

Strong bisimulation is a concept of behavioural equivalence that requires the respective processes to match each other’s state transitions step by step in an interlocked way. In contrast to such a strict notion of equivalence, when modelling business processes one wants to differentiate between observable actions and actions

---

\(^{14}\) In the vending machine example, \(\mathcal{A} = \{ 2\text{ cents}, \text{ tea button pressed, coffee button pressed, tea output, coffee output } \}.

\(^{15}\) When two processes, \(P\) and \(Q\), are strongly bisimilar we write \(P \sim Q\).
that are internal to a process and thus cannot be observed from the outside. Business processes should then be regarded as equivalent based only on their externally visible interactions with the environment. This corresponds to the definition of behaviour in respect to an interface: a company should be able to publish its process specifications, to let partners know which inputs they have to give in a collaboration, which outputs they can expect, and in what order. How these processes are implemented internally remains the private knowledge of service providers or service requestors, respectively. By showing the equivalence of interface and internal behaviour in a formalized way companies could prove that their service implementation does indeed conform to the behaviour promised to business partners. Strong bisimulation is therefore not well suited for establishing business process equivalence and has to be relaxed in order to allow for an arbitrary number of process-internal actions in-between external interactions.

Such a less strict observation equivalence concept is indeed offered by the Pi-calculus, namely weak bisimilarity. In it, processes are seen as black boxes. The internal reduction

\[ a.P | a.Q \xrightarrow{\tau} P | Q \]  

(21)

of a process \( X \) is denoted by \( \tau \), the unobservable action, called reaction.\(^{16}\) \( \text{Act} \), the set of possible process actions – inputs and outputs – thus gets an additional member, \( \tau \). In the following, Elements of \( \text{Act} \setminus \{ \tau \} \) will be denoted by \( \lambda, \mu,... \), elements of the complete \( \text{Act} \) set by \( \alpha, \beta,... \). The generalized reaction rule of Pi-calculus is then

\[ P \xrightarrow{\tau} Q \land R \xrightarrow{\lambda} S \Rightarrow P | R \xrightarrow{\tau} Q | S . \]  

(22)

This means that two processes, \( P \) and \( R \), that evolve by a complementary action into \( Q \) and \( S \), respectively, will silently react with each other when composed in parallel.

Until now, the notion of LTS has been somehow unrelated to Pi-calculus expressions. We therefore define:

An LTS of concurrent processes over \( \text{Act} \) (including \( \tau \)) has a set of process expressions \( \mathcal{P} \) as its states.

(23)

According to this, Pi-calculus process expressions play the same role as states in a state automaton or an LTS. The following definitions will therefore refer directly to process expressions rather than Labelled Transition Systems. Fig. 5 illustrates the equivalence of states and process expressions with the LTS of the print server example from the beginning of this section three.

\(^{16}\) From now on, internal variables will be omitted in most cases and only the channel names remain. This does not affect the validity of statements and yields shorter expressions that are easier to grasp. At the same time the omission can be interpreted as a shift of focus from value to event communication.
As has been argued before, an observation equivalence relation has to allow the regarded processes to execute any number of internal reactions accompanying their observable actions. Still, they should be seen as equivalent. If we understand \( \tau \) to denote a sequence of zero or more reactions the definition of weak simulation is as follows:

Let \( S \) be a binary relation over \( \mathcal{P} \), the set of process expressions. Then \( S \) is a weak simulation if and only if,

\[
P \xrightarrow{\tau} P' \Rightarrow \exists Q' \in \mathcal{P} \text{ such that } Q \xrightarrow{\tau} Q' \wedge P' S Q'
\]

\[
P \xrightarrow{\lambda} P' \Rightarrow \exists Q' \in \mathcal{P} \text{ such that } Q \xrightarrow{\tau} Q' \wedge P' S Q'.
\]

The first expression of the definition states that a process weakly simulating another can match reactions of the latter with a sequence of one or more, or even no reaction. The second line states that the same process must match the simulated process' observable actions (meaning input and output actions, except reactions) with the same action, accompanied before and after by reaction sequences of arbitrary length. This is exactly the needed relaxation of the simulation concept.

In the same way strong bisimilarity (20) is based on strong simulation (18) weak bisimilarity is based on weak simulation:

A binary relation \( S \) over \( \mathcal{P} \) is a weak bisimulation if both \( S \) and \( S^{-1} \) are weak simulations. \(^{(25)}\)

Like strong bisimulation, weak bisimulation (or observation equivalence) is an equivalence relation, i.e. it is reflexive, symmetric, and transitive. Obviously, the set of weak bisimulations is a superset of the set of strong bisimulations: Whenever processes are strongly bisimilar they are also equivalent in respect to observation.

\(^{(17)}\) We say that two processes, \( P \) and \( Q \), are weakly bisimilar (written \( P \approx Q \)) when there is a weak bisimulation \( S \) with \( P S Q \).
4 Exemplary Formal Modelling of Business Processes

After having demonstrated some basic syntax and semantic rules, and also the central equivalence concepts of the Pi-calculus, two (very) simple examples of business processes will be introduced, modelled, and shown to be weakly equivalent.\footnote{The examples are both taken from [RM99 : 61-63], with the second one modified slightly to reduce the number of states to consider in stating weak bisimilarity.}

Imagine a workshop with two workers on an assembly line. A conveyor belt, represented by the channel $i$, brings them jobs (the jobs are not done in collaboration), and after completion the workers put them onto another conveyor belt, represented by the channel $o$. A job can be either easy ($E$) or difficult ($D$), but the workers are well trained and treat all jobs the same. Thus, the definition of a worker is

$$W = i_E W' + i_D W'$$

where $W'$ is the initial state. According to this definition a worker takes a job input, easy or difficult, completes it (which is not really contained in the equations) and outputs it again. After that she is ready to take a new job. The workshop itself is just a parallel composition of the two workers:

$$Workshop = W | W.$$  \hspace{1cm} (27)

Fig. 6 shows the LTS of the workshop. From the initial state on top of the figure, if a job is accepted, be it easy or difficult, there are two possible transitions, depending on who of the workers takes the job. The system can then accept another job through the other worker or return to the initial state by outputting the job. When both workers are busy one of them can put back the finished job, and then the other worker can output hers or (if she takes longer) the first worker takes a new job and both are busy again.

Fig. 6. LTS of the workshop, showing the possible state transitions. Since there are only observable actions in the model it can be seen as the public specification of a business process.

\footnote{The “+” sign denotes alternatives, i.e. when one is chosen the others are discarded.}
Unfortunately, the workshop belongs to a bigger company, gets outsourced, and the workers are fired. The new service provider employs two jobbers that are less experienced than the workers and need a hammer to complete the difficult jobs. Alas, for reasons of cost cutting there is only one hammer that the jobbers must take turns in using. The hammer can be picked up \((gh)\) by a jobber and must be put down \((ph)\) before the other can use it – just like a semaphore. Its definition is

\[
H = gh.H' \\
H' = ph.H.
\]

The definition of a jobber gets more complex than that of a worker:

\[
J = i_E J_E + i_D J_D \\
J_E = o_J \\
J_D = gh.ph.J_E.
\]

When doing a difficult job, a jobber must now first pick up the hammer (i.e. perform the complementary action of the hammer’s \(gh\)) and put it down after finishing before he can output the job. The whole job shop system is

\[
\text{Job Shop} = J \mid J \mid H
\]

Whereas the workshop only had four states the job shop has fifteen, which can be boiled down to eight \((X \text{ and } Y \text{ can both take the values } E \text{ or } D)\):

(i) \(J \mid J \mid H\)

(ii) \(J_E \mid J \mid H\) (two combined states)

(iii) \(J \mid J_E \mid H\) (two combined states)

(iv) \(J_E \mid J' \mid H\) (four combined states)

(v) \(ph.J_E \mid J \mid H'\)

(vi) \(J \mid ph.J_E \mid H'\)

(vii) \(J_E \mid ph.J_E \mid H'\) (two combined states)

(viii) \(ph.J_E \mid J_E \mid H'\) (two combined states)

If we define \(\mathcal{P}\) as a set of process expressions to contain the states of the workshop as well as those of the job shop it is possible to construct a binary relation \(\mathcal{S}\) over \(\mathcal{P}\) that contains the pair \((\text{Workshop}, \text{Job Shop})\) and is a bisimulation, consisting of the fifteen (!) pairs

(a) \(W \mid W', \quad J \mid J \mid H\) (i.e. \(\text{Workshop}, \text{Job Shop}\))

(b) \(W \mid W', \quad J_E \mid J \mid H\)

(c) \(W \mid W', \quad J \mid J_E \mid H\)

(d) \(W \mid W', \quad ph.J_E \mid J \mid H'\)
(e) $W \mid W', \quad J \mid \overline{p}hJ_E \mid H'$

(f) $W' \mid W', \quad J_E \mid J_Y \mid H$

(g) $W' \mid W', \quad J_E \mid \overline{p}hJ_E \mid H'$

(h) $W' \mid W', \quad \overline{p}hJ_E \mid J_X \mid H'$

Obviously the workshop process matches the job shop’s unobservable reactions with no action at all, in compliance with the bisimulation definition (24, 25). Take for instance the job shop transition from state (b) with $X = D$ to state (d) in the bisimulation listed above. The transition is a reaction, namely taking up the hammer, and the workshop at the same time just remains in its $W \mid W'$ state. By exhibiting this relation it is now formally proven that $\text{Workshop} \approx \text{Job Shop}$. The service provider can be reassured that their implementation of the service, although different from the original workshop process, conforms to the workshop specification, i.e. does indeed behave equivalently towards the service requestor.

5 Conclusion

This paper has presented some essential aspects of the Pi-calculus as a formal method for process modelling and has put it into the context of business process modelling. Although the workshop/job shop example is simple enough it has served to demonstrate the power of a formal conceptual framework in offering the possibility to rigidly state (business) process equivalence in respect to observable behaviour. It has been shown that the Pi-calculus meets essential requirements of a basis for the modelling of business processes formulated in section 2.4. Data flows can be modelled as well as event communication; Pi-calculus respects the non-determinism inherent in concurrent, interactive systems and differentiates between externally observable and internal behaviour, a requirement that classical behavioural models in the form of state automata do not fulfill; observational equivalence of different process formulations can be stated in a mathematically consistent way. Furthermore, a simple mechanism for expressing constraints, in the form of a semaphore, has been demonstrated. Time constraints as an important class of business process constraints, though, are absent. Research is being conducted in this area, trying to express deadlines by Pi-calculus constructs, which would be also central for models of real-time software systems.\textsuperscript{20}

Petri nets have been used for a much longer time than algebraic methods like Pi-calculus to formally model processes and in many aspects possess similar expressive strength. Compared to Pi-calculus expressions though, they have a fixed connection structure and thus lack the possibility of dynamically changing their behaviour by interaction (think of modeling the outsourcing of business processes where this could be useful). Since data flowing between processes may, in Pi-calculus, represent whole processes itself such dynamics can be expressed. Currently, dynamic Petri nets are an

\textsuperscript{20} Cp. [JC].
active area of research so the deficiency in their dynamics seems to be alleviated in the future.\textsuperscript{21}

The many variants of process calculi make it difficult to see if one of them (and also, which one) will become canonical and lead to a truly universal process modelling language that could be compiled or interpreted and executed directly. For the nearer future the envisioned Business Process Management Systems will probably have to deal with a variety of process dialects. There is already a Turing-complete programming language based on Pi-calculus available, called PICT, but it is still a research project.\textsuperscript{22}

A number of rivalling industry initiatives exist, aiming at a standard for business process modelling, such as the Business Process Modelling Initiative with their Business Process Modelling Language (BPML), or the consortium of IBM, Microsoft, and BEA with the Business Process Enactment Language (BPEL). Since the BPEL group comprises three of the biggest players in the area of BPM they are currently seen as candidates for setting the future standard. Predicted by the protagonists of the Third Wave, Business Process Management Systems will, when they become reality, allow for new rationalization gains and accelerate the pace of global economic integration even more. Hence, formal business process modelling is at present a very active area of scientific research combined with strong standardization efforts from the industry. In what time frame these will lead to business process servers, real products in the sense they can be bought to provide applications from a wide range of manufacturers with process definitions via a standardized interface, remains to be seen.

References


\textsuperscript{21} Cp. [MO00].

\textsuperscript{22} Cp. [PS].
Theory of Business Process Modelling:
The Pi-Calculus


Business Process Analysis

Catharina Gramlich

Hasso-Plattner-Institute for Software Engineering
University of Potsdam, Germany
Catharina.Gramlich@student.hpi.uni-potsdam.de

Abstract. Business processes and workflows have gained an increasing importance as a means for the management of companies. Insufficiently or even erroneously designed business processes induce additional costs and therefore constitute a high risk of inefficiency. Because of the complexity of processes, support by information technology systems has become a crucial factor for efficiently analyzing processes with regard to errors and flaws. This paper gives an overview of the properties that are relevant for business process modeling and examines Petri nets as a well-suited approach for modeling and analyzing workflows. The soundness criterion for verifying the correctness of workflow nets is presented. Furthermore Woflan, a software tool for analyzing Petri nets using soundness as criterion, is depicted by an exemplary workflow net diagnosis.

1 Introduction

Business processes constitute an important approach for understanding the work conducted by companies. A business process describes the general activities done to produce a good or deliver a service. Derived from such processes are the workflows that are actually implemented by the company. There exist a number of information technology solutions to support the definition, execution and monitoring of workflows. Because of the high complexity of business processes, these workflow management systems have become increasingly important.

In order to achieve a sustainable competitive advantage by reducing costs and improving the quality of products and processes, it is vital to check workflows with regard to errors and potentials of improvement. Since workflows often comprise a large number of individual activities and resources, it is desirable to use workflow management systems for automatically analyzing workflows. Therefore there is a need for tools in this area. In order to prevent erroneously or unfavorably defined workflows from being used in practice, the analysis has to be done before a workflow is put into operations.

An analysis consists of several steps. At first an understanding of the activities a company carries out has to be gained. It is therefore necessary to create a model of the observed processes using a modeling language appropriate for the process that is to be described. The resulting model describes the characteristics and objectives of the process and serves as a foundation for analysis. The actual analysis phase then deals with ways of how to dissect a modeled process in order to find errors. It is also desirable to
able to detect possibilities of enhancing a process to better fit to its demands. Because of the strong link between modeling and analysis it is vital to choose an appropriate modeling technique that enables the analyst to objectively evaluate the properties of the process.

This paper gives an overview about business process modeling and analysis and focuses on Petri nets as an example modeling technique. The second section deals with the most important terminology used in the field of business processes. Section 3 introduces the concepts needed for effectively creating business process models and briefly mentions some examples for commonly used modeling techniques. The fourth section presents Petri nets as a formal approach of modeling business processes and explains how analyses are facilitated by this technique. Woflan – a software tool for conveniently analyzing Petri net-based business processes – is portrayed in section 5. The last section summarizes the presented issues and mentions the advantages and disadvantages of formal modeling techniques in general.

2 Business process terminology

In the field of business processes there are a number of important terms needing a precise definition. This section presents a notion of the terms widely used in this area and tries to integrate business process modeling and analysis into its context.

2.1 Business processes and workflows

Business processes constitute a modern perception on the operations of a company. Conventionally a company was depicted as being fragmented into separate functions or departments. It became obvious that, especially for large corporations, such a disjointed point of view is no suitable approach because it does not help well enough to explain how the company jointly works on producing an outcome.

The notion of business processes opposes this traditional view. A business process is defined as “a set of activities that, taken together, produce a result of value to a customer” [5]. In contrast to other views, the company is treated as a network of actors and activities that together aim at delivering a product or service to the customer.

Examples for general business processes are “order fulfillment”, “insurance claim treatment” and “selling a product to a customer”. It is also possible to structure a complex process into a number of sub-processes and thereby build a hierarchy of activities. The sub-processes are then modeled and analyzed individually. “Selling to a customer” might for example be separated into “contacting the customer”, “processing the order” and “ invoicing the customer”. The results of one sub-process serve as input for the following sub-process.

Strongly related to the concept of business processes is the notion of a workflow. The term workflow is used for describing the tasks and their coordination, the executing people and other involved resources, as well as the needed input and produced output information that together form a business process. Workflow and business process are thus very similar. It is quite common to use the terms “business process” and “workflow” as synonyms although there are minor differences. A business proc-
ess is a rather abstract and more descriptive notion of how a company works, while a workflow can be considered as an automated business process, i.e., a process that can be executed for example with the help of a Workflow Management System. Computerized support is especially important for complex workflows consisting of several interrelated tasks that need to be coordinated.

2.2 Overview of related practices

A company usually delivers several products and services. Thus a company can identify a set of business processes by analyzing its business domains. In order to gain a thorough understanding of a process, with regard to specifying its characteristics and objectives, a model of the process is built. A complex process is thus separated into manageable units which, taken together, describe the properties and goals of the entire process. The resulting model should comprise the characteristics of the process as precisely as possible while still aiming at creating an understanding for it as a whole. It has to be possible to get an overview of the process, but still the individual tasks, the involved resources and needed information belonging to the modeled process should be identifiable. There is a variety of different modeling methods available, each focusing on different aspects of business processes. Section 3 describes Petri nets as an example for modeling techniques.

The generated process model can then be analyzed for possible mistakes, e.g., by using formal analysis techniques. The superior goal of these analyses is to find a starting point for reducing costs (time and money) brought on by inadequately designed processes. Depending on the chosen method for modeling a business process, there are a number of different analysis techniques available. These methods focus on diverse aspects like performance or feasibility of a process definition.

When errors or problems become obvious during analysis, the process has to be redesigned, e.g., by applying Business Process Improvement or Business Process Reengineering. Both of these approaches share the cross-functional view that is characteristic for the concept of business processes in general. The difference between Improvement and Reengineering lies in the rigor of enforcing changes. While Improvement (e.g., Total Quality Management) advocates incremental adjustments in order to achieve a gradual improvement, Reengineering calls for a “clean slate” beginning. Reengineering is often accompanied by a higher risk of failure, because of the profound transformations to a company’s processes (see [5]). A comparison of the two approaches to business process redesign can be found in [6].

The correlation between the activities mentioned in this section is summarized in the Petri net in Fig. 1. The identified business processes of a company are modeled and analyzed individually. Detected errors are repaired either through Business Process Improvement or Reengineering. Since a redesigned process might still contain errors or can potentially be designed in a better way, redesigning cannot be a one-time activity. Instead, the corrected or changed processes must be modeled and analyzed iteratively.
Since modeling and analysis of business processes are strongly linked together, this paper not only focuses on analyzing but also describes considerations on how to model a process. Refer to [3] for more information about the terminology used in the context of business processes.

3 Modeling Business Processes

As it was pointed out, modeling a business process is the starting point for analyzing it with regard to errors and potential improvements. Not the process itself but its describing model is examined, making the choice of an appropriate modeling method very important. This section presents aspects that have to be taken into consideration when deciding about the modeling method to be employed. Furthermore, Petri nets as a suitable modeling approach are introduced.

3.1 Process modeling methods

Because of the high complexity of most business processes, it is vital to use computerized support when defining, analyzing and executing a workflow. Therefore the market for Workflow Management Systems has evolved over the last few years.
When looking at a business process the following perspectives can be employed (cp. [1]).

- The *functional* point of view that focuses on the individual tasks that have to be completed.
- The *behavioral* point of view refers to the coordination between these tasks, e.g. by describing the order of execution and the pre-requisites that have to be met.
- The *organizational* perspective describes who is actually executing a task and how the involved individuals interact with each other. These individuals do not necessarily have to be human beings but can be all kinds of resources of a company (e.g. machines, single employees or teams of people). Thus the organizational perspective is rather resource-oriented than task-specific.
- The *informational* point of view focuses on the data that is needed, produced or manipulated by executing the tasks.

Table 1 shows examples for modeling techniques focusing on different aspects of business processes.

<table>
<thead>
<tr>
<th></th>
<th>Functional perspective</th>
<th>Behavioral perspective</th>
<th>Organizational perspective</th>
<th>Informational perspective</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event Process Chains</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>UML Activity Diagrams</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Petri nets</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pi-Calculus</td>
<td>x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDEF0</td>
<td></td>
<td></td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>Data Flow Diagrams</td>
<td></td>
<td></td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Role Activity Diagrams</td>
<td>x</td>
<td></td>
<td></td>
<td>x</td>
</tr>
<tr>
<td>Entity Relationship Models</td>
<td></td>
<td></td>
<td></td>
<td>x</td>
</tr>
</tbody>
</table>

Table 1: Examples for modeling methods

It must be guaranteed that the model created using one of the available methods, serves its purpose as good as possible. The overall goal of modeling is to facilitate a complete understanding of the described process. In order to do so, the following features characterizing a model of high quality can be identified:

- *Understandability*: Even people inexperienced with the modeling language or the domain of the modeled process should be able to understand the most important characteristics of the process.
- *Correctness*: The model has to be consistent in itself and avoid errors or infeasible constructs.
- *Expressiveness*: It must be possible to cover all important features of a process. The modeling language therefore has to provide the constructs needed to describe the process or be extensible by new constructs respectively.
3.2 Petri nets as a modeling method

Petri nets as a formal modeling language employ the behavioral perspective on workflows. Models based on Petri nets express the tasks belonging to a business process and how these tasks are coordinated. When using Petri nets it is possible to analyze properties of workflows with tool support and thus find errors or unsuitable constructs. An automatic analysis makes it possible to efficiently deal with large workflows. Therefore, Petri nets constitute an appropriate modeling method for large business processes emphasizing the control-flow between individual tasks. This section gives an intuitive overview on Petri nets and explains how workflow attributes can be mapped onto a Petri net.

A Petri net is a bipartite graph consisting of two types of nodes: transitions and places. Directed arcs connect one transition and one place with each other. Places can be marked with tokens. The state of a Petri net is described by the current marking (i.e. the distribution of tokens over places). A transition is enabled (ready to fire) when all its input places contain a token. The execution of a Petri net is achieved through letting transitions fire meaning that a token from each input place of the firing transition is consumed and on each output place of the transition a token is produced.

Thus a Petri net focuses on the dynamic structure of a process, i.e. it describes the order in which the individual tasks (mapped onto transitions) are executed and how the coordination between the tasks is accomplished. Regarding this control flow, constructs for sequential, parallel, alternative and iterative execution of tasks, as shown in Fig. 2, can be identified. For further details and other examples for such workflow patterns refer to [10].

![Control flow constructs in Petri nets](image)

**Fig. 2.** Control flow constructs in Petri nets

An advantage of Petri nets is the availability of a graphical representation promoting the understanding of the modeled process. Even people not familiar with Petri net theory can intuitively understand at least roughly what is expressed by the net.

Petri nets are backed up by a solid mathematical basis. Petri net theory is a well understood research area so that many analysis methods, e.g. for determining properties or conducting performance measures, have been developed. (Refer to section 4 for examples of analysis techniques; see [9] and [11] to get more information on the mathematical description of Petri nets.)
3.3 Mapping of workflow attributes onto Petri nets

The mapping of a workflow onto a Petri net can be done in several steps depending on the attributes of the workflow that should be included in the model. Transitions, being the active part of a Petri net, are chosen to represent the tasks executed during the workflow. The control flow guiding the execution is represented using the constructs for sequential, parallel, alternative or iterative execution. An important issue is the notion of a workflow net introduced in [14]. A workflow net should contain exactly one input place, a single output place and all tasks must be on a path leading from the input place to the output place. This assumption is useful in the sense that the beginning of a workflow execution as well as the termination of the workflow are clearly specified.

In order to model other aspects of workflows, different extensions can be used. To integrate data in a Petri net and describe how that data flows from one task to another, colored Petri nets are applicable. Here attributes are assigned to tokens making them distinguishable from each other. Thus it is possible to follow the way one specific token, representing a data item, takes in a Petri net. Different resources responsible for executing a number of tasks of a workflow can be depicted by using swimlanes and thus dividing the tasks into distinct subsets. Time can be included in the model by assigning duration to tasks or by using triggers.

In order to cope with very complex workflows consisting of a large number of tasks, a set of transitions can be combined to form a comprehensive transition. Thus a hierarchy of subnets each representing a part of the workflow is created. Following these guidelines a Petri net model describing the important features of a business process can be prepared.

4 Analyzing workflows nets

After the workflow model has been generated, it can be analyzed for errors. When Petri nets have been chosen as modeling method, a number of automatically executable checks are possible. This section briefly describes what aspects of workflows can be analyzed in general and presents soundness as an approach for verifying the correctness of workflow nets.

4.1 Analysis categories

Independent of the chosen modeling language, three major categories can be identified when analyzing a workflow. It is useful to validate a workflow, i.e. check whether it behaves as expected when executed. The second area of analysis is verification, meaning that the correctness of a workflow is decided. This refers to evaluating whether the workflow contains any errors or contradictions that might prevent it from being executed at all or at an optimal level. Furthermore, as a third major analysis category, performance measures can be conducted.

In the case of Petri nets as modeling method for workflows there is a variety of techniques available for analyzing these three domains. In order to validate a work-
flow net it is possible to execute a number of fictitious cases. The received results are then compared with the goals that should have been achieved by the process (see [13]). For conducting performance measures, like calculating the throughput or the average time needed to complete a workflow case execution, it is also possible to simulate the execution of workflow cases or to realize a Markov-chain analysis [7].

Stemming from the underlying mathematical theory, there exists a large number of formal analysis techniques for determining structural and behavioral properties of Petri nets (see [8], [15]). In order to verify the correctness of workflows, it is for example necessary to prove the absence of deadlocks and other errors that might prevent a workflow from being completed. The next section presents the soundness criterion and its usage for verifying the correctness of workflow nets.

### 4.2 Soundness as a verification criterion

For checking the correctness of a Petri net based workflow one can employ the criterion of soundness. This criterion combines three properties: (according to [15], cp. also [14])

- **Option to complete**: It should always be possible to complete a case that is handled according to the process. The absence of deadlocks and livelocks in the workflow definition is guaranteed by this condition.

- **Proper completion**: It should not happen that the workflow process signals completion of a case while there is still work in progress for that case. There must not be any fragments of work left from already completed cases.

- **No dead tasks**: There should be an path through the workflow that executes every task contained in it. This restriction means that every task has a meaningful role in the workflow.

The option to complete demands that, starting from every reachable marking of the workflow net, there must be a sequence of transition firings ending with one token left in the output place.

Proper completion implies that after the execution of a workflow there must be exactly one token in the output place and no other tokens left from that case elsewhere in the Petri net. This condition is sensible with regard to independent executions of cases. A case that has left artifacts anywhere in a workflow net, might illicitly influence a subsequently executed case. Once the output place is reached, there must not be no tasks left over for being executed from that case.

The absence of dead tasks refers to the initial marking and demands that every task contained in the workflow can potentially become enabled.

Checking soundness reveals errors that result in a non-terminating workflow. Errors that prevent a workflow from being completed are for example:

- **Deadlock**: A case gets stuck in some state where it is not possible to execute any tasks while the case is not yet finished.

- **Livelock**: A case is trapped in an infinite loop where it is possible to execute tasks but no real progress is possible.
Such structural errors in workflow nets can be discovered without specific knowledge of the workflow application. There are also errors that can only be found with background knowledge, e.g. a wrong ordering of activities. In the remainder of this paper such errors are neglected. Instead, focus is put on automatically decidable workflow net properties. The following section presents Woflan, an example for an analysis tool that is able to verify the correctness of Petri net based workflows by employing the criterion of soundness.

5 Woflan - Diagnosing Petri Net based workflows

Woflan (WOrkFLow ANalyzer) was developed by the Eindhoven University of Technology, the Netherlands, and is able to analyze Petri net modeled workflows aiming at finding structural and dynamic related errors. The tool accompanied by describing papers is available as freeware at [17]; Version 2.2 was evaluated for this paper.

Woflan itself cannot be used for defining and constructing workflows. That step has to be done using commercial workflow management tools like COSA, Meteor, Protos or Staffware. Woflan provides filters for importing workflow descriptions produced by these tools. The imported workflows can then be checked for errors. As criterion for choosing whether a definition is correct or not, Woflan verifies the soundness of the workflow net.

5.1 The diagnosis process

To implement a verification algorithm for checking the soundness property the following theorem is applied by Woflan:

- A workflow net is sound if and only if the short-circuited net is live and bounded. [11] (A proof of this theorem can be found in [12].)

A short-circuited net is a workflow net extended by an extra transition connecting the output and input place with each other. Liveness means that, starting in an arbitrary marking, all transitions can potentially become enabled. Boundedness requires that for each place there is a natural number \( n \) such that for every reachable state the number of tokens in the place is less than \( n \) [14]. For a Petri net to be bounded, there must not be any places that can potentially be marked with an infinite number of tokens.

The diagnosis process applied by Woflan is depicted in Fig. 3. In order to be able to verify the correctness of a workflow it is necessary that the conditions imposed for a workflow net are fulfilled. Woflan therefore starts with checking statically whether the imported net has exactly one input place, one output place and if there are no dangling tasks contained in the net. If one of these conditions is not met, the diagnosis is aborted and the user will be asked to correct the workflow definition.

When the diagnosis can be resumed, boundedness as well as liveness of the net are checked. When either boundedness or liveness of the workflow net is not given, it can
be concluded that the net is unsound. In that case the diagnosis is aborted and the results established so far are presented to the user. When during one of the analysis steps the first error is encountered, Woflan is able to go on diagnosing the rest of the net in that particular diagnosis step in order to gather more information and maybe find other mistakes.

![Diagram of Woflan diagnosis process](image)

**Fig. 3. Overview of the diagnosis process employed by Woflan**

A goal of Woflan is to present the results of the diagnosis in a way that can also be understood by users who are not familiar with Petri nets and the underlying theory. Moreover the user should be guided to finding and correcting the sources for errors in an unsound net. Woflan therefore does not only present the final result stating whether a workflow is sound or not, but also states the places and transitions that prevented the net from meeting that criterion.

### 5.2 Exemplary workflow net diagnosis

In order to demonstrate which workflow net properties Woflan is able to determine, an example workflow of a simplified book-ordering process is used in this section. The example was modeled using “Protos” by Pallas Athena Process Management [18]. A customer places an order for a book, the publisher receives this order and sends the book and a bill to the customer. When the customer receives book and bill, the bill must be paid (Fig. 4).
Fig. 4. Book ordering workflow

The example contains structural errors that Woflan detects during the diagnosis. All results found by Woflan are presented in a sequence of dialog steps, so that the user can focus on an individual property, and also in a comprehensive list that can be used for identifying the possible causes for errors. The remainder of this section describes the properties Woflan checks for the example workflow.

Checking the workflow net conditions

Woflan at first verifies whether the Petri net fulfills the workflow net conditions. This holds true for the example net since it contains one input place (c1), one output place (c7) and all transitions are on a path from input to output place.

Checking boundedness

A major disadvantage of the soundness criterion is that it is rather difficult to check for a complex workflow net whether it is sound or not. Soundness can be decided for an arbitrary Petri net but this verification is “expensive in terms of time and space complexity. In fact, the problem of deciding liveness and boundedness [which together imply that the net is sound] is EXSPACE-hard” (cited from [11], cp. [2]).

In order to check boundedness efficiently and provide the user with information about potential causes of errors, Woflan implements a number of tests using the following statement:

A workflow net terminates properly, when all places can be covered
- by threads of control,
- by uniform invariants or
- by weighted invariants.
A thread of control specifies that tasks have to be executed in a certain order and thus refers to the routing defined in the workflow net. A single thread of control can be imagined as a sequence of activities performed by an individual resource and coincides with the route a specified token takes in the workflow net.

A place invariant is a weighted sum over the places, staying invariant under each possible transition firing. A uniform invariant is a place-invariant with weights zero and one. Weighted invariants allow weights greater than one. A thorough explanation of these properties and their tests implemented by Woflan is beyond the scope of this paper. See [15] and [16] for more details on the mathematical background of these properties.

If threads of control, uniform invariants and weighted invariants do not cover the workflow process definition, Woflan directly checks whether all conditions are proper by computing the Minimal Coverability Graph (MCG). Based on the MCG the decision whether all conditions are proper is made. The coverability graph for the book-ordering example comprises all markings reachable from the initial state of the workflow. In parentheses the currently marked places are put down, the arcs represent the firing of transitions (Fig. 5). It becomes obvious that it is not possible to reach an end state with only one token in the output place. It can be concluded that the workflow cannot terminate properly and therefore is unsound.

![Initial marking](#)

**Fig. 5.** Minimal Coverability Graph for the book-ordering example

**Checking liveness**

Since for the book-ordering workflow it could already be concluded after the boundedness-check that soundness is not given, Woflan refrains from deciding liveness for the example. However, for workflow nets that are bounded, the liveness property still needs to be checked for concluding whether the net is sound. Woflan implements a liveness check that makes usage of the MCG as well as the Occurrence Graph of the short-circuited workflow net (see [15] for details).
Further properties checked by Woflan (Confusions and Mismatches)

Another property checked by Woflan apart from the soundness tests, is the presence of *confusions* in the net. A confusion (or conflict) means that two or more transitions have different sets of input places, but share at least one of them. This implies that when one transition fires another one might become disabled. In this case there is a real choice between tasks. Woflan is able to detect confusions in workflows and encourages the user to remove the causes for that property. An example for a confusion is shown in Fig. 6. The book-ordering example does not contain any confusions.

![Diagram of possible confusion between task C and D](image)

Fig. 6. Possible confusion between task C and D

*Mismatches* of two types can also be found by Woflan. An AND-OR mismatch, like it is shown in Fig. 7 (a), implies a missing synchronization while an OR-AND mismatch (Fig. 7 (b)) tries to synchronize alternatively executed tasks and therefore represents a danger of having a net that is not terminable under all circumstances.

![Diagram of AND-OR and OR-AND mismatches](image)

(a) AND-OR mismatch  (b) OR-AND mismatch

Fig. 7. Mismatches

Woflan detects an AND-OR mismatch in the example workflow. The tasks “prepare bill” and “send book” can be executed in parallel but are not synchronized afterwards. This is also the cause for the workflow being unsound. This error can be cor-
rected by replacing the OR-join of “prepare bill” and “send book” is replaced by an AND-join. “Pay bill” will then become enabled only after both “prepare bill” and “send book” have fired. The resulting workflow net can be proven as being sound.

**Presenting the results**

The final results for the book-ordering workflow are presented in the screen shown in Fig. 8. The “Diagnosis” entry in the list summarizes the results. In this case it was shown that the workflow net is not sound. The properties mentioned in this section can be found under the “Properties” entry.

![Woflan's diagnosis results for the book ordering workflow](image)

**Fig. 8.** Woflan’s diagnosis results for the book ordering workflow
6 Conclusion

An intention of this paper was to show that Business Process Analysis is an important issue for managing the work conducted by companies. It is necessary to be able to analyze workflows with regard to errors or inappropriate design. Although there are a number of modeling approaches available there is no model that would address all features of a workflow to the same extent. Therefore it is important to choose the modeling technique according to the demands of an individual workflow. Because of the graphical representation and the abundance of analysis techniques, using Petri nets as modeling technique is very useful when describing control-flow oriented workflows.

Moreover, formal modeling methods (like Petri nets) have a number of advantages. Because the semantics of the model are pre-defined, there is less room for ambiguities and thus misunderstandings are prevented more effectively. It is also very helpful to have formal analysis techniques for applying tool support. As shown by Woflan, it is possible to have a workflow checked for some types of errors automatically.

Tool support in analyzing workflow properties is increasingly important for dealing with the complexity of processes conducted in modern companies. In order to facilitate the analysis of workflows even for inexperienced users it is furthermore important to provide these tools with an easy to use interface. When a workflow net is proven as being unsound the analyzer does not necessarily get a hint where the cause for this problem lies. It is therefore difficult to find and correct an error which results in an unsound net. A drawback of Woflan in this context is that it does not use any graphical representation of the diagnosed workflow nets but instead relies on the textual representation of the determined results. This also complicates finding the sources of detected errors in the workflow definitions in some cases.

The soundness criterion for verifying the correctness of workflow nets can be seen as a minimal requirement any workflow net should fulfill. Still, other aspects like the readability or maintainability of a Petri net are not addressed by that criterion. It is possible to create a sound net which is not structured well enough to be understandable or too abstract to actually be useful in practice.

Another issue currently being discussed is that soundness as a minimum requirement is actually too strict. When a workflow is modeled, more execution possibilities than the ones that will occur in practice might be contained in the Petri net. The so called Relaxed Soundness as a less rigid requirement on workflow nets examines only the paths through a workflow that were originally intended to happen. In the practical implementation of the workflow, it has to be assured that only the intended cases actually occur. See [4] for a detailed description of relaxed soundness.

7 References

17. Woflan homepage http://tmitwww.tue.nl/research/workflow/woflan/
Flexibility In Workflow Management Systems

Christian W. Günther
Chair of Business Process Technology
Hasso Plattner Institute of Software Systems Engineering, University of Potsdam, Germany
christian@hellskitchen.homeunix.org

Abstract. Especially in the Workflow domain it is crucial for successful business to have a mechanism allowing for change of business processes without halting the whole system. However, implementation of viable mechanisms for handling such change faces critical problems and open questions. This paper focuses on the particular problem of transferring running workflow instances to a newly changed process definition. Two approaches for tackling this are examined more closely for their suitability and compared with respect to practicability. Finally a universal concept for integration of multiple limited solutions is sketched.

1. Introduction

Today’s enterprises are typically faced with handling business processes on an extraordinary high stage of complexity everyday. Being composed of a large number of activities, which can be accomplished by a defined, set of employees each, finishing these business processes successfully without tool support is on the one hand usually complicated and hard to control. On the other hand frictionless handling is mission critical for the organization’s survival in hard-fought marketplaces. Orchestrating and controlling such large processes composed of hundreds of single activities is naturally not deemed a task being handled best by humans.

During the past years workflow management systems (WFMSs) have increasingly gained importance in real-life enterprises, bridging the gap in controlling complex workflows. What they are is software systems assisting an organization in all aspects of handling its business processes, while their main field of usage certainly lies in controlling the flow of documents between employees in large-scale office environments. Once installed a WFMS can aid in specifying, executing, monitoring and coordinating the flow of work items within the organization. These so-called workflows are specified in a general manner, called Workflow Process Definitions; they consist of a set of activities plus interconnecting links, specifying the control flow of the process. Modeling WF Definitions, besides the fact that merely processes having existed implicitly within the organization long since are now being modeled in an explicit manner, is actually one challenging and thus much error-prone task. Once actually executed, WF definitions are spawning cases, where every case has an individual execution sequence or path of activities contained in the original definition.
No matter how carefully modeled, for every WF definition the need for changes occurs once in a while. The reasons for such change are numerous and can stem from within the organization as well as being induced from outside parties [3]. Subject of change can be almost every aspect included in Workflow modeling, like modifications regarding the role set of the organization (e.g. group hierarchies, membership relations), yet in this paper the focus shall be limited to structural changes affecting the set of activities and the interconnecting control flow [2].

Structural changes can be of one out of four different kinds [2]. The first one, Extend, describes a change introducing new activities to the definition, with a Reduce type change activities are correspondingly removed from the flow. A hybrid variant of these two, i.e. removing some activities while adding others, is called Replace. Finally, a Re-Link type change describes that the set of activities within a WF Definition remains the same, yet their respective order or, more precisely, the set of control links interconnecting them, is altered, yielding one in a way altered control flow between the single tasks and thus different semantics.

Another criterion for classifying change is the effect such change has on the system as a whole [2]. If the change is merely affecting one single case (or, more generally, a distinct selection of cases), the procedure belongs to the class of Ad-hoc changes. These are generally applied in response to individual errors or exceptions detected at run-time, or they follow a request from outside parties like a customer that wishes one special activity to be skipped, because he is in an especial shortage of time. Handling this kind of changes usually poses no great challenge to the WFMS, as the designers implementing them can take care for guaranteeing, that the altered cases can be finished correctly. In contrast to these individual patches, change procedures that take effect on every case of a WF Definition, belong to the class of Evolutionary Changes. Reasons for such comprehensive change are either reengineering efforts, i.e. actions necessary to take for correcting a faulty definition or for improving overall flow performance. Or these procedures are responding to corresponding changes in the organization’s environment, e.g. changes in common practice in one particular field of business, or in legislation.

It is clear that cases already finished at the time the change occurs are naturally not affected. Correspondingly, cases started after the change procedure can be assumed to execute without problems, given the fact that the changed WF Definition is correct and has no errors. Thus, the actual problems introduced by change procedures are mainly affecting cases that are in execution during the time these changes are implemented. The simple solution of halting the entire WFMS for this task is generally not acceptable in practice [5]. Basically there exist four possible solutions, how to deal with these cases in execution:

1. **Forward Recovery**: All cases in question are aborted, and must then be finished manually.
2. **Backward Recovery**: Cases in question must be rolled back to their initial position, then they are re-started according to the new definition.
3. **Proceed**: Cases in question are finished according to the old definition, newly started cases follow the new definition.
4. **Transfer**: Cases in question are paused, transferred to the new definition and then resumed.
Flexibility In Workflow Management Systems

The problem with forward recovery is obvious: when one imagines a large number of cases to be finished by hand, it is clear that such procedure is, be it additionally due to a certain point in time, especially demanding regarding work force – this might overstrain the workforce of most organizations and is thus impracticable. Backward recovery is also not the solution of choice for many domains; each activity already accomplished has to be made undone, which is highly problematic in a semantic sense and often plain impossible (consider industrial production, scrapping a half-finished manufacture is often cheaper than dissection and re-integration). The Proceed solution is found implemented in some WFMS on the market, it is basically easy to put into practice, yet it also carries some major drawbacks. Consider the above example of applying evolutionary change to a WF Definition in response to a change in legislation. Regarding a bank handling a credit grant, such processes have a typical run time of years or even decades. Now if the law changes affects this process and it has yet another ten years to run in one case, a Proceed solution would not only be highly uncomfortable (as bank employees have to keep track of multiple versions of the process having accumulated over time) – just allowing the process to finish according to the old definition is also simply illegal.

In this example, the only true solution is a transfer of all running cases to the new definition changed in accordance to the new law, and most changes in practice do in fact request to be handled that way. However such transfer is a non-trivial task, it is difficult to implement and bears serious risks of introducing an error that renders transferred cases unusable. The result of these difficulties is that necessary changes are postponed and the WFMS is increasingly evaded, leading to a situation where it becomes more of a pain than a helping hand. In practice, no comprehensive and fail-safe solution able to handle all emerging problems optimally has been found to date, this being topic of intensive research among scientists. Finding a vital solution is nonetheless crucial for the future success of WFMS’s as a whole and their expansion into dynamic and thus change-prone areas of business in particular.

The remainder of this paper is organized as follows: After introducing some essential preliminaries in the second section two previously published approaches for dealing with dynamic change are presented. In the fourth section these are subsequently compared and an attempt to integrate them is made, followed by a Conclusion.

2. Preliminaries

Although describing practical processes, WF Definitions in their actual incarnation, i.e. complicatedly formatted text files, tend to be rather non-intuitive to comprehend. Therefore, for examining approaches already taken to tackle dynamic change in WF Definitions and reason about possible solutions it is necessary to have a common language about the topic, ensuring that it is not syntactic difficulties hindering the access to the actual semantics. Abstract graphical modeling has long since proven to provide this intuitive yet precise means; for this sake one method of modeling WF Definitions in a graphical manner, called \textit{WF-Nets}, is introduced in the following. In
the second subsection one property defined on WF-Nets, namely *Soundness*, is introduced, aiding in evaluation of change correctness.

2.1 WF-Nets

![WF-Net example](image)

One example of WF-Nets is shown in Fig. 1. It is obvious that WF-Nets are a subclass of Petri Nets; transitions represent the workflow’s activities, places are corresponding to states the flow can be in and arcs interconnecting transitions and places in a bipartite manner represent the control flow of the WF Definition. The actual marking of the net’s places can be used to visualize the current execution state of a specific case. What makes WF-Nets special in contrast to usual Petri nets is that they have each one dedicated input and output place, such that it can be strictly defined when a flow is in its initial state and when it is completely finished.

2.2 Soundness

One property that can be defined upon WF Definitions with the assistance of WF-Nets is *Soundness*, actually being a set of properties one would intuitively regard as minimal requirements toward a working WF Definition. To be *sound*, a WF Definition has to comply with the following requirements:

1. *Safeness* – What safeness basically means is the guarantee that in every state a case of a WF Definition can be in, there is always a maximum of one token in each place of the WF-Net.
2. *Proper completion* – When the single output place of a WF-Net is marked it must be guaranteed that no further place within the net contains a token.
3. *Absence of deadlock* – From any reachable marking of the WF-Net, it must be always possible to reach the exclusive marking of the output place by firing an arbitrary number of transitions.
4. *Absence of dead tasks* – For each transition in the WF-Net, there is at least one valid firing sequence starting from the initial marking enabling that transition. This basically means that every activity within the WF Definition has the chance to be actually executed.
If a WF Definition is sound, most problems that can emerge during execution of this definition are avoided. There are also a number of further desirable properties resulting from soundness aiding in theoretical reasoning about WF Design and change, described in detail alongside thorough theoretical foundations for WF-Nets in [1].

3. Inheritance-based approach

Changing a WF Definition can always be interpreted as exchanging an old definition by a new one. The first approach examined, namely the inheritance based approach presented in [1], takes advantage of this fact by defining four inheritance relations between WF Definitions. These relations are subsequently used to define inheritance-preserving transformation rules, equipped with adjacent transfer rules, such that when sticking to inheritance-preserving change it is always possible to safely transfer running cases. In the remainder of this chapter, the concept of inheritance, transformation rules and transfer rules are subsequently introduced, followed by a short discussion pointing out advantages and limitations of this approach.

3.1 Inheritance between WF Definitions

The notion of inheritance is widely familiar from the Object Orientation paradigm, which is increasingly gaining popularity throughout the industry these days. Inheritance can be described generally as a relationship between two classes, in this case classes of WF Definitions, where one class takes the role of a superclass and the other one is called subclass. The subclass incorporates all of its superclass’s features and extends it with additional ones; in this domain the subclass has a superset of the superclass’s activities. Thus, the subclass can be used to emulate the observable behavior of the superclass, as it is generally more powerful regarding its language of possible execution sequences. This emulation of behavior can be achieved by applying two basic mechanisms restricting the behavior of the subclass when executed.

The first method is called Encapsulation, yet it is not to be confused with the correspondingly named principle in Object Orientation. What encapsulation means in this context is removing activities from the subclass WF Definition, or correspondingly blocking them such that their execution is prevented.
This principle can now be used to define the first class of inheritance:

**Protocol Inheritance**: If by Encapsulation of all newly added activities within the new WF Definition we can achieve the identical observable behavior as in the old WF Definition, then the new definition is a subclass of the old one under Protocol inheritance. The example definition \( \mathcal{P} \) depicted in Fig. 2 is a simple sequence of three activities A, B and C. Now this definition is changed such that a new branch featuring activity D can be executed alternatively to activity B, as shown in Fig. 3. If now in \( \mathcal{Q} \) activity D is being blocked, i.e. encapsulated, all possible execution sequences (in this example, this is only A-B-C) of the new definition are also possible to be executed using the old definition \( \mathcal{P} \), i.e. encapsulation results in the same observable behavior. Thus, \( \mathcal{Q} \) is a subclass of \( \mathcal{P} \) under Protocol Inheritance.
The second method is called Abstraction and describes renaming newly added activities to . Activities labeled with are silent transitions, i.e. their execution is hidden from the external observer. Now the second class of inheritance can be defined:

**Projection Inheritance:** If by Abstraction of all newly added activities in the new WF definition we obtain the same observable behavior as in the old definition, the new definition is a subclass of the old one under Projection Inheritance. This is illustrated by WF Definition $R$ shown in Fig. 4, here a new branch containing activity D has been added that can be executed in parallel to activity B. If the actual execution of activity D is hidden from the observer, i.e. abstracted from, the identical observable behavior like in definition $P$ is obtained. Such, $R$ is a subclass of $P$ under Projection Inheritance.

The remaining two classes of inheritance are combinations of Protocol and Projection Inheritance and can be introduced without further preliminaries:

**Protocol/Projection Inheritance:** If all newly added activities can be both encapsulated and abstracted from and the new definition shows the same observable behavior as the old one in either case, then the new definition is a subclass under Protocol/Projection inheritance. An example for Protocol/Projection Inheritance is WF Definition $S$, shown in Fig. 5; the newly added Activity D is introduced within a looping branch and, after having been executed, marks the place it has initially taken its token from. Both hiding every execution of D and blocking its execution leads to $S$ showing the same observable behavior like $P$.

**Lifecycle Inheritance:** If, by encapsulating some of the newly added activities in the new definition and abstracting from the remaining ones, the identical behavior as with the old definition is observed, then the new definition is a subclass of the old one under Lifecycle Inheritance. In definition $T$, depicted in Fig. 6, activity D has been added to $P$ within a new alternative branch to B; another activity E was inserted sequentially between B and C. It is obvious that when blocking D and hiding the execution of E the definition $T$ shows the same observable behavior as the initial definition $P$. It is consequently a subclass under Lifecycle inheritance.

As for relations between the four different classes of inheritance, Protocol Inheritance and Projection Inheritance are orthogonal (i.e. they do not imply or except each other), thus enabling for the definition of the two combined Inheritance classes. Protocol/Projection Inheritance is the most restrictive of the four, implying both Protocol and Projection Inheritance. These two both imply Lifecycle Inheritance, the least restrictive class. The choice of which inheritance class to keep to when implementing change remains to the user, as the adjacent semantics cannot be covered within the principle of inheritance itself. It is deemed indicated to leave such decision to the management or the workflow architect as each inheritance class implies certain features, whose usefulness is dependent on the very nature of the WF Definition subject to change.

The correctness of these inheritance relations as well as the adjacent features can be proven as they are based on sound theoretical foundations. However, such proof is not within the scope of this paper, aiming to provide a general overview of the features provided; further this would require deep theoretical prerequisites, which are at this point out of focus. For such deeper understanding of the underlying principles
and thorough proof of both discussed approaches, the reader is referred to the respective publications.

3.2 Inheritance-preserving transformation rules

Knowledge about the existence of inheritance relations itself bears no significant use in practical application; this concept is rather to be used for validating change procedures against. As designing change and afterwards checking for inheritance compliance is obviously not considered a well structured and time efficient approach, resembling the low edge paradigm of trial and error, the approach depicted in [1] presents four inheritance-preserving transformation rules. Their sole application can guarantee for compliance of the new definitions to a dedicated inheritance class each and such provide a means for convenient yet consistent implementation of change.

For motivating the use of these transformation rules, a small example is to be introduced, illustrating their joint application to obtain one large-scale change.

The example illustrated in Fig. 7 shows a small workflow for ordering an item with a mail order enterprise. On the left is the old WF Definition, a simple sequence of first receiving an order plus adjacent payment (e.g. a paycheck), then handling the order, followed by shipping the ordered goods and finally archiving the order.

After some time the enterprise realizes that this model needs improvement: They want to allow billing after the order, reject an order in case the requested item is not available and send information to the customer in case shipping will be delayed. The envisioned new WF Definition is illustrated on the right of Fig. 7. In the following it will be described how the four transformation rules can be used to implement this change while preserving inheritance.

![Fig. 7. Order example, original and final Workflow Definition.](image-url)
3.2.1 PJ3S
Transformation rule PJ3S can be used to introduce new parallel branches of action to an existing WF Definition. It is a Projection Inheritance preserving transformation rule, i.e. the resulting definition is a subclass of the original definition under Projection Inheritance. When speaking of WF-Nets, what PJ3S basically allows is to insert a subnet $N$ between two transitions (i.e. activities) of the original net $N_0$ previously connected by a sequence, which results in a new WF-Net $N_1$. The requirement towards $N$ is, that if the in- and output places of this WF-Net are connected by a transition labeled $\square$ (a silent activity), the resulting net must be live and safe. Liveness means, for any marking of the net, it must be possible to enable any transition by firing an arbitrary number of transitions.

In short, this requirement states that after $N$’s input transition has fired, thus enabling $N$ to be executed, it must be ensured that (after a respective runtime) the output transition is enabled. In case this property holds true it can be guaranteed that, if $N_0$ was a sound WF-Net, $N_1$ will as well correspond to the Soundness property.

![Diagram](image)

**Fig. 8.** Order example after application of PJ3S.

The WF-Net depicted in Fig. 8 shows the original WF Definition after PJ3S has been used to introduce the new activity Billing within a branch parallel to task Shipping. It is obvious that this new WF Definition is a subclass of the original definition under Projection Inheritance as, if activity Shipping is abstracted from, the new definition will exhibit exactly the original definition’s observable behavior.

3.2.2 PPS
The Protocol/Projection preserving transformation rule PPS can be used to introduce a new looping sequence into an existing WF Definition. The special feature required from the newly inserted subnet $N$ is, that its input and output place (which it shares with the original definition) are identical. One further requirement is that, like in PJ3S, extension $N$ (an already short-circuited subnet) is also a live and safe WF-Net. This ensures that if $N$ is executed and thus taking a token out of its input place, this
token will afterwards be returned to that very place. Thereby it is guaranteed that $N_0$ is a sound WF-Net if the precondition holds that $N_i$ was previously sound.

![Diagram](image_url)

**Fig. 9.** Order example after application of PPS.

In Fig. 9 the activity *Send_Msg* (for sending the customer a notification, if shipping is delayed) has been introduced to the WF Definition resulting from the previous step as a looping subnet, which can optionally be executed before activity *Shipping*. This definition is a subclass to the previous definition under Protocol/Projection Inheritance as both hiding and blocking execution of *Send_Msg* yields identical observable behavior.

Notice that, with respect to the original WF Definition, this definition is a subclass under Projection inheritance, the composite application of different transformation rules always results in the inheritance class being the greatest common denominator of the inheritance classes preserved by all used transformation rules.

### 3.2.3 PJS

*PJS* is a *Projection Inheritance preserving* transformation rule that allows for inserting a new subnet sequentially. Informally this could be described as replacing an arc of the original definition $N_0$ by a whole subnet $N$, thus resulting in a new sequence. Once again, the requirement towards the extension subnet $N$ is that, if short-circuited, it must be live and safe, guaranteeing for preservation of the Soundness property.
In the above figure 10 the application of PJS is illustrated, which has here been used to insert the activity Check_Available (checking for availability of ordered items before processing the order) between activities Rcv_Order and Handle_Order.

It has to be stressed that this example uses most simple extensions; instead of the single activity Check_Available one could also insert a whole subnet, if this is live and safe. This requirement guarantees that after the inserted subnet has been enabled, it will, after considerable time, enable the following part of the original definition, thereby ensuring error free execution (regarding syntactic preliminaries of the control flow) of the new definition, which is one guarantee of the Soundness property.

3.2.4 PTS

The last one of the transformation rules presented in [1] is called PTS and is Protocol Inheritance preserving. PTS can be used to enhance an existing WF Definition by a new alternative branch inserted between two places of the original definition $N_0$ that must be previously connected by a control path. As seen in the preceding rules, the extension executed within the alternative branch can be of any scale, from one single activity to a complex subnet. Once again the requirement towards the extension subnet $N$ is, that it must be a live and safe WF-Net, if short-circuited; if this precondition holds potential soundness of the original definition $N_0$ is preserved to the new definition $N_1$. 
PTS can now be used to implement the final step in changing the example WF Definition, as shown in Fig. 11. Here the new activity Reject has been added within a new alternative branch starting in the input place of activity Handle_Order and ending in the flow’s output place. Encapsulation, i.e. blocking, of Reject restricts this definition to the previous WF Definition’s behavior, obviously PTS preserves Protocol Inheritance.

### 3.3 Transfer rules

The presented inheritance-preserving transformation rules provide a valuable tool for designing change in Workflow Definitions as their sole application guarantees that, under the precondition that the original definition was sound, the resulting new definition will also execute correctly. However this guarantee is intuitively only recognized with respect to WF-Nets having only their input place marked, i.e. WF Definitions in initial state, so far. The main problem identified in implementing change is the transfer of running cases; so the most valuable tool presented in [1] is surely the presentation of transfer rules, which can guarantee a safe transfer of each reachable marking (i.e. each possible case state) of the old WF Definition to the new one.

One remarkable feature of the approach in [1] is that not only transfer from super- to subclass is defined; additionally adjacent rules for transferring cases in the opposite direction are presented. This allows for inverse application of each transformation rule (deducing a superclass from a subclass) while having as well the means of transferring running cases.

---

**Fig. 11.** Order example after application of PTS.
Given the fact that PJ3S has been used to deduce a subclass from a WF Definition, [1] provides two rules that can be used to transfer running cases from super- to subclass. The marking of places inherited from the superclass can be identically transferred to the new definition, yet the question remains when it is necessary to mark the newly added places. To solve this, one has to take into account all places of the old WF Definition that are located along the control flow between the two transitions the new parallel subnet has been inserted into. Fig. 12 illustrates such situation, in this example places p and q would have to be examined. If all places in question do not contain any token, a marking of the newly added subgraph is not necessary. However, if any of these places contains a token, then additional marking of the newly added subnet is necessary for which [1] presents two distinct paradigms of transfer.

The Conservative (or Pessimistic) approach assumes that it is appropriate to execute the newly added branch if in doubt (as in this situation). Thus transfer rule \( r_{PJ3S,C} \) states that the input place of the newly added parallel subnet (place \( r \) in the above example) is to be marked. On the other hand, following the Progressive (or Optimistic) approach, the newly added subnet is not executed if in doubt, such its output place (in Fig. 12 this is place \( i \)) is to be marked. To ensure correct execution of the case after transfer it is crucial that one of either the new branch’s input or output place is marked, as otherwise the synchronizing transition (in the example this is \( C \)) cannot fire after the previously existing branch has been executed. The decision, whether to use the Progressive or the Conservative approach when transferring cases, cannot be made on the syntactical level. It depends strongly on the environment of the WF Definition and the definition itself, and should subsequently be decided by management or chief architects.

If transfer from sub- to superclass is to be implemented and the subclass has been deduced following rule PJ3S, the applicable transfer rule \( r_{PJ3S}^{-1} \) is defined as follows. Tokens in places inherited from the superclass are transferred identically; if places contained within the new parallel branch are marked, then these tokens are simply ignored on transfer. This is correct, as the state of the newly added branch has no equivalent in the superclass definition and is not needed for safe execution.
When a case is to be transferred from super- to subclass while the subclass has been deduced via transformation rule PPS, transfer rule \( r_{PPS} \) serves this purpose. As PPS merely adds new behavior to the WF Definition (in a looped subnet), it is absolutely sufficient to use transfer rule \( r_{ID} \); thus \( r_{PPS} \) is equivalent to \( r_{ID} \). The identity transfer rule \( r_{ID} \) states that only the marking of places inherited from the superclass is transferred, the newly added part remains unmarked in any situation. Transfer in the reverse direction is implemented using rule \( r_{PPS}^{-1} \); this rule is mapping the newly introduced looped subnet’s state to its joint input/output place. This means, if at least one place contained within the newly added part contains a token, then the input/output place is marked with a token in the target definition.

Fig. 13 shows an example of a Protocol/Projection inheritance relationship implemented by PPS; when transferring from left to right, it is obvious that an identical transfer is correct. The reverse direction would imply that places \( r, s, t \) and \( u \) be mapped to place \( p \) in the superclass definition on the left.

For a transfer from superclass to a subclass deduced using rule PJS, the transfer rule \( r_{PJS} \) is equivalent to the identical transfer rule \( r_{ID} \). As PJS merely introduces new behavior and does not affect the inherited part’s execution sequence, an identical
Flexibility In Workflow Management Systems

transfer taking only inherited places into account is correct. For a transfer in the reverse direction, rule $r_{PTS}^{-1}$ is used to map all tokens potentially present in states within the newly introduced sequence subnet to one token in the new subnet’s output place.

These two rules can be explained using Fig. 14; when transferring from super- to subclass, all states contained within the original definition are as well present in the target definition, thus an identical transfer is perfectly valid (which is what $r_{ID}$ is based on). When transferring in the reverse direction (aside from an identical transfer of places $i, p, q$ and $o$) the newly introduced places $r, s, t, u$ and $v$ are mapped to the added part’s output place $p$.

![Diagram](image)

**Fig. 15.** Transfer rules $r_{PTS}$, $r_{PTS}^{-1}$ and $r_{PTS}^{-1}$.  

Transfer rule $r_{PTS}$, used for transfers from super- to a subclass deduced using PTS, is once again identical to the identity transfer rule $r_{ID}$. When examining Fig. 15 depicting such situation, this can be grasped intuitively; as any state of the original definition is also present in the target definition; thus an identical transfer is valid.

For transfer in the opposite direction there exists once again both a Progressive and a Conservative solution: All tokens potentially present in states within the newly added alternative branch have to be mapped to one token in a place, that is also present in the target (i.e. superclass) definition. The Conservative approach states that this place be the alternative branch’s input place, $p$ in the above example; Using the Progressive approach, the output place (i.e., place $q$ in the example) has to be marked.

Notice that every transfer rule presented implies $r_{ID}$ (i.e., tokens in all states present in both the old and the new definition are mapped identically); yet for a more concise presentation this implication has not been explicitly stated for each rule.

### 3.4 Discussion

The inheritance-based approach is one very comprehensive regarding the tools and concepts provided. Not only does it present transformation rules for changing WF Definitions, it is also based on an intuitive model of WF Definition relations, namely the idea of inheritance, and presents adjacent transfer rules that actually provide the means for transferring running cases.
When sticking to inheritance-preserving transformation it is guaranteed that soundness of the original definition is preserved and a safe transfer, i.e. non-problematic continuation of case execution afterwards, is ensured. However, the approach has some limitations considering the range of possible changes. In practice, most desired changes in WF Definitions can be accomplished using the presented transformation rules, thus preserving inheritance with all implied positive aspects. Yet, it is clear that this approach is limited to changes of type Extend, Reduce and Replace, where it is obvious that in everyday use, sooner or later, the need for changing the control flow structure of a WF Definition, i.e. Re-Link change, will emerge.

![Diagram](image_url)

**Fig. 16.** Sequentialization of billing and shipping in the order example.

One can for example imagine that, regarding the initial order example, the parallel execution of billing and shipping is after some time not esteemed an optimal solution. Maybe the enterprise has had a considerable amount of customers that had already been sent the items ordered, yet for arbitrary reasons they have not paid the respective bills. To resolve this problem the management decides to perform shipping of items not before billing has been completed successfully.

The respective change in the WF Definition is depicted in Fig. 16; formerly parallel activities *Billing* and *Shipping* have now been sequentialized. This change procedure can however not be implemented using the transformation rules presented in [1], so for comprehensive and useful implementation of change as a whole, another mechanism capable of handling Re-Link type change is required.
4. Change regions

While the inheritance-based approach presented is very detailed and provides guarantees like preservation of soundness it has the serious drawback of limiting change to Extend and Reduce type procedures. The second approach, described in [4], is of considerably earlier date, yet it provides a means for handling running cases whatever the change may be; therefore the problem of transfer is concentrated on so-called Change Regions and evaded by introduction of a transfer method called Synthetic Cut-Over Change. In the following the problem with Re-Link type changes is described, followed by the concepts of Change Regions and Synthetic Cut-Over Change. The chapter finishes with a short discussion of the approach presented.

4.1 The dynamic change bug

Although Re-Link changes are sometimes desired, they can lead to significant problems when transferring running cases, for example one major defect that can occur is called the Dynamic Change Bug.

In Fig. 17 a situation similar to the one presented in 3.4 can be observed, two tasks C and D having been executed in parallel in the old definition \( P_0 \) have now been changed to run sequentially in the new definition \( P_1 \). If one now imagines a case of the old definition \( P_0 \), with places \( g \) and \( k \) marked to be transferred to \( P_1 \), the question arises of how to accomplish such transfer.

The presence of a token in place \( g \) suggests that the intuitively equivalent place \( q \) be marked in \( P_1 \), as both are output places of activity \( B \) and input places of activity \( C \). Yet the problem is, that this would later enable transition \( D \), which has however already been executed according to \( P_0 \). Another possibility that may come to mind is marking place \( s \), as in \( P_0 \) the corresponding place being input to activity \( E \) and output to activity \( D \), namely place \( k \), is also marked. Obviously this is also erroneous, as in this case activity \( C \), previously enabled in \( P_0 \), would never be executed. Marking both places in question is no solution as well, because then both activity \( D \) is executed
twice and the net is no longer sound (with the output place marked, the net would still contain another token).

4.2 Notion of change regions

It can be intuitively realized that the problem presented in 4.1 does not affect the whole of the old and new WF Definitions. For example, a sole token in place $f$ of $P_o$ can be mapped to $p$ in $P_i$ without complications. Principally the effects of change are limited to a rather small part of the WF Definition, and subsequently the area creating problems on transfer is also limited to this part (the remainder can be transferred using $r_{id}$, as every place is present in both definitions). So the first step the approach presented in [4] takes to tackle the problem, is to narrow it down to an area called the Change Region, which is defined as being the smallest possible part of the WF-Net affected by the change.

The approach does not clearly state how to determine this region, yet it must be noted that it is always possible to select a part of the definition containing all activities affected by the change; in the worst case this equals selecting the whole net. On the other hand the question of selecting the optimal change region is not only a syntactical one but also has to take into account semantic relations between activities not being explicitly modeled. Therefore it is advisable to, as a first solution, have the region defined by WF designers implementing the change.

As change describes the replacement of an old WF Definition by a respective new definition, there is for every old change region in the original WF Definition one respective new change region in the new definition; thus change can be perceived as replacing the old change region with the new one. One solution is to select the change region in such way that if in both old and new WF Definition the change region is replaced by one virtual transition, both definitions are identical. In Fig. 18 the selection and extraction of the old region $N_i$ from the old definition $P_o$ is depicted, further the respective new change region $N_2$ is shown.

When comparing the old change region to the new one, a property of the adjacent change can be defined: If the language, i.e. the set of possible states, of the new change region (and such, the new definition) is larger than the language of the old region, then the change has the upsizing property. If otherwise the language of the old
region is more powerful, then the *downsizing* property can be assigned to the change. Now if the change has the upsizing property (an example would be changing \( P1 \) to \( P0 \) in the example presented), then there is no problem. Every state a case of the old definition could have been in can be identified as well in the new definition, such that a safe transfer is always possible. However, if the change has the downsizing property, like in our dynamic change bug example, there is a problem that cannot be solved using the means already presented. For handling such downsizing changes, the approach presented in [4] provides a change principle called *Synthetic Cut-Over Change*.

Notice, that one can imagine a third class of changes having neither upsizing nor downsizing property, however both the old and the new region can have states (or markings) that cannot be transferred to the respective other region. In such case the use of the method presented in the next paragraph is naturally also needed.

4.3 Synthetic Cut-Over Change

Mere knowledge about the existence of the change region concept does not solve any of the related problems. Yet we know that the definition without the change region can be identically transferred, and such the solution must lie in how to deal with the change region. Simply replacing the old region by the new one yields the same problems as identified above, dynamic change bug and the like.

What the approach described in [4] proposes is, that the old and new change region be *fused* and subsequently the old region not be replaced by the new but by the resulting *fused* change region. Fusing the change regions is done as follows: The output places of the two regions are merged to one single place, and when this fused region is connected to the remainder of the WF-Net, only this merged output place and input places of the new region part are connected. Input places of the old region remain abandoned.

![Fig. 19. Fused change region and Synthetic Cut-Over Change.](image)

This principle of exchanging the old region by a new *fused change region* is called *Synthetic Cut-Over Change (SCOC)*. Fig. 19 shows the corresponding fused change region \( N_{\text{fused}} \) and the new definition \( P_{\text{SCOC}} \) resulting from the application of SCOC,
with respect to the example presented above. Obviously, newly started cases will, after activity A has been completed, enter the fused change region in place p, and subsequently follow the new change region part of the fused region. Cases being in execution during the switch from old to new (fused) definition, i.e. cases that have any places of the old region part marked, will continue using the old region part until they have finished the region.

4.4 Discussion

The most positive aspect about the change region based solution is definitely, that it can handle every possible change procedure and guarantee a safe transfer. Yet the disadvantages are obvious: what SCOC basically does is the same as what has been described as Proced solution in the introduction, only that it remains restricted on the change region part of the net. Therefore, the original problem is only evaded. Remember the example of the credit granting process in a bank, if this process has a runtime of 20 years but the selected change region still has a runtime of five years, not much is gained in tackling the problem, the new process will still yield illegal behavior (given the fact that the new law must be implemented before those five years and also that it affects the change region part of the definition).

Another disadvantage of SCOC is, that the resulting net does not correspond to the Soundness criteria, as it contains transitions (in the old change region part of the fused region) that cannot be enabled by firing a sequence from the initial marking, such that those transitions must be conceived dead. If one wants to return to a correct WF definition whenever possible, a versioning and garbage-collection like mechanism has to be implemented that replaces the fused change region by the new one, as soon as there are no longer any processes running in the old change region part.

Further, in the fused change region there exist multiple activities with the same label, which is on the one hand illegal (yet does not disturb regular execution in general), and on the other hand can produce serious problems regarding flow monitoring. How can one tell to which degree the flow has been finished, when the position of an activity cannot be positively located? However, monitoring problems exist in various other aspects regarding the area of dynamic workflow change and solving them is outside the scope of this paper.

5. Comparison and Integration

The inheritance-based approach has clear advantages regarding the overall smoothness of transfer. As for each applicable transformation rule there exists a corresponding transfer rule, every case can be instantly transferred to the new definition, as long as only the inheritance-preserving transformation rules have been used in its deduction. This transfer has some further desirable features like preservation of soundness and thus non-erroneous continuation, yet under the precondition that the original definition has been sound.
On the other hand, like mentioned before, the most grave drawback of this method is, that some desirable changes simply cannot be performed using the transformation rules provided. This is maybe the largest benefit of the approach presented in [4]; by selecting an appropriate change region virtually any change can be implemented while guaranteeing a safe transfer. The adjacent disadvantages are obvious; transfer is not seamless and after all the solution is nothing more than a simple Proceed of running cases, sophisticatedly limited to the change region part of the definition.

However both approaches can be integrated to provide a complementary, more comprehensive means for handling dynamic change. Most aspects of change difficulties can be solved using inheritance-preserving transformation rules, as far as possible. Regarding changes where the preservation of inheritance is not feasible, SCOC can be used to provide a fallback, simply covering the problems remaining unsolved by the inheritance-based approach. Such approach has been sketched by the running order example, most of the desired changes could be handled preserving inheritance. Only rather small a part of the WF-Net would have to be declared as change region, thus both evading most of the problems that might stem from a purely SCOC solution and being able to solve virtually all desired kind of changes. As the area of dynamic change in WF Definitions currently lacks a method comprehensive enough to provide smooth and instant transfer of any change to be imagined, it is proposed to use limited approaches like the inheritance-based one that can solve transfer for a particular class of changes optimally. The inheritance method provides a first tool within an imagined box of methods for tackling most problems related to dynamic change. However, should the need emerge for implementing a special kind of change that cannot be handled by the existent comprehensive tools, change regions can be defined and SCOC can be used to solve the remaining problems.

6. Discussion and conclusion

In this paper two approaches for tackling the problems associated with dynamic change have been presented. The first inheritance-based approach is very concise and can serve as more of a real solution to the problem, yet the class of changes it can provide safe and smooth transfer for is limited. The second approach based on the notion of change regions is far less advanced and strives more for containment of the essential problems within an area called change region, while actual transfer resembles more the naïve Proceed solution. However, it is a comprehensive tool for providing safe transfer, as virtually every desired change in WF definitions can be solved by SCOC, providing safe and non-erroneous transfer for. Finally a combined, extensible solution has been sketched that uses for each part of the problem the most appropriate solution and leaves non-resolvable aspects to be handled by SCOC.

There have been several further approaches addressing the problem of dynamic change. The idea of [6] is at first glance similar to [1], structural relationships between WF Definitions are identified and classified, including deviations in control flow allowing for Re-Link type change. Yet this approach lacks a transfer solution adjacent to the presented transformations; further it allows for changes leading to situations like the Dynamic Change Bug and it is questionable if such system of
relationships can actually be made safely transferable. In [5] an adaptive approach is sketched requiring manual partitioning of a workflow for handling dynamic change, also one serious limitation regarding the implementation of tool or system support.

The combined approach presented in this paper provides the means necessary for implementing one pragmatic mechanism capable of tackling all problems that may emerge in practice. If constantly extended and refined following ongoing results in research, it can be built towards a comprehensive mechanism, steadily perfected. Its major drawback is that the overall result does generally neither comply with an inheritance relation nor is soundness preservation guaranteed. This results directly from SCOC’s limitations with respect to inheritance and soundness compliance; the outcome is a more complicated process of correctness checking and further meta-information would have to be generated and kept for later check-up and comprehension.

One great challenge lies in integrating further limited approaches into the proposed framework in practice, this has to be accomplished with care to preserve overall correctness. A well-designed mechanism, assigning each aspect or part of the change procedure the optimal transformation model and thus transfer mechanism, can ensure that the transferred case is handled as closely as possible regarding its compliance to the new definition’s semantics. Automating the process of case transfer reduces the need for manual adjustment and can thus significantly decrease adjacent risks - improving the value of Workflow Management Systems and paving the way for their future success in areas prone to change.

References

Abstract. Today, great potential is attributed to the Service Oriented Architecture, to alter and simplify the interoperability of services by arbitrary service providers. The current implementation of the Service Oriented Architecture are Web Services. As an Internet technology, this is likely to generate a very large, distributed and heterogeneous environment of services. The need for explicitly modeling business processes in such an environment is apparent and yet, various competing approaches exist. In this paper, a process execution engine based on Workflow Patterns is presented and discussed, providing a generic platform able to cope with the requirements of the Service Oriented Architecture and flexible enough to map arbitrary workflow languages.

Introduction

Today, the Internet is becoming a commodity and as such, influencing the way companies do business. Concurrently, development of software from scratch is more and more replaced by assembling from existing software components. Put together, the idea of integration of data, knowledge and processes among business partners over the Internet becomes apparent and is a current topic in the industry and in research. The new paradigm is the Service Oriented Architecture (SOA) [1][2]. Combining services, especially inter-organizational, to applications and processes is key to leverage the benefits of the SOA, thus workflow engines will play an important role. In the course of this paper, implementation concepts for such an engine based on Workflow Patterns [7] are explored.

In computer science, building reusable software components has been a long-standing goal. The first approach started with the idea of functional decomposition, leading to application programming interfaces as syntactical description of operations. Object orientation was the next concept, encapsulating data and functionality in logical units of objects as instances of classes. Yet, objects still only have a syntactical description. SOA, as a new paradigm, is focusing less on the software components and their implementation, but the services they provided. These services are bound to a semantic contract, which not only defines a syntactical interface, as objects do, but also describes its behavior, properties and location. In the future this contracting will include non-functional properties [3]. The most prominent implementation of the SOA are Web Services, which focus on universal interoperability between applica-
tions by using Web standards. Web Services use a loosely coupled integration model to allow flexible integration of heterogeneous systems.

The SOA paradigm has evolved out of the need to quickly adapt to customers and market conditions. Organizations offer services, implemented by a combination of various software components. Combinations of services produce applications and new services. Semantic contracts allow dynamic rebinding/remodeling during runtime for instant adaptation to both external and internal factors. Choosing these combinations and giving meaning to them is the domain of business processes. Modeling and executing business processes within an organization is a well-known topic. Yet, today most processes are not modeled at all, i.e. the user implicitly executes a process by using standalone software, or are implicitly hard-coded into the software. Even if workflow management systems are used, little support for the dynamic aspects of the SOA is provided, even less for inter-organizational workflows.

This situation is being addressed within the scope of Web Services by extending the notion of universal interoperability to business process modeling. One prominent effort, backed by large-scale vendors, is the Business Process Execution Language for Web Services (BPEL4WS) [6]. BPEL4WS is an open specification issued by the OASIS Group, based on XML and can be used both to describe abstract process description, similar to a business protocol, as well executable scripts and is has well-defined extension points.

Still, BPEL4WS is work in progress and has competitors. Thus, when designing a workflow engine for the SOA, BPEL4WS can only be seen as an orientation point. Even worse, there is no universal theory to workflow modeling. W.M.P van der Aalst, et al. faced a similar problem trying to evaluate suitability and expressive power of various contemporary commercial workflow management systems. This has let them to the concept of Workflow patterns [7]. A pattern “is the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts.” [4] Instead of exploring what a workflow management system can express and how to evaluate these constructs, workflow patterns take the reverse approach, trying to identify business requirements and deducing patterns from them.

Workflow patterns concentrate on control flow aspects, but help solving the complexity problem when evaluating workflow management systems. This paper takes the concepts of workflow patterns and applies them to design a workflow engine for the SOA. Our design goal is an embeddable, extensible and scalable engine that can easily map arbitrary workflow languages and deal with the dynamics of the SOA. Workflow patterns seem to be a suitable approach to reach this design goal. Moreover, by providing a library of workflow patterns, implementing a workflow language in our engine should become little more than mapping the constructs of the language to the workflow patterns in the library.

In the second part of this paper, the requirements for such an engine are shown. Then, Workflow Patterns and their applicability are discussed, leading to a proposed engine design. In the third part, implementation concepts for control flows and the Workflow Patterns expressible by them are presented. Afterwards, an introduction is given on how to map an arbitrary workflow language to the proposed engine. Finally, in the outlook topics of future work are outlined.
Designing an engine for process execution in the SOA

As seen, the SOA paradigm introduces new requirements and challenges for a process execution engine. This chapter deals with these requirements, how Workflow Patterns can be applied, and finally the proposed engine design is shown.

Requirements of the Service Oriented Architecture

As already discussed, the SOA focuses not on specific implementations, but the services they provide. Especially when looking at Web Services, this will lead to highly distributed and heterogeneous environments, where the involved information systems have little in common. Also, since dealing with the Internet, the potential number of involved systems is extremely high. Consequently, platform independence and scalability are very important requirements.

Another aspect already mentioned is the dynamics of the Service Oriented Architecture. Standards are still evolving and these emerging standards clearly define extension points. Thus, the foundation building upon is in constant adaptation. This adaptation even extends to runtime, as dynamic changes to the availability and non-functional properties of services lead to very late binding, possibly rebinding of services.

As Web Services are the most prominent implementation of the Service Oriented Architecture, consideration of their requirements is well advised. First, XML is the basis for data objects and service calls. Even the process languages are specified in XML. This is not very likely to be different in other SOA implementations, as XML has proven to be the Silver Bullet for structuring, describing and expressing. Even if it does change, mapping to XML should be easily done. More importantly, Web Services are stateless. At first a disadvantage, integration of arbitrary legacy systems is at least simplified, sometimes even the only reason why possible. Still, the burden of correlating individual, stateless service calls to their containing, stateful processes is shifted to the execution engine.

Using Workflow Patterns as a design approach

Workflow Patterns are a viable approach to reducing complexity when evaluating workflow languages and systems. The underlying idea of this paper is to use these patterns as a design principal where possible, as a guide when not, and at the very least as a verification mechanism.

Van der Aalst’s thesis, that mapping arbitrary workflow languages to abstract patterns is helpful for evaluation is extended or in some way reversed to the thesis, that a set of implemented patterns will simplify the implementation of arbitrary workflow languages as it is less an implementation, but a mapping process.

Positive side effect, this approach will lead to an easier to understand engine design, as the control flow mechanisms are explicitly modeled and their behavior is well-documented.
The notion of Scope

When looking at Workflow Patterns, their conceptual design and therefore their implementation seems straightforward. Yet, some issues remain. Most prominently, activities shown in Workflow Patterns are either completely stateless or at best there is one global state. This simplifies looking at the patterns, but when moving to an actual implementation, this restriction has to be removed, as it is very limiting.

Generally speaking an activity has a context, including but not limited to variables, events, exceptions, processes, transactions and obviously control flows. Workflow patterns abstract from this context to a point, where this is causing problems. For example in the Synchronizing Merge van der Aalst notes

“The main difficulty with this pattern is to decide when to synchronize and when to merge. Generally speaking, this type of merge needs to have some capacity to be able to determine whether it may (still) expect activation from some of its branches.”

Thinking in contexts, most workflow patterns can be characterized to affect either the beginning of a context (e.g. Parallel Split) or the end of a context (e.g. Synchronized Merge). As shown, there needs to be communication between a pattern starting a context and the pattern ending this context. Also, workflow patterns can call other workflow patterns, i.e. contexts can be nested.

Consequently, this will bring us to the notion of scope, grouping a set of activities into a context. This scope will have just one control flow, combining a starting and an ending pattern. A scope itself can be seen as an activity (i.e. sending a start signal to its control flow), thus by treating a scope as an activity, the required nesting of contexts is achieved.
Core Engine Design

The design goal is an embeddable engine. For this purpose, the engine treats the outside world as an interface, it can bind and over services with:

![Structural Overview](image)

The engine itself consists of a front-end class, and three different functionalities underneath:

- **Process Schema**: The process schema is a static representation of the process shared by all its instances. It does not contain any instance data. The important classes are ProcessSchema, acting as root, ProcessScope providing context, and ProcessActivity as an atomic task. Also a ProcessScope has a number of handlers, this paper will focus on the ControlFlowHandler only. ProcessScope inherits from ProcessActivity, as scopes themselves can be seen as an activity (i.e. pass control its ControlFlowHandler), providing a simple way of nesting scopes. ProcessSchema inherits from ProcessScope, as a schema cannot exist without a root scope and in fact is little more than such.

- **Process Instance**: A process instance represents the runtime environment of a process schema. ProcessInstance acts as a central point of access to all contained ProcessThreads. Each ProcessThread has an individual scope call stack, represented by the anonymous class `now()`. In this stack all instance information is stored.

- **Schema Generation**: The engine has a pluggable schema generation concept, represented by the SchemaGenerator interface. One possible implementation is introduced in chapter 0.
The following class diagram shows the relationship between the mentioned classes:

![UML class diagram of the core engine](image)

Since focusing on implementing Workflow Patterns, the behavior and interaction of ControlFlowHandler is shown in more detail. The exact signature is:

```java
ProcessActivity ControlFlowHandler.nextActivity
(ProcessThread currentThread,
ProcessActivity lastActivity)
```

Three different behaviors are defined:

1. `nextActivity = ControlFlowHandler.nextActivity(currentThread, null)`
2. `nextActivity = ControlFlowHandler.nextActivity(currentThread, lastActivity)`

3. `null == ControlFlowHandler.nextActivity(currentThread, lastActivity)`

The first behavior, passing in null as `lastActivity` is considered a start signal, i.e. the `ControlFlowHandler` should execute its starting pattern behavior. Then, the `ControlFlowHandler` is called again (2.) to decide which activity should be executed next. The passed in `lastActivity` is the previously returned activity, after its execution. Eventually all activities are executed and the `ControlFlowHandler` will execute its ending pattern behavior. This is signaled by returning null (3.), which in effect will end the scope and return execution to the parent scope's `ControlFlowHandler`, by calling `.nextActivity (currentThread, lastScope)`. 
Evaluating pattern implementation within the engine design

In this chapter, we will look at the various workflow patterns and how to implement them with the given engine design. For this purpose, the existing Workflow Patterns are grouped as suggested by the engine design. For each group a ControlFlowHandler is presented and how to express the group’s Workflow Patterns.

As seen, the ControlFlowHandler is called before and after each operation with the current ProcessThread and the preceding ProcessActivity as parameter. Passing a null activity signals the beginning of a scope to the control flow, returning a null activity signals the end of a scope to the engine.

During the call, the ControlFlowHandler has full access to the ProcessThread, can access and modify its call stack and spawn new threads by the newThread(nextActivity) method. Note that threads will automatically terminate, when they leave their scope. This means that a ControlFlowHandler cannot introduce parallel execution to parent scopes, only to nested scopes.

The challenge, when mapping workflow patterns to the engine design, is to determine how to map workflow patterns to a scoped environment. The present approach is based on the following principals:

− A ControlFlowHandler should always combine a starting and an ending workflow pattern. This is necessary to map workflow patterns to a scoped environment.
− If a ControlFlowHandler has a synchronizing ending workflow pattern, the parent ControlFlowHandler determines the activity to execute after synchronization, i.e. on synchronization the ControlFlowHandler should return null. This will enforce a clean separation between activities before and after the synchronization instead of putting them in one scope and moving the information about which activities belongs where into the ControlFlowHandler.
− Even if a ControlFlowHandler spawns ProcessThreads, it should always use the instance that activated it to return the execution to the parent scope. Otherwise the execution will die, as child threads cannot escape their scope.
− A ControlFlowHandler keeps its state in the ProcessThread instance only, thereby making instant reuse possible (compare figure 2). To do so, ProcessThread offers special support for internal variables (not shown in figure 2).
− An instance of an activity can be a member of several scopes, but never multiple times in one scope. Instead, multiple instances have to be created. This is mandatory to allow ControlFlowHandlers to coordinate themselves with instance references.

Based on this, in the following ControlFlowHandlers will be shown and the Workflow Patterns expressible by them. For an in-depth discussion of the shown patterns, please refer to the Workflow Pattern homepage [7].
SynchronizingControlFlowHandler

The SynchronizingControlFlowHandler can be used for all patterns synchronizing on a single thread of control. Its exact behavior is defined by its parameters:

- int numberOfActivitiesToSynchronizeAfter
  This parameter indicates how many activities should be executed before the thread of control is returned to the parent scope. This parameter is only honored, if and only if $0 < \text{numberOfActivitiesToSynchronizeAfter} < \text{number of activities in scope}$, otherwise all activities need to complete before synchronization.

- boolean executeInListOrder
  This parameter decides, whether the activities are executed in strict order or on availability determined by their join conditions. The SynchronizingControlFlow guarantees that an activity is executed at most once.

- boolean executeInParallel
  This parameter controls, if the execution of activities in this scope are done in parallel. Note: Contained scopes have their own ControlFlowHandler.

With combinations of these parameters, the following Workflow Patterns can be expressed:

Sequence An activity in a workflow process is enabled after the completion of another activity in the same process.
To obtain this pattern, the parameters

\[
\begin{align*}
\text{numberOfActivitiesToSynchronizeAfter} &= 0, \\
\text{executeInListOrder} &= \text{true}, \\
\text{executeInParallel} &= \text{false}
\end{align*}
\]

must be set. This produces a sequential control flow.

Exclusive Choice is point in the workflow process where, based on a decision or workflow control data, one of several branches is chosen.
This pattern is expressed by the parameters

\[
\begin{align*}
\text{numberOfActivitiesToSynchronizeAfter} &= 1, \\
\text{executeInListOrder} &= \text{false}, \\
\text{executeInParallel} &= \text{false}
\end{align*}
\]

This will execute the first activity, which join condition evaluates to true and then return.
**Deferred Choice** is point in the workflow process where one of several branches is chosen. In contrast to the XOR-split, the choice is not made explicitly but several alternatives are offered to the environment. However, in contrast to the AND-split, only one of the alternatives is executed.

From the implementation point of view, this pattern is exactly the same as the Exclusive Choice, as the number of activities is not limited to two and the selection is done just before executing the chosen activity.

**Interleaved Parallel Routing** is point in the workflow where a set of activities is executed in an arbitrary order: Each activity in the set is executed, the order is decided at run-time, and no two activities are executed at the same moment.

To achieve this pattern, the parameters

\[
\begin{align*}
\text{numberOfActivitiesToSynchronizeAfter} &= 0, \\
\text{executeInListOrder} &= \text{false}, \\
\text{executeInParallel} &= \text{false}
\end{align*}
\]

must be set. This will execute sequentially all activities in the scope, but the order is determined by evaluating the activities’ join conditions.

**Parallel Split** is point in the workflow process where a single thread of control splits into multiple threads of control, which can be executed in parallel, thus allowing activities to be executed simultaneously or in any order.

As we have explained before, we always need to group a starting and an ending pattern into one control flow. This is a starting pattern, which is chosen by setting \(\text{executeInParallel=true}\).

**Multi-choice** is point in the workflow process where, based on a decision or workflow control data, a number of branches are chosen.

Again, this is a starting pattern and in our environment synonymous to the Parallel Split, since activities have join conditions.

**Synchronization** is point in the workflow process where multiple parallel subprocesses/activities converge into one single thread of control, thus synchronizing multiple threads.

This pattern is exposed, when the parameters

\[
\begin{align*}
\text{numberOfActivitiesToSynchronizeAfter} &= 0, \\
\text{executeInListOrder} &= \text{true}, \\
\text{executeInParallel} &= \text{false}
\end{align*}
\]

are set. As the name of \texttt{SynchronizingControlFlowHandler} implies, some form of synchronization is always chosen.
**Discriminator** is a point in a workflow process that waits for one of the incoming branches to complete before activating the subsequent activity. From that moment on, it waits for all remaining branches to complete and "ignores" them. Once all incoming branches have been triggered, it resets itself so that it can be triggered again.

To obtain this pattern, the parameters

\[
\text{(numberOfActivitiesToSynchronizeAfter=1,} \\
\quad \text{executeInListOrder=true,} \\
\quad \text{executeInParallel=true)}
\]

must be set. All activities are started and the synchronization will take place as soon as the first activity terminates.

**N-out-of-M Join** is a point in a workflow process where \( M \) parallel paths converge into one. The subsequent activity should be activated once \( N \) paths have completed. Completion of all remaining paths should be ignored. Similarly to the discriminator, once all incoming branches have "fired", the join resets itself so that it can fire again. This pattern is a discriminator with `numberOfActivitiesToSynchronizeAfter` set to \( N \).

**Synchronizing Join** is a point in the workflow where multiple paths converge into one single thread. If more than one path is taken, synchronization of the active threads needs to take place. If only one path is taken, the alternative branches should reconverge without synchronization.

This pattern is synonymous to the Synchronization, as the number of threads is known and execution is continued when the last thread terminates.

**LoopingControlFlowHandler**

The `LoopingControlFlowHandler` is used for structural loops. As an advanced feature, execution of loops in parallel is supported. Its exact behavior is defined by its parameters:

- **String loopCondition**
  This parameter is the condition to evaluate, whether the loop should be executed (again).

- **boolean evaluateBeforeExecution**
  This parameter decides, when the evaluation relative to the execution of the loop takes place. Thus, we can alter the behavior between a `do` loop and a `while` loop.

- **boolean executeInParallel**
  This parameter controls, if the execution of each loop-iteration is done in parallel.

- **boolean synchronizeExecution**
  Since we have the option of parallel execution, this parameter determines whether all loop iterations need to complete, before the thread of control is returned to the parent scope. The combination `executeInParallel=false` and `synchro-
nizeExecution=false will spawn a single thread executing the loop and return immediately.

This ControlFlowHandler is able to implement all structural loop known in block-oriented languages. Also, the following Workflow Patterns can be expressed:

**Multiple Instances With a Priori Design Time Knowledge** - For one process instance an activity is enabled multiple times. The number of instances of a given activity for a given process instance is known at design time. Once all instances are completed some other activity needs to be started. This is a loop in parallel over a statically initialized counter.

**Multiple Instances With a Priori Runtime Knowledge** - For one case an activity is enabled multiple times. The number of instances of a given activity for a given case varies and may depend on characteristics of the case of availability of resources, but is known at some stage during runtime, before the instances of that activity have to be created. Once all instances are completed some other activity needs to be started. This is a loop in parallel over a runtime-initialized counter.

**Multiple Instances Without a Priori Runtime Knowledge**. For one case an activity is enabled multiple times. The number of instances of a given activity for a given case is not known during design time, nor is it known at any stage during runtime, before the instances of that activity have to be created. Once all instances are completed some other activity needs to be started. This is a loop in parallel over a termination expression evaluated at runtime.

**Multiple Instances With Synchronization** - For one case an activity is enabled multiple times. The number of instances may not be known at design time. After completing all instances of that activity another activity has to be started. This is a loop in parallel with synchronizeExecution=true.

**Workflow Patterns implemented by reference reuse**

Some Workflow Patterns can be expressed by using the reference to an instance of a ProcessScope or ProcessActivity more than once in a process schema. This is called reference reuse.

**Simple Merge** is a point in the workflow process where two or more alternative branches come together without synchronization. This pattern can be implemented by reference reuse of the activity following the merge. If we want to guarantee that no thread of execution can execute a scope in parallel, we could implement a ControlFlowHandler synchronizing on its scope instance.
Multi-merge is a point in a workflow process where two or more branches reconverge without synchronization. If more than one branch gets activated, possibly concurrently, the activity following the merge is started for every activation of every incoming branch.

This pattern is nothing more than a reference reuse of activities. The only restriction is, that an instance of an activity should never be contained more than once in a single scope. Yet, this is easy to workaround by inserting a proxy activity calling the reference.

Arbitrary Cycles is a point in a workflow process where one or more activities can be done repeatedly.

When reading about Arbitrary Cycles, they appear to be straightforward. Yet, Workflow Patterns neglect the context. Thus, when using reference reuse to build an Arbitrary Cycle, one should never forget, that this would create growing scope stacks. Thus, in most cases, structural loops as discussed earlier should be used, even if recursion is supported by the engine design.

For the rare cases, that there is a context free arbitrary cycle, i.e. a redo from start, this can be implemented as a ControlFlowHandler popping the scope back to the referenced loop point in the stack. But, this may have unforeseen side-conditions, requiring further analysis before implementing such.

Mapping workflow languages to the engine

So far, the engine design and how to implement workflow patterns in it have been shown. Now, a brief look at how to map workflow languages to the engine is taken. To do so, the engine should have a pluggable schema generator (see figure 2). As proof-of-concept implementation BPEL4WS was chosen. BPEL4WS is an XML-based language. Thus an XML-based schema generator is needed. Instead of writing one from scratch, readily available Open Source software is used. After some evaluation, Jelly [5] was chosen.

Jelly is a Java and XML based scripting and processing engine. Jelly is completely extendable via custom tags in a similar way to JSP custom tags, yet it has no dependencies on Servlets or JSP. For each XML-namespace a library can be plugged into Jelly. No other tool evaluated has such a clean distinction of tags and namespaces. The actual Jelly-tag is then responsible for adding the expressed information to the process schema. This is done once before execution. Afterwards Jelly is discarded and the process schema is reused by all instances. Further explanations go beyond the scope of this paper, it is only mentioned for completeness.
Summary and Outlook

In the course of this paper, we have introduced the Service Oriented Architecture and implementation concepts for a workflow engine based on the idea of Workflow Patterns.

A viable concept was shown and work is done on a proof of concept implementation. So far, the focus was on control flows and other areas belonging to a complete engine were only mentioned, if at all. A scope has more handlers than control flow and variable definition, i.e. event handling. The required interaction between all handlers has not been analyzed thoroughly, yet.

More importantly, most workflow languages cannot express all patterns directly, but offer workaround solutions. This will lead to suboptimal process schemas. Detecting these workarounds, possibly by Process Mining, and restructuring the process schema accordingly is future work.

Restructuring is one aspect of Process Planning, which also is future work. Looking closely at the engine design, the necessary preliminaries are already there, as the engine does not rely on an execution stack, but models an explicit scope stack processed non-recursively. Yet, little more than these preliminaries have been accomplished.

References

Abstract. Product lines are a natural part of day-to-day work in well-established engineering disciplines like mechanical or electrical engineering. Products sharing many features and differing only by some degree are developed together from a set of generic assets which are reused through all products and are capable to handle common as well as variable features. This approach leverages existing resources, increases productivity and improves product quality. In order to achieve these benefits in the software industry as well the sound principles of product lines must be transferred and adapted to the development of software products. This paper introduces the idea of software product lines, explains the core concepts, the phases of the associated software development process, characterizes currently used methods and identifies benefits, risks and problems.

1 Introduction and Motivation

In past and present times software engineers design software systems following the approach “only one system at a time”, that means although a software company may have many software systems developed in parallel, each project is run as standalone. As most software engineers know, each software product involves significant investments in requirements analysis, prototyping, architecture and design, documentation, process and method definition, tools, training, implementation and testing. Following the “only one system a time” approach these efforts have to be invested over and over again, anew for each software product to build – it is like re-inventing the wheel hundreds of times [1]. So software projects that overrun their schedules and budgets and are difficult to maintain are rather the standard than the exceptional case.

By taking a close look at the software industry one can observe that software companies usually have not one but multiple products on the market which mostly belong to the same application domain. So they will frequently share a large set of common features and differ only by some degree. Would it not be a fine idea to exploit the knowledge about these commonalities and variabilities to jointly develop products? What if these products would be developed based on a common set of generic assets, following a generic architecture and reusing large-grained software components?
Many software companies find that exactly this practice of building sets of related products together can yield remarkable technical, organizational and business advantages. These companies realize that they cannot longer afford to develop multiple software products “one system at a time”, because they need to introduce new products and add functionality to existing ones at a rapid pace, achieve large-scale productivity gains, maintain market presence, leverage existing resources and achieve mass customization [2]. They begin to understand the preconditions under which the reuse of former efforts and investments on conceptual as well as on implementation level is possible. They adopt a product line approach for their software products.

This paper gives an introduction (according to Merriam-Webster Dictionary prolegomenon, plural prolegomena, means prefatory remarks, specifically: a formal essay or critical discussion serving to introduce and interpret an extended work) to the idea of software product lines. It is not the aim of this paper to describe all aspects of software product lines in detail especially in respect to their concrete implementation, but to mediate an understanding of the basic concepts behind them. Therefore the three core concepts of software product lines are described in section 2. Section 3 takes a more detailed look on the first phase of the software product line development process. While the second phase of this development process together with basic management issues is examined in section 4. In section 5 five methods currently used for implementing software product lines are shortly characterized. Section 6 gives an overview about benefits, costs and problems of the software product line approach and section 7 concludes the paper.

2 Software Product Line Concepts

Within the software product line approach three key concepts can be identified which realize the peculiar aspects about it – the dual life-cycle model, the management of commonalities and variabilities across all members of the product line and the consequent reuse of assets on all levels of abstraction. These concepts are common to all current methods used for product line development. In this section these three key concepts shall be described by starting with setting up a common terminology that will be used throughout this paper.

2.1 Terminology

Setting up a terminology is necessary because different methods use certain terms in different interpretations, which in turn have changed over the years of product line practice. But while terminology may differ it is important to understand the concepts behind them.

A domain is an area of knowledge and expertise characterized by a set of concepts, problems, solutions and terminology understood by practitioners in the area and that is implemented through a number of products [3].

A feature is a logical unit of behavior that is specified by a set of functional and quality requirements [4].
A (software) product line is a set of (software-intensive) systems sharing a common, managed set of features that satisfy the needs of a particular market segment or mission [5].

A (software) product family is a group of (software-intensive) systems built from a common set of assets [5].

(Core) assets are those assets that form the basis for the software product line. A software asset is a description of a partial solution (such as a component, architecture or design document) or knowledge (such as requirements database or test procedures) that software engineers use to build or modify software products. Assets are collected in a so-called (core) asset base [3].

Mostly product lines are built as product families so the both definitions were merged and so were the terms as well. Today the following definition of Paul Clements and Linda Northrop from the Software Engineering Institute [6] which in given in [3] is commonly used: “Software product lines are sets of software-intensive systems sharing a common, managed set of features that satisfy the needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way.”

2.2 Dual Life-Cycle Model

The dual life-cycle involves two main processes – core asset development and product development. Core asset development is the process used to create artifacts useful across the entire product line, while product development is the process used to produce single products by adapting and tailoring the core assets produced during core asset development.

Analyzing the commonalities and variabilities of current and projected products, a technical architecture, an inventory and production strategy of pre-existing assets, architecture styles, design patterns and architectural frameworks are inputs to the core asset development. The outputs of the core asset development are a preliminary list of the products the product line will support, the core assets themselves and a production plan for how the core assets will be used in the development of single products.

Core assets often include the architecture, reusable software components, domain models, requirements statements, documentation and specifications, performance models, schedules, budgets, test plans, test cases, work plans and process descriptions – anything that can be reused across multiple product line members. Among the core assets the product line architecture and the software components are of paramount importance.

In the product development products are developed using the pre-build core assets. Therefore product requirements are developed and refined with the existing core assets in mind, reusable software components from the asset base are tailored to fit the specific needs of the product and assembled following the product line architecture and test cases are run on the product. So the outputs of product development are products that systematically reuse the core assets. It is important to note, establishing a core asset base does not mean, that every asset must be used in every product. This would lead to a “least common denominator”-kind of asset base, which is not nearly
so useful as a set of core assets that can be largely used in a fair number of products [7].

Usually high start-up investments \( I \) are required during core asset development and these investments first pay back during product development. Before introducing a product line there must be careful considerations whether and when the start-up investments pay back in order to make the software product line an economic success. Therefore the costs of products following the “one system at a time” approach \( (C_S) \) and the costs of products following the product line approach \( (C_P) \) must be estimated. Given the number of product line members \( (N) \) Equation (1) must be true – then introducing the product line approach makes (economically) sense:

\[
I < N \times (C_S - C_P)
\]

Or the other way round – if start-up investments and costs are known this equation helps to determine how many members the product line must have to make the start-up investments pay back – this correlation is displayed in Fig. 1.

**Fig. 1.** Cost estimations for product development with single-system and product line approach

There is no one-way relationship between core asset and product development – in the contrary there is a strong feedback loop between the two. There are two scenarios: in the first one a software company has already successfully fielded a product line and in the second one a company has multiple products in its portfolio and seeks to establish a product line from them. In the first case, the value of the core assets is realized through the products that are developed by using them and the core assets themselves are refreshed as new products are developed, for example by generalizing product-specific features or components and adding them to the asset base. And in the second case the core asset base must be developed or mined first from the existing legacy systems before new products can be built from the core assets.

In both cases there is a constant need for strong and visionary technical and organizational management deciding to invest the resources in the development of the
core assets and to develop the cultural change to view new products in the context of the product line [1] Fig. 2 visualizes the relationship of the two life-cycle phases.

![Fig. 2. Activities and work products of the dual life-cycle](image)

### 2.3 Managing Commonalities and Variabilities

When a software company adopts a product line approach it has to manage the commonalities and variabilities of the members of the product line. There must be careful considerations which features are shared by all members of the product line and which are not. Handling the common and variable features of product line members is critical to the success of the product line.

The assumption is, that all members of a product line have more features in common than they have differences, otherwise they would probably not represent a product line. Thereby a **commonality** is a quality or feature that all the products in a product line share, while **variabilities** are qualities or features, which only occur in some of the product line members [8]. A very simple example would be a car: all cars have wheels, brakes, lights, seats and so on – these are commonalities. All cars have a motor as well, but this can be a gasoline, diesel or electro engine. The motor is a commonality, but the concrete engine types are variabilities.

In Fig. 3 the relationship of the terms domain, product line, commonalities and variabilities is visualized. Within the domain there exists a product line with three members (P1, P2, P3). The rectangles represented the features of these three products. There is a space of common features (gray rectangle in the middle), which shared by all products – these are commonalities. But on the other hand these products differ as well – that are variabilities.

Commonalities have the highest reuse potential, because assets based on these commonalities are automatically reused in the product development. That’s why assets resulting from commonalities build the “core” of the core asset base. But for handling variabilities it is necessary to define places in the core assets where a variability can take effect. These places are called variation-points. During product development these variation-points are resolved, that means a concrete solution of the variability takes the place of the variation-point. Resolving variation-points in different ways leads to different variants, which are certain representation or
incarnation of the variability in the product line. Again the car example: in the car product line architecture there is an variation-point “motor”, this variation-point can be resolved with either “gasoline engine”, “diesel engine” or “electro engine” and this leads to three different car variants.

Typically there are three abstract types of solutions for a variation-point. The first solution is replacing the variation-point with an already defined alternative architecture, design sub-model or software component. This solution itself can contain variation-points, so resolution can be a recursive process. The second one is represented through a parameter, whose abstract value is substituted by a concrete value. And the third solution is just an open frame where the developer creates an own solution – this solution occurs when the range of variability cannot be foreseen during architecture or design phase [3].

Fig. 3. Domain with a product line containing three members

On the level of software component and product development there are a number of different techniques how to create and resolve variation-points, for example inheritance, delegation, aggregation, dynamic link libraries etc. In section 3.4 more examples are given. Based on the categorization of [9] there are six different types of variabilities: positive (feature is added), negative (features is removed), optional (0 or 1 out of 2 features is realized), alternative (n out of m features are realized), function (feature is changed) and platform (environmental conditions are changed).

As an additional characteristic every variation-point has a certain binding-time, when the resolution of the variation-point occurs. The choice of a binding-time limits the possible techniques for variation point resolution. Five typical binding-times can be identified: product architecture and component configuration, build (compile and link), installation, start-up and runtime. [8] The interrelations of the terms given before are displayed in Fig. 4.
Every variability can be characterized and described by answering four basic questions. Together with the type of variability this characterization gives hint to the core asset and product developers how to handle and implement variation-points in order to create a particular variant of a product line member.

1. What does varies and what are the possible variants? (What is the semantic of the variability? How do variants differ from each other?)
2. Where does the variability occur? (Which are the variation points involved? Which components are concerned?)
3. When is the variability expressed? (When are the according variation-points resolved?)
4. How will the variability be implemented? (What is the type of resolution expected for the according variation-points?)

2.4 Reuse on all Levels of Abstraction

When applying the product line approach, reuse does not only happen on the level of implementation but on a conceptual level as well. Former reuse approaches mainly target mainly small-grained components. Software companies have built reuse libraries containing algorithms, modules, classes or components. Nearly everything a developer produces goes into that library. From a developer’s point of view locating these small pieces and integrating them into a system can take longer than to build them anew. The benefits of small-grained reuse depend on the predisposition of the software engineer to use what is in the library, the suitability of the items is in the library, the documentation and the integration of the library units into the rest of the system.

This kind of reuse is called opportunistic reuse and following this approach “clone and own” is a very common activity, “Clone and own” means, a developer locates a component in the library, takes the programming code (“clone”) and changes it to
fulfill his needs (“own”). This changed component might have little in common with the source component and has to be maintained separately [3].

In the software product line approach the reuse is in contrast planned, enabled and enforced – the opposite of opportunistic. The asset base includes those artifacts in software development that are most costly to develop from scratch – namely, the requirements, domain models, software architecture, performance models, test cases, software components, etc. All of the assets are designed to be reused and are optimized for use in more than a single system – not only the architecture and software components are prepared to handle variabilities by providing variation-points.

During the actual product development the reusable core assets from the asset base are used, but “clone and own” does not happen now, because the assets are equipped with variation-points and can be tailored by various strategies (depending on the implementation platform) to their specific use in the product. Core assets are used “as is” and if a core asset must be changed, the changed asset goes back into the core asset base, so all developers work from the same basis.

3 Core Asset Development

As described in section 2.2 before the product development process of the software product line approach comprises of two plus one activities – the dual life-cycle activities core asset and product development, which are both under the supervision of (technical and organizational) management. These three essential activities are closely linked and highly iterative. Core asset and product development can occur in any order – products can be built from core assets, like the other way round core assets can evolve through mining and re-engineering of existing products.

Core assets are the basis for the development of products in the product line. Central core assets are the architecture that the members of the product line will share, as well as software components that are developed for systematic reuse across the product line. Requirements specifications and domain models are core assets, as is the statement of the product line's scope. Each core asset should have associated with it an attached process that specifies how it will be used in the development of actual products. That plan itself is a core asset.

The goal of the core asset development activity is to establish a production capability for products and contains itself a number of sub-activities, which are described in this section.

3.1 Domain Analysis

Domains are areas of expertise or knowledge, which is characterized by a set of concepts and terminology understood by practitioners. It also includes an understanding of recurring problems and known solutions within the domain that can be used to create a single as well as a set of products. To build products it is usually required to gather knowledge from several domains. In the context of product lines it is therefore helpful to differentiate between the application domain of the product line
and associated rather technical sub-domains. Before an organization starts implementing a product line it has to achieve a decent knowledge about the domains that are concerned. According to [3] understanding the relevant domains is based on four steps:

1. Identifying domains relevant for the product line.
2. Identifying the recurring problems and known solutions within these domains.
3. Identifying features of products, which can occur in these domains.
4. Capturing and representing the information in ways that allow to be communicated to the stakeholders and used across the entire product line efforts.

Understanding the relevant domains is the first step to understanding the commonality and variability that can be expected to occur among the products identified in the product line's scope. Therefore the application domain and all associated sub-domains are analyzed in respect to the features they provide for products. Features are usually represented in tree-like hierarchies, have interrelations and dependencies, some are optional or alternative, others mandatory, some forbid to occur with others at the same time etc. Features, which were identified as mandatory to products in a domain are commonalities, while optional or alternative features are variabilities among the members of the product line. The gathered domain expertise should be documented in the form of a domain model. Following [3], this will be especially useful to answer the following questions:

1. Which features tend to be common across systems in the domains and what variations are present? (This information will inform the process of scoping, in which the commonality and variations for the product line will be established.)
2. Which subsets of features might be packaged together as assets for the product line? (This information will inform the architecture creation effort for the product line, by suggesting potential subsystems that have occurred in other systems in the domain).
3. What constraints apply to systems in the domain(s)? (This information will probably influence the architecture and component development by identifying non-functional requirements like standards, legal restrictions, business constraints, specific hardware platforms, etc.)
4. Which assets typically constitute members of the domain(s)? (This information suggests a list of assets that an organization could begin to search for in its own legacy inventory or on the open market.)
5. Whether or not to continue with the product line development effort? (This information can help management gain confidence in the soundness of the decision to adopt a product line approach.)

3.2 Scoping

Scoping is the activity of setting the boundaries of the product line within the concerned domains (see Fig. 3). This activity is based on the information from the domain analysis (especially the feature model) and its goal is to determine what feature is “in” what is “out” according to one or more members of the product line. This is highly necessary since there cannot be a product line where everything is “in”,
because such a product line would cover all the concerned domains, which is economically and technically hardly manageable.

Setting the scope of a product line is critical because the danger of mis-scoping is imminent. If the scope is too large, the members of the product line will vary too much, the core assets will have to be so generic (because there are more variabilities than commonalities) that they will be useless and the product line will collapse into the old “one system at a time” product development. If the scope is too small, the core asset may not be generic enough to accommodate future growth and the product line may not have enough customers to recoup the investment in the core assets. And if the scope bounds the wrong products, the product line will not find a market.

The main inputs for the scoping activity are information about market drivers and competition from market analysis, business goals set by the marketing department and decent understanding of the application domains expressed in domain models. The result of scoping is a document containing decisions what product will or might in the future constitute the product line and what products will be explicitly excluded. Within this document there will usually be some kind of feature-product-matrix, listing all features (rows) and all existing or envisioned product (columns) and displaying what feature will be contained in what product. Like most product line activities scoping must continue after the initial scope has been defined because new market opportunities may arise.

3.3 Requirements Engineering

The product line scope bounds the products included in the product line, the product line requirements engineering refines the scope by more precisely defining the characteristics of the products in the product line.

In the context of product line development two kinds of requirements have to be regarded: requirements that are common to the entire product line (commonalities) and those specific only to single members of the product line (variabilities). There must be a clear distinction between the two kinds – especially the common requirements constitute an important and tangible core asset and should be maintained separately from requirements that are particular to single products.

Variabilities in requirements specifications are again expressed in terms of variation-points. By resolving the variation-points they can be turned into product-specific requirements during product development. These variation-points can be rather small-grained, such as providing a symbolic like "max_nbr_of_users" that can be resolved with a concrete value such as "150" or may be substantial, such as leaving a placeholder for an entire, later specification.

The distinction between common and product-specific requirements is necessary because common requirements are usually met by core assets, which are used without modification, while product specific requirements are met by assets, which must be either tailored, adapted or newly produced during product development. So the complete set of requirements for a particular product is defined as the set of common requirements (with variation points) plus product-specific requirements that are expressed relative to the product-line common requirements [3].
The product line approach requires a capability to trace requirements to core assets as the architecture, design models, software components and to parts of the delivered products as well. This is needed to prove to customers and higher management, that up-front investments in domain analysis and architecture are actually leading to measurable efforts and that software assets and products meet their requirements.

3.4 Software Architecture

The product line architecture is central to the core asset base used to construct and evolve the products in the product line. According to [11], a software architecture of a computing system is the structure or structures of the system that consist of software components, the externally visible properties of those components and the relationships among them. So does the product line architecture – it specifies the structure of the products in the product line and specifies the interfaces for the software components that will be in the asset base.

Products in a software product line exist simultaneously and may vary from each other in terms of their behavior, quality attributes, platform, network, physical configuration, middleware, scale factors and lots of other ways. Therefore the software architecture will satisfy the common product line requirements and those specific for the individual products in particular by explicitly admitting a set of variation points required to support the spectrum of products within the scope. According to [3] variation-points can be built into the product line architecture by mechanism as described in Table 1.

<table>
<thead>
<tr>
<th>Variation mechanism</th>
<th>Situation when used</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inheritance and polymorphism</td>
<td>An object or method needs to be implemented differently for each product in the product line.</td>
</tr>
<tr>
<td>Extensions and extension points</td>
<td>Parts of a component can be augmented with additional behavior or functionality.</td>
</tr>
<tr>
<td>Parameterization</td>
<td>A component's behavior can be abstractly characterized by a placeholder, which is defined at build time. Macros and templates are forms of parameterization.</td>
</tr>
<tr>
<td>Abstract interfaces</td>
<td>Choosing between different implementations, which are all faithful to the interface.</td>
</tr>
<tr>
<td>Configuration and module interconnection languages</td>
<td>Definition the build-time structure of a system, including selecting (or deselecting) whole components.</td>
</tr>
<tr>
<td>Generation</td>
<td>A higher-level language can be used to define a component's desired properties.</td>
</tr>
<tr>
<td>Compile-time selection of different implementations</td>
<td>The variable <code>ifdef</code> can be used when variability in a component can be realized by choosing different implementations.</td>
</tr>
</tbody>
</table>
Identifying the allowable variation-points is part of a product line architecture's responsibility as well as providing built-in mechanisms for achieving them. For many other system qualities, such as performance, availability, functionality, usability and testability, there are no major peculiarities that distinguish architecture for product lines relative to single-system developments.

The documentation of the product line architecture, which is as core asset itself carries two obligations. The first is to document the product line architecture in respect to different architectural views. A view is a projection of the architecture that includes certain kinds of information and suppresses other kinds. For example, the process view will show the processes in the software and how they communicate or synchronize with each other. The layered view will show the layers of the architecture, but will not show the processes. The second documentation obligation is to describe the architecture's attached instantiation process by describing the architecture variation points, their rationale and how to resolve them.

3.5 Software Components

One outcome after completing the software architecture is a list of components that will populate the architecture. This list is given to development, mining, and acquisition teams, which will supply the software components the system is made of. Szyperski gives in [12] a definition of the term component as follows: “A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.”

In the context of software product lines components are the units of software that go together to form products, as dictated by the product line architecture. In the definition given above, "deployed independently" means that software components are installed into a product line's core asset base and thus are made available for use in one or more products. The "third parties" are the product developers, who compose the component with others to create systems. The “contractually specified interfaces” are of paramount importance, to ensure that components fit together across the usage in different product line members.

In the context of software product line the definition given above must be expanded with the aspect of handling variabilities. Components that are included in the core asset base must support the variabilities by setting variation-points specified in the product line requirements and architecture. Components from the core asset base should not be modified during product development, but should be used “as is”. If they must be modified, it suggests they were not specified correctly. The reason for that is – any modification would have been repeated every time a new release of the particular core asset component is incorporated into the product being developed. This would result in additional work for every release of every modified component.

So if no modifications to components from the asset base are allowed, what techniques instead should be used in order to provide variability on the level of implementation? The following enumeration provides some of them (not every technique can be used with any programming language (aggregation and delegation, inheritance, parameterization, method / function overloading, dynamic class loading,
static libraries, dynamic libraries, conditional compilation, C++ template classes, reflection and XML / XSLT techniques.

3.6 Production Plan

A production plan describes how products are produced from the core assets. As noted above, each core asset should have an attached process that defines how it will be used in product development. The production plan is essentially a set of these attached processes with the necessary “glue”. It describes the overall scheme for how these individual processes can be fitted together to build a product. It is, in effect, the reuser's guide to product development within the product line. It ensures, that each product in the product line will vary consistently with predefined variation-points.

Without the production plan, the product builder would not know the linkage among the core assets or how to utilize them effectively and within the constraints of the product line.

Production plans can range from a detailed process model to a more informal guidebook. The production plan should describe how specific tools are to be applied in order to use, tailor and evolve the core assets. The production plan should also define metrics to measure organizational improvement as a result of the product line practices and the plan for collecting the data to feed those metrics.

4 Software Development Considerations

Following a product line approach means in short words, that each member of the product line is formed by taking applicable software components from the asset base, tailoring them through preplanned variation mechanisms, adding any new components if necessary and assembling the collection according to the rules of the product line architecture. Thereby building a new software product becomes more a matter of assembly of already existing assets than one of creation – the predominant activity is integration rather than programming [3]. This idea of product development together with a short overview over software product line management issues is examined in this section.

4.1 Product Development

The product development activity depends on the outputs from the core asset development: the product line scope, the core assets and the production plan – plus the requirements for the individual products.

In a product line, the effort involved in software system integration lies along a spectrum. At one end, the effort is almost zero. If all of the products’ variabilities are known in advance, an integrated template of a generic system with formal parameters can be produced. An elaborated production plan specifies all variation-points and how they can be resolved. The final products are generated by supplying the actual parameter values specific to the individual product requirements and then launching
the construction tool (for example the UNIX "make" utility). In this case, each product consists entirely of core components; no product-specific code exists. This is the "system generation" end of the integration spectrum.

At the other end of the spectrum, considerable coding may be involved to bring the right core components together into a cohesive whole. Perhaps the components need to be wrapped or perhaps new components have to be designed and implemented especially for the product. In addition the production plan might be more like an informal document that must be adapted to the needs of the particular product. In this case, the integration more closely resembles that of a single-system project.

Most software product lines occupy a middle point on the spectrum (see Fig. 5). Obviously, the closer to the generation side of the spectrum the production approach is, the easier integration will be and the more products can be turned out in a short period of time. Production circumstances may prevent achieving pure generation. Perhaps a new product has features, which were not considered or the application domain prevents knowing all of the variabilities up front. Or perhaps the variabilities are so numerous or complex or interact with each other in such complicated ways, that building the construction tool will be too expensive.

Integration assumes a greater role for software product lines than for single-system development simply because of the number of times it is performed. A product line with a large number of products and upgrades requires a smooth and easy process for each product. Therefore, it pays to select a variation mechanism that allows for reliable and efficient integration when new products are turned out. This means some degree of automation. For example, if the variation mechanism chosen for the architecture is component selection and de-selection, an integration tool that carries out the requirements by selecting the right components and feeding them to the compiler or code generator is needed. If the variation mechanism is parameterization or conditional compilation, an integration tool that checks the parameter values for consistency and compatibility, then feeds those values to the compilation step. Hence, the variation mechanism chosen for the architecture will go hand-in-hand with the integration approach [3].
4.2 Management and Software Process Issues

Management plays a critical role in the successful fielding of a product line as all activities must be given resources, coordinated and supervised. The core assets and the plans for how they are used to build products do not just materialize without planning and they do not come for free. They require organizational foresight, investment, planning, direction and a strategic thinking that looks beyond a single product. The disciplined use of the assets to build products does not just happen either. Management must direct, track and enforce the use of the assets. Software product lines are as much about business practices as they are about technical practices. Management at both the technical (or project) and organizational levels must be strongly committed to the software product line approach.

Technical management oversees the core asset and product development activities by ensuring that the groups who build core assets and the groups who build products are engaged in the required activities, follow the processes defined for the product line and collect data sufficient to track progress. Project progress is evaluated through systematically collected metrics that are of great importance for product lines. Because an approach like software product lines requires radical changes in the enterprise and there is a temptation for management to give up, especially in the early stages, before clear ROI data are available. The metrics include: time to market, percentage of reused software, lines of code per programmer, increased number of products shipped, product volume shipped, number of new features released per year, product volume shipped per person, etc. [1].

Organizational management must establish the proper organizational structure to support product lines and must make sure that the organizational units receive the right resources in sufficient amounts. Organizational management determines a funding model that will ensure the evolution of the core assets and then provides the funds accordingly. Organizational management also orchestrates the technical activities and iterations between the essential activities of core asset development and product development.

Both technical and organizational management also contribute to the core asset base by making available for reuse those management artifacts (especially schedules and budgets) used in developing products in the product line.

Finally, someone should be designated as the product line manager and that person must either act as or find and empower a product line champion. This person must be a strong, visionary leader who can keep the organization squarely pointed toward the product line goals, especially when the going gets rough in the early stages. Leadership is required for software product line success.

5 Current Methods

The concepts discussed so far reside on a rather conceptual level, but in order to put the product line approach into practice these concepts have to be manifested in specific and concrete methods. In this section a number of methods and models are presented, that cover especially the software engineering part of product lines.
Common to all methods are the three key concepts described in this paper – the dual life-cycle, managing commonalities and variabilities and reuse in order to produce a set of core assets from which particular products can be derived. The methods differ in scope, visualization paradigm, terminology and support for the product line key activities. It is not the aim to describe the methods in detail, but to shortly present them with their contribution to the product line success.

5.1 FODA

Feature Oriented Domain Analysis is a method developed 1990 by Kyo C. Kang et. al at the Software Engineering Institute and is described in detail in [13]. FODA is regarded as the mother of most of today’s product line methods. This method focuses on domain analysis in order to gather information about commonalities and variabilities of systems all belong to a certain domain. Domain analysis is divided in three main activities – context analysis, domain modeling and architecture modeling (see Fig. 6) – each of these activities is carried out via certain techniques and FODA describes the process in which these techniques are carried out.

Fig. 6. Parts and models of FODA

During context analysis the scope of the domain (that is likely to yield the future domain products) is defined. Therefore relationships between the candidate domain and the elements external to the domain, availability of domain expertise, constraints, common problems and their solutions are analyzed and evaluated.

The most influential and popular technique of FODA resides within the domain modeling phase: the feature analysis. The purpose of feature analysis is to capture the stakeholders’ understanding of the general capabilities of applications in a domain. Since the primary interest is in the commonality of a family of applications, the feature model captures the common and variable features of the applications in the domain.

The outputs of feature analysis are feature diagrams represented as trees in which features build hierarchies. A node in the tree is interpreted as a feature that is refined by several sub-features. A leaf is a feature that shall not be further refined. Features can have three types of relationships – mandatory (a feature is a definite part of the
super-feature, represented by a simple line), optional (a feature can be part of the super-feature but does not have to be, represented by the circle at the line’s end) and alternative (one out of n features must be part of the super-feature, represented by an arc between several lines). Two types of cross-tree relationships (composition rules) are needed to define relationships between features that do not have a super-/sub-feature relationship – requires (one feature requires another) and mutex (features mutually excludes each other). Fig. 7 shows an exemplary feature diagram for an automotive domain.

![Feature Diagram for Automotive Domain]

**Fig. 7.** Part of a feature diagram for an automotive domain

The purpose of architecture modeling is to provide a software solution that realizes the features defined in the domain modeling phase. A FODA architecture model is a high-level design of the applications in a domain. Therefore, the FODA method focuses on identifying concurrent processes and domain-oriented common modules, and on allocating the features, functions and data objects defined in the domain model to the processes and modules.

FODA has become a bit old-fashioned over the past 10 years. Many recent developments like component-based modeling are for obvious reasons not incorporated in FODA. Although FODA has a defined architecture modeling phase it lack a specific (visual) description language for architecture and design models. Another aspect is FODA’s orientation towards domains instead of products – domains are problematic to scope and engineer because a domain captures many extraneous elements that are of no interest to an software company that wants to field a product line.

**5.2 PLA**

Product Line Analysis is another approach proposed by the Software Engineering Institute that focuses on requirements engineering of a product line. A detailed description is given in [14]. The method utilizes techniques from domain analysis with FODA and object technology namely use-cases as described in [15] but that focused on the needs of product lines rather than domains.
The requirements models for a product line include currently needed capabilities and associated product variations. Since requirements will change over the life of the product line, product line analysis must also incorporate (at least placeholders for) anticipated requirements in the requirements models.

Requirements are elicited, analyzed, specified, verified and expressed through four different types of models. Common to all these models is that they comprise the commonalities and variabilities of product line members in their own terms.

1. The use-case model specifies the product line stakeholders and their key interactions with the product line.
2. The feature model specifies the stakeholders’ views of the product line.
3. The requirements-object model specifies the product line responsibilities that support those features.
4. The dictionary defines the terminology utilized in the work products and supports a consistent view of the product line requirements.

Fig. 7 shows the relationships between use-case, feature and requirements-object model. For each feature in the feature model, there is a set of requirements objects in the object model that realize that feature. Combinations of requirements objects and the exchanges of information among them satisfy the goal of each use case. The responsibilities identified by each use case are incorporated in the requirements objects. The dictionary contains definitions for all feature names, use-case goals, requirements object names and the names of the information exchanged between requirements objects. The four work-products are generated in a process that consists of four steps: initialization, recursive refinement, analysis (commonality and variability, consistency, feature-interaction, model quality, requirements priority) and verification.

The problem of PLA is its very narrow scope. PLA is limited to domain analysis and requirements engineering phase. It does not give any hint how to derive an architecture from the requirements or how to create software components.
5.3 FAST

The Family-Oriented Abstraction, Specification and Translation process has been developed by David Weiss at Lucent Technologies Bell Laboratories. The method is in-depth described in [16]. FAST is based on three hypotheses. Together these hypotheses suggest that it is possible and useful to find a family among similar programs. The commonalities between several programs reveal a family, while their variabilities show the boundaries of the family.

1. The redevelopment hypothesis claims that software development is very often redevelopment. It consists of creating new variations on existing software systems. Usually, there are more similarities than differences between variations.
2. According to the oracle hypothesis, it is possible to predict the changes that are likely to concern a software system. Future changes can be derived from earlier changes.
3. The organizational hypothesis concerns both software and software development organization. Each of them can be organized to take into account of predicted changes regardless of the type of changes.

![Fig. 9. FAST software process pattern](image)

The FAST process can be divided into three steps or sub-processes – domain qualification to identify families worthy of investment, domain engineering to invest in facilities for producing family members and application engineering to use those facilities to produce family members rapidly (see Fig. 9). With each sub-process different roles, activities and work products are associated. Some activities may proceed concurrently while some others may require a particular order. A more
precise process description of FAST activities is given in the PASTA (Process and Artifact State Transition Abstraction) model, which is also described in [16].

Remark: FAST uses the term product family like given in section 2.1 of this paper; domain and application engineering can be seen as conceptual equivalent to core asset and product development.

The sub-process domain qualification analyzes a software family from an economic perspective. It estimates the number and value of family members and the cost to produce them, in order to create a sound economic basis for the product family.

The domain engineering sub-process analyzes how the products of the product family share the common basis and how they differ from each other, develops and acquires the core assets of the product line. The purpose of domain engineering is to specify and implement the application engineering environment.

Among the activities comprising domain analysis the creation of a decision model and the design of the application modeling language (AML) are important. The decision model defines the decisions that an application engineer must make to specify and produce a new member of the product line. The AML shall be capable of expressing requirements and decisions in a more formal way. FAST offers two approaches for generating family members from the AML: compilation and composition. The composition approach creates a modular design for the family and implements each module as a template. A composer is needed to generate family members by composing completed templates. Composition approach requires a software design that is common to all family members and acts as a basis for generating family members. Compilation approach requires building a compiler including a parser for the AML, a semantic analyzer for the parsed form and a code generator.

During domain implementation an environment that satisfies the domain model is developed. Domain implementation includes providing the toolset as well as creating a library of templates which form the application engineering environment. This environment is used to produce applications in the application engineering sub-process. The application engineering environment enables analyzing the specifications written in AML and provides ways to generate code from the model that means to map the abstraction into an implementation.

Application engineers use the application engineering environment provided by domain engineers to produce applications of a family quickly. After identifying and refining the requirements for the application the application engineer represents the requirements as an application model in AML. According to the application model, the application engineer uses the toolset to generate a deliverable set of code and documentation.

5.4 PuLSE

Product-Line Oriented Software Engineering is developed at the Fraunhofer Institute for Experimental Software Engineering (IESE) [17]. An introduction to PuLSE is given in [18]. PuLSE is an abstract framework for developing and managing software
product lines and consists of three parts: deployment phases, technical components and support components (see Fig. 10).

The deployment phases describe the product line life-cycle: initialization, product line infrastructure construction, usage and evolution. The second and the third step are equivalent to core asset and product development, but the other two are particular to PuLSE. During these phases several technical components are used which contain the technical know-how needed to operationalize the development of the product line. Support components capture the experience with the PuLSE approach in the context of different organizational structures, different mature organizations and different entry points.

Based on the observation that universal models are not appropriate – every deployment phase and technical component is customizable to the context in which they will be applied in order to ensure that process and products match their requirements. This customization takes place in the initialization phase. During infrastructure construction the product line infrastructure is scoped, modeled and the architecture is defined. This infrastructure is used to create product line members during infrastructure usage. The task of evolution and management is evolving the infrastructure over time and manage it.

The technical components are used throughout these deployment phases. Every technical component defines activities, a customizable plan how they can be performed and related work products. The following list describes in short words each technical component, its abbreviation and the associated deployment phase.
Customizing (PuLSE-BC) – Initialization. In Customizing an instance of the PuLSE methodology is tailored to the enterprise context in which it will be applied. Therefore customizing comprises three parts – baselining (gathering information), evaluation and customization.

Scoping (PuLSE-Eco) – Construction. Scoping is used to determine anticipated product line members and map out their characteristics from an economic viewpoint. The outcome is a kind of feature-product matrix called product map enriched with evaluation functions and benefit analysis.

Modeling (PuLSE-CDA) – Construction. During Modeling the product line concepts and their interrelationships are elicited, structured and documented. It refines the economic scope by creating so-called storyboards (action sequences as workflow diagrams or message-sequence charts) and setting-up a decision model.

Architecting (PuLSE-DSSA) – Construction. Architecting supports the definition of a product line software architecture, which covers current and future products as described by the product map. The basic idea is incremental development of the architecture guided by scenarios. Scenarios are derived from storyboards and come as generic (functional requirements) and property-related scenarios (quality requirements). Implementation-specific decisions and possible resolutions are captured in a configuration model, which extends the decision model.

Instantiating (PuLSE-I) – Usage. Instantiating aims at planning, specifying, instantiating and validating one member of the product line. Therefore a product plan is set up based on the stakeholders’ requirements in order to determine how many existing assets can be reused. Driven by the decision and configuration model the product line model and architecture is instantiated and validated. Then the low-level design and the programming code are developed by either reusing or creating new assets. No modifications to the core assets are allowed during the Usage phase – all change requests are handled by PuLSE-EM.

Evolving and Management (PuLSE-EM) – Evolution. Evolution and Management is used to monitor and control the evolution of the product line infrastructure, which is built in the Construction phase, over time. Therefore it coordinates the activities of the other PuLSE components – gathers work products, consolidates, evolves, maintains them and handles change requests.

The strength of PuLSE – its customizability and abstract character – is at the same time its greatest weakness. PuLSE describes the software process and the work products in abstract terms. It does not give hints how and by which means to describe the architecture or how to incorporate flexibility in the software components on the level of implementation. PuLSE is quite uncertain about reuse of software components (produced in the construction and used in the usage phase), is not component-oriented model and therefore does not explain how to implement variability. In order to apply PuLSE, an organization has to pick concrete methods and techniques (that might be already existing in the company) that instantiate the abstract parts of PuLSE.
5.5 KobrA

Komponentenbasierte Anwendungsentwicklung (Component-based Application Development) is developed like PuLSE at the Fraunhofer Institute for Experimental Software Engineering. KobrA is described in depth in [19]. KobrA is a process model for component-oriented development of application frameworks. It makes components the focus of the entire software development process, not just the implementation and deployment phases and adopts a product-line strategy for their creation, maintenance and deployment. Components are described with a full, UML-based representation that captures their entire set of characteristics and relationships. This not only makes the analysis and design activities component-oriented and independent of (but compatible with) specific component implementation technologies.

The central artifact in a KobrA project is the framework. A framework provides a generic description of the software elements making up a product line. Generic means, that a KobrA framework embodies all variabilities within a product line, not just the commonalities. Therefore all possible features are captured within the framework and decision models are used to describe the choices that distinguish distinct members of the product line.

---

![Diagram of KobrA Komponent and its Associated UML Diagram Types](image)

**Fig. 11.** A KobrA Komponent and its Associated UML Diagram Types

In KobrA everything is treated as a component – on the highest level even the product itself is a component. In this sense the framework is a tree-structured hierarchy of components, in which the parent/child relationship represents...
composition (a parent is composed-of its children). To distinct the KobrA idea of a component from the common notion of a software component KobrA uses the term Komponents (KobrA component) instead. KobrA is based on four principles:

1. Uniformity. All behavior rich elements should be viewed as Komponents; thereby Komponent development becomes Komponent assembly.
2. Parsimony. Only a minimal set of concepts used to avoid redundancy.
3. Locality. All models should be local to a Komponent. There are only very little global models.
4. Encapsulation. Komponent specifications (what) must be separated from component realizations (how).

Komponents are described from two viewpoints – the specification and realization viewpoint (see Fig. 11). The specification describes the externally visible characteristics of the Komponent and defines the requirements that it is expected to meet. The realization describes how the Komponent satisfies these requirements in terms of interactions with lower-level (sub-) Komponents. The overall framework consists of a set of Komponent specifications and realizations inter-related by carefully controlled consistency, traceability and realization relationships. Komponents are recursively development. Every Komponent, regardless of its granularity or location in the tree, is manipulated using the same basic set of concepts.

Once a framework has been completed, specific applications can be instantiated from it by the resolution of the decision models. This is done recursively as well because the Komponents’ decisions model are also structured in a tree-like manner. The result is an application with the same structure as the framework, but with all generic aspects and not-required features removed.

The parent/child relation between Komponents and their tree-like structure is critical to the success of KobrA. If errors and quality problems occur near the top of the tree this can have a disproportionate effect on the quality of the overall product and the success of a project. KobrA therefore provides certain techniques – from inter-model consistency checks, inspections, traceability models and early tests to ensure the quality of the framework.

6 Discussion

Implementing the three main concepts of the software product line approach – dual life-cycle model, management of commonalities and variabilities and reuse on all levels to abstraction in all steps of the software engineering process leads to success. But as there are benefits there are risks and costs as well. These aspects together with a discussion of current problems of the software product line approach are described in this section.

6.1 Benefits and Costs

During establishing the product line core assets usually high start-up investments are required. But if the product line is carefully scoped and the core assets are
intentionally designed to be reused and capable of handling product variabilities these investments pay back during the development process of single products.

Once the core asset base is established savings and benefits can be achieved in all of the steps of software development process. Beginning with requirements engineering, an extensive phase for each single product can be saved because the core asset base already contains requirements common to all members of the product line plus requirements specific to single products. Based on the requirements the product line architecture is developed, so the most important design step for any product of the product line is largely completed and the architecture can be used for each product and only need to be instantiated. Then software components in the asset base are used as the single products’ building blocks either unchanged or adapted using different variation mechanism. This kind of reuse emphasizes more on composition of existing components than building them anew. With each new product there is extremely high confidence that timing and performance problems have been worked out and that the bugs associated with real-time and distributed computing – scheduling, data consistency problems, synchronization, network load and absence of deadlock – have been eliminated. Generic test cases, data, plans and processes have already been built and only need to be tailored on the basis of the variation-points related to the product. Production plans, baseline budgets and schedules from previous product development projects have already been developed and provide a reliable basis for the product work plans. Configuration control boards, configuration management tools and procedures, management processes and the overall software development process are in place, are robust, reliable and responsive to the organization's special needs. Tools and environments purchased can be amortized across the entire product line. Fewer people are required to build products and the people are more easily transferred across the entire line because of the commonalities of products and the production process.

With these benefits also come costs. Introducing product line in a software company is more than just a technical decision; it is a business decision as well. As mentioned before the introduction implies heavy start-up investments to produce the core asset as well as ongoing efforts to maintain the asset base. The investment may be measured in dollars spent in building the assets and training the staff or it may be measured in lost business because the company’s output declines during product line introduction [5].

These costs mainly originate from the fact that most core assets must be capable to handle commonalities and variabilities. Early steps in the development process that are sometimes omitted because of a lack of time (domain analysis, scoping, requirements engineering or architecture definition) have to carefully worked out in order to achieve all the benefits in the later steps. It simply takes time, people and money to create the core asset base and the management must be convinced, that the investments hold their promise of future benefits.

It takes a certain degree of maturity in the software company to field a product line successfully, because the cultural and technical implications can be far-reaching, for example reaching CMM maturity level 2 in configuration management is regarded as necessary [20]. A deep understanding of requirements engineering is necessary to build the basis for the core assets, as well as the architecture has to be defined be experienced architects and so on.
Technology change is not the only barrier to successful product line adoption. Changes in management and organizational practices are also involved. Successful adoption of the software product line approach is a mix of technological, process, organizational and business improvements.

6.2 Problems and Risks

As a relatively young approach software product lines have to deal with problems at various levels of severity. During the core asset and product development many activities like architecture definition or component implementation can be carried out wrong or incomplete – these are no particular problems of product lines since this can happen in single-system development as well. But there are some serious problems to the approach that can endanger the product line success in short or long term. These risks are inherent to the product line approach and exist due to the nature of this approach. In order to make a product line a success these problems must be solved.

Handling Product Line Architecture and Core Asset Evolution. A software product line evolves from its initial release of the architecture, design, adaptable software components and other core assets. The evolution of core assets results from the same factors causing the evolution of any system – desire to incorporate technological change, repair of existing problems, addition of new functionality or restructuring of existing functionality to allow for more variants. Propagation of changes in core assets to multiple deployed products in the product line requires more configuration management maturity. If the core assets are not updated to reflect the latest change requests, variants will be created from products rather than from the product line assets and, over time, components will not be reusable and approach’s benefits will be lost [21].

Synchronizing Products with New Releases of the Core Asset Base. There are a number of products derived from the core assets of a peculiar version, when core assets are changed through evolution. It is clear, that new products are developed from this new version of the core asset base. What happens if a new release of an old product becomes necessary? Is this task accomplished with the old core assets, which would be probably cheaper? Or are the new ones used which is probably more expensive because of necessary adoption of the product to the new core assets? The former case is not intended by the product line approach because it would lead to different versions of the core asset base that would have to be maintained separately and that would be very similar to single-product-development. But maintaining multiple versions of a core asset can be necessary due to backward compatibility.

Ensuring Conformance of Products with the Core Asset Base. During construction, systems tend to drift away from the original architecture and the original design. It must be ensured that assets from the core asset base are used “as is” with the utilization of the prescribed variation-points. Project managers must prevent that old “clone and own” approach used by product developers. When maintenance is needed on the original source fragment, it is very likely it will also be needed on all of its copied and modified versions as well. This is one of the worst cases of reuse; it is in fact a detriment to asset evolution. Product developers as well as whole companies must overcome the “not invented here”-syndrome which manifests itself as a
reluctance to accept the core assets of a product line developed by a third party and to evolve them. The desire to start a “green-field” development, rather than rely on the work of others (which may be difficult to understand and hence difficult to modernize), may be too great.

**Tool Support.** During the Fourth Product Line Workshop [22] held by the SEI the problem of insufficient tool support was discussed in detail. There is currently a lack of tools that support composition of products (semi-) automated from core assets by tailoring them based on certain criteria of the product. Another aspect is automated conformance and consistency checking. Is the product conformant with the architecture? Are all resolutions to variation-points consistent with each other? Tools capable of generating code from core assets are lacking as well. Because there is no “one fits all” tool a broad variation of tools (including COTS and company-intern tools) used today, so that tool-interoperability (necessary for traceability) arises as another problem.

**Achieving Decent Domain Knowledge and Avoiding Mis-Scoping.** The scope of the product line must target the right products, as determined by knowledge of similar products or systems, so a decent knowledge of the application and technical (sub-) domains of the product line is necessary. This knowledge can be gained by a thorough domain analysis (see section 3.1) if the organization has only little experience in the concerned domains. But as software companies do not really do “green-field” development of product lines, because of existing legacy systems, they start core asset development by mining existing products. Because of the importance of deep domain knowledge, “green-field” efforts suffer from a lack of initial feasibility proof [1]. There is a major risk if the scope of the product line is not properly determined. Too broad a scope renders the core assets too complex and too generic to be effectively reused – too narrow a scope does not justify the cost of core-asset development and maintenance [1].

7 Conclusion

Software engineering is a relatively young part of computer sciences and does not benefit from the legacy of rules and codified standards and practices found in other engineering disciplines like mechanical, electrical or civil engineering. In these disciplines product lines are a natural part of the day-to-day work – there is a huge number of prominent examples ranging from the automobiles, computer hardware to house building. In those domains product lines already help to achieve large-scale productivity gains, decrease time-to-market, increase product quality and customer satisfaction, efficiently leverage existing resources, maintain market presence and sustain future growth. Past and current research efforts have been placed at the transferring sound and established product line principles to software engineering in order to achieve to same goals for the software industry.

In this paper the three core concepts of software product lines are described – the dual life-cycle, comprising the parts of core asset and product development, the management handling of commonalities and variabilities across members of a product line and the reuse on all levels of abstraction involving not only reuse of
programming code, but of requirements, architecture, designs etc. Then the parts of core asset development, which are relevant to software engineering, are examined more in detail. Special emphasis is put on the question how to handle variabilities within the various steps and work products of core asset development. Based on the core assets single products are produced in the product development phase. In contrast to single-system development product development here becomes more a matter of integration and composition of pre-existing components. Some aspects of technical and organizational managements are also concerned, since introducing a software product lines is not only a technical but a business decision as well. The number of methods dealing with software product lines increased steadily since the early 1990s. Starting with FODA, over PLA, FAST, PuLSE and ending with KobrA different methods are shortly introduced in respect to their support and approach to software product lines. Benefits and cost as well as some severe problems complete this introduction to software product lines.

The proof that software product line approach is not only of academic interest but also of industrial relevance is provided by a growing number of companies where software product lines are successfully introduced and maintained. Participants of the First Product Line Practice Workshop [5] held by the SEI answer the question why they adopt a product line approach in the following ways: “Couldn’t afford to do business old way”, “Had to do something; else we’d be dead” or “Developing/maintaining separate products is path to disaster”. There are various well documented success stories of software product lines, for example the one of CelsiusTech, a Swedish defense contractor supplying shipboard command and control systems to navies around the world (described in [23]). Cummins, one of the leading companies for diesel engines with more than 50 horse powers ([3]) and various contractors of the US American Department of Defense (described in [24] and [25]) prove that large, complex, embedded real-time systems with high safety requirements can be developed following the product line approach.

References

6. Homepage of the Software Engineering Institute (SEI) at the Carnegie Mellon University, Pittsburgh, PA, USA.
17. Homepage of the Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany. URL: http://www.iese.fhg.de
30  Sebastian H. Schenk

   URL: http://www.sei.cmu.edu/pub/documents/02.reports/pdf/02tn018.pdf
   URL: http://www.sei.cmu.edu/pub/documents/01.reports/pdf/01tr030.pdf
Interactive Groupware Systems

Peter Aschenbrenner
Hasso-Plattner-Institut für Softwaresystemtechnik at the University of Potsdam
Peter.Aschenbrenner@student.HPI.Uni-Potsdam.de

Abstract. This paper classifies interactive cooperative systems in the context of CSCW and Groupware and discusses the centralized and replicated architecture variant of such systems with special regard to the requirement of interactivity. For the replicated architecture variant synchronization strategies, like the use of Application Data Units are introduced and depicted on the concrete example of a tool, the Network Text Editor.

1. Introduction

The systems that are discussed in this paper are software systems that allow several users to simultaneously work on the same document. These systems belong to the fields of Groupware and Computer Supported Cooperative Work (CSCW). They are however not the only elements that are related to these domains. CSCW is a field of research that deals with information technology that supports cooperative work. This means that research is done on systems that allow cooperative work and on the effects that such systems have on the way people are working [1, 2]. Because of that CSCW is not restricted to the domain of computer science but also contains elements of sociology, psychology and ergonomics. The gained insights of this domain that are specific to software systems are used in the scope of Groupware. Groupware systems are therefore practical implementations of cooperative systems that incorporate the knowledge of the CSCW domain [1]. Examples for groupware systems are whiteboard applications that allow users to draw geometrical shapes on a virtual board or joint editing applications. Popular groupware systems, e.g. Microsoft Netmeeting, typically consist of several systems that deal with specific purposes like instant messaging and video conferencing.

Groupware systems can be classified into two categories, synchronous and asynchronous systems. Both categories have in common, that they have to deal with multiple users who can interact with the system from various geographic locations. Because of that one task that both categories of Groupware systems have to implement is the bridging of the regional distances of the users. The categories however differ in the way the users are working with the document. Synchronous Groupware systems enable the users to simultaneously work on the same document. In these systems typically sessions are created for a document that is to be worked on. The users then join such a session to manipulate the document. One of the main tasks of these systems is to synchronize the work of the users within a session. Examples for synchronous systems are the already mentioned whiteboard and joint editing...
applications. Asynchronous systems only allow sequential modification of a
document by different users. The main task of these systems is to bridge the gaps in
time that exist during the modification of the individual users. An example for an
asynchronous Groupware system is an email system.

Groupware systems enable the users to interactively modify a document therefore
belong to the synchronous category of Groupware systems. Interactivity in this case
also means that the response time on user inputs is very short. As this document
focuses on interactive systems only synchronous systems are addressed in the
following. Section 2 discusses the centralized and the replicated architecture variant
of Groupware systems, Section 3 shows aspects of synchronization in the replicated
architecture, Section 4 presents details these synchronization aspects on basis of the
Network Text Editor and Section 5 contains a discussion of the presented topics.

2. Architectures of Synchronous Groupware systems

According to [4] two basic architecture variants exist for synchronous Groupware
systems, the centralized and the replicated architecture. Each variant fulfills certain
criteria that lead in two different ways regarding the overall goal of the desired
application. In the following subsections both variants are discussed and evaluated
based on the requirement for a short response time on user inputs.

2.1 Centralized Architecture

In Fig. 1 it is shown the Centralized Architecture variant of synchronous Groupware
systems. The diagram is a Fundamental Modeling Concepts (FMC) Block diagram
[5]. The block diagram depicts the static structure of a software system. The two
different node types that exist are rectangular nodes, which are called agents and
rounded nodes. Agents represent active parts of a system while rounded nodes depict
passive elements. Passive elements typically are channels, to which the small rounded
nodes correspond to and storages that are displayed by bigger rounded nodes.
In the centralized architecture there is only one instance of the groupware application. All users that want to interact with that instance do so via a client. The task of such a client is to direct the input of users to the application instance and display the state of the application. This is why those clients are called GUI-Client in the diagram. The State of the Groupware Application instance also contains the document that is manipulated. This state has to be sent to all clients every time it is changed by the application. The grey rectangular shapes that contain some agents depict possible hardware boundaries.

The advantage of such an architecture variant is that only one agent, the application instance, can modify the state. Avoiding inconsistencies of this state is therefore relatively inexpensive as strategies to avoid inconsistency can be locally implemented in the application. The drawback of the centralized approach however is that for every action the users wants to perform on the state, the input has to be transferred to the application instance, executed there and then the whole state has to be submitted back to all clients. Only then the effect of the user input can be displayed. This is especially problematic as the channel between a client and the application instance can be very slow. This leads to a possible high response time on user inputs of the systems.

Besides these advantages and disadvantages the centralized architecture can also be used to share so called “conference unaware” applications. These applications are standard applications that were not designed to usable by multiple users at the same time. Sharing these applications means that an additional system has to simulate a single user which interacts with the conference unaware application. Such a system is called “cooperation shell” as it can bundle the inputs of many users and therefore allows enables the use of cooperative work on the conference unaware application.
Fig. 2 depicts the possibility to enable several users to work with Word using NetMeeting [6].

![Diagram](image-url)

**Fig. 2.** Sharing “Conference Unaware” Applications using the Centralized Architecture variant

This figure depicts that the *Cooperation Shell* corresponds to the groupware application instance of the centralized architecture variant. The *GUI-Clients* still serve the same purpose. In the concrete example of NetMeeting, these clients are also activated by NetMeeting and display a copy of the screen that contains the *Conference Unaware Application*. NetMeeting only allows one user to work with the underlying application. This active user however can specify another user to become the active user. By doing so he is no longer the active user. Nevertheless the possibility of sharing conference unaware applications does not help in implementing interactive synchronous Groupware systems and therefore another architecture variant has to be evaluated.

### 2.2 Replicated Architecture

In contrary to the centralized variant, the replicated architecture consists of several instances of the Groupware application. Fig. 3 depicts the replicated architecture variant of Groupware systems.
In this variant every user that takes part in a session has its own instance of the Groupware application. Every Instance has its own state. This state consists of a Local State and a Global State. The local state contains state variables the only affect the local instance of the application, e.g. the color of the GUI or the state of the scrollbar in a text editor. The global state represents the state of the overall groupware application and therefore also contains the document that is manipulated. This global state is replicated for every instance of the application, hence the name replicated architecture. It is obvious that this global state has to be consistent for all instances. The global state can be further refined into ADUs, Application Data Units. The semantics of those will be described in section 3.2.

The replicated architectures main advantage is that strategies can be implemented that allow a short response time on user inputs. These strategies that will be discussed in the next chapter require the ability to immediately modify the state of the application without prior sending of messages to other application instances. Another advantage is that the overall Groupware system is not dependant on a single instance, like it was in the centralized architecture. If in this architecture one instance quits the session this does not affect the other instances. The disadvantage of this architecture variant is however the high expense for synchronizing the replicated state. This is especially true as the underlying Communication System can be very slow and many messages can be lost. Despite this disadvantage the replicated architecture can be used to implement the desired interactive Groupware system, but synchronizing of the state has to be considered to grant a well defined behavior.

3. Synchronization in Replicated Architectures

As described above synchronization in replicated architectures has to deal with the problem that each application instance has a replicated copy of the state. A
requirement for the synchronization in interactive Groupware application is that it may not delay the work of the users. A synchronization strategy that deals with that requirement is the Loose Consistency Approach [4]. This approach is presented in the following subsection. The following subsections describe Application Data Units, additional aspects of synchronization and the use of synchronization protocols.

3.1 Loose Consistency Approach

In this strategy, the actions a user takes are immediately executed on the local instance of the replicated state. The user can therefore immediately see the result of his changes as no communication of the application instances is necessary. The resulting inconsistency of the state is intended and has to be resolved. Communication of the application instances is necessary for this task. In a joint editing application for example this approach is necessary as users may not want to wait for every character they type to appear on the screen. The strategy of immediately executing user actions is also called parallel execution variant, as the actions of several users can possibly be executed in parallel. In contrary a serial execution variant also exists. In this variant the inputs of the users are sent to all application instances and have to be confirmed before they are executed. This is called serial because the application instances execute the actions in the same order, which lead to a sequential execution of all actions. Replicated Databases can be used to implement the serial execution variant as the concept of distributed transactions directly concurs in this execution variant. The Loose Consistency Approach however is not possible with Databases.

As the Loose Consistency Approach is suitable for interactive Groupware systems, the resulting inconsistencies of the state have to be dealt with. To do so effectively the changes that were made on a local copy of the replicated state have to done in the same way on all the other copies. By further refining the replicated global state, this can be discussed in more detail.

3.2 Application Data Units

Application Data Units [3] are an application dependant partition of the global state. The effects of this partition are discussed on the basis of a small example. Fig. 4 depicts two different scenarios in which two users change or create graphical objects in a whiteboard application. Each graphical element in these scenarios is a separate ADU.
Fig. 4. Two scenarios of creating and changing graphical elements by the users A and B.

In the upper scenario user A creates the circle and user B creates the rectangle. Both actions are executed immediately. Due to the time it takes for the change messages to travel on the communication system, each user sees different results on his screen. The output on the screen of user B is shown on the left, the one of user A on the right side. The lower scenario also depicts simultaneous actions of A and B. In this case A changes the color of the circle and B add a border to the triangle. The left side displays the output on both users’ screen before the changes. The results of the changes are shown on the right side.

These two scenarios show that ADUs have to be considered when synchronizing change messages in order to resolve an inconsistent state. In the upper scenario the ordering of changes messages is important, as different result occur when applying the changes in different orders. The lower scenario however gives an example in which the order of the change messages is irrelevant. Such change messages therefore do not have to be synchronized.

In general, change messages that affect the creation or deletion of ADUs and messages that deal with the same ADU have to be executed in the same order. The other messages can be executed in any order on the copies of the state. Due to the Loose Consistency Approach that is used, in certain situations actions may have already been executed that should have been delayed until other change messages arrive at a local copy of a state. In this case all changes that are affected have to be undone and executed with the missing actions in the correct order.

A reasonable partition of the global state can therefore aid in minimize the effort that is needed to resolve inconsistencies of the global state. ADUs can also be organized in a hierarchical manner to further optimize the need for synchronization within an application. Besides the use of ADUs for synchronizing in Groupware systems, other aspects of synchronization can be considered to enhance the performance of the overall system.

3.3 Aspects of Synchronization

Minimizing the amount of messages that need to be sent would ease the effort the need for synchronization as fewer messages would have to be ordered in right
manner. A strategy to achieve this goal is to combine several user actions into one user operation. When applying such a user operation on copies of the replicated state, it creates the effect as applying every single user action. The benefit of such user operations are that no change messages have to be sent for every user action but only for every operation. An example for user operations could be found in a joint editing tool, in which change messages are only sent for every line that is completely edited and not for every character that was typed in. The use of user operations however is application specific as it depends on every application what actions can be combined in a reasonable way. As described later on the example of the Network Text Editor in some cases user operations are even avoided, though possible.

Another aspect to consider about synchronization in this domain is the persistent storage of the global state. When a user explicitly wants to save the state, he expects to save a consistent version of the state. Therefore it has to be ensured that all change messages are processed and that until this has been confirmed no other change messages may be created.

Handling of so called Late Joiners is also an issue with session based applications. Late Joiners are instances of the groupware application that join the session later than the others. A mechanism has to exist that enables the Late Joiner to receive the current consistent global state. Therefore all change messages that were created before the entry of the Late Joiner have to be retransmitted. This leads to a high amount of messages to be sent over the communication system. A way to minimize this amount is the use of checkpoints. Checkpoints are persistent storages of the local state at set time intervals. In order to get the latest state Late Joiners have to receive the latest checkpoint and the messages that were sent after this checkpoint. Therefore not all messages have to be sent. The aspects of synchronization that have been discussed are usually handled by synchronization protocols.

3.4 Synchronization Protocols

As described above the aspects of synchronization in interactive Groupware systems are very specific to each different type of application. Therefore the protocols that are used are also application specific. Nonetheless scalable protocols exist [3] that can be used to implement application specific synchronization protocols. One central requirement for these protocols is to support Application Data Units. An example for such a protocol is the Reliable Multicast Framing Protocol.

Common tasks that are handled by the synchronization protocols are the synchronization of the application time and the detection and resolving of inconsistencies. A synchronized application time is necessary as the instances of the application need a mechanism to order the messages that are sent within the system. This is only possible if these messages contain timestamps on which these ordering can be done. An example for the use of synchronization protocols is given on the basis of the Network Text Editor.
4. The Network Text Editor

The Network Text Editor, NTE [3, 7], is a joint editing tool that was developed at the University College London. It is based on the IP Multicast Protocol which allows the sending of messages to many recipients. NTE allows users to simultaneously type in text. Each text is organized in text blocks and lines within a block. If a user starts to type on the document where no block exists a new block is created. The Application Data Units in this case are the lines of text within such a block. This was done because it is very unlikely that several users edit the same line of text within a block. Therefore the need for ordering change messages is greatly reduced as only messages that concern the same text line have to be ordered.

As described above the use of user operations would also fit in this context, as the result of editing a line would be sufficient to send instead of the change for every single character. Due to the characteristics of the underlying communication system however in which a relatively high loss of packets was identified the use of user operations was not implemented. For every character that is typed by the user a change message is that. All such messages contain the complete line and therefore redundancy is introduced. This redundancy is desired because lost change messages can often be compensated by later messages which contain the missing changes.

The mechanism for the synchronization of the application time in the NTE works in the following manner. If an application instance joins a session its application time is not initialized. When such an instance receives a message it sets its application time to the time that is contained in the message. This is possible because each message gets a timestamp of the application time when it is sent. If the application needs to send a message and has not set its time, it sets its application time to the time of the machine it runs on.

To further adjust the application time each application instance checks all timestamps of the messages it receives. If a timestamp shows that another instance’s application time is greater than the own, than it is increased to the value of the timestamp. An application time that is lower than the own time is ignored, as decreasing of the application time is not allowed. Fig. 5 [3] depicts an example in which five instances of the application synchronize their application time according to the described mechanism.
The diagram shows how six instances of the NTE, S1…S6, adjust their Application Time during the processing of messages that occur at certain points in Real Time. S2 sends the first message in the system. S1 and S3 receive this message shortly afterwards and set their application time to that of S2. The short delay that exists due to the time the messages need to travel on the network is omitted here, so S1, S2 and S3 have the same application time. Shortly after this the instances S4 and S5 join the session. S5 sends the next message and sets its own time. All other instances receive the message of S5 and S4, which has no message received yet, sets its time to that of S5. The next message that is sent is either sent by S1, S2 or S3. This message syncs the time of S4 and S5. The last message is sent by the new instance S6. S6 has set its application time to a higher application time than the other instances. The message of S6 results in the increase of the times of all other instances.

This mechanism is sufficient for the other protocols that are within the NTE to detect and resolve inconsistencies. The strategy for inconsistency discovery uses a second timestamp that is contained in every message. This “changestamp” contains the date of the last change on the global state that was executed on the sender’s copy of the state. If an application instance receives a message which contains a changestamp that is later than the latest change of the own copy of the global state, a possible inconsistency is discovered. However this does not ensure that an inconsistency has occurred, as for example the appropriate change message has not yet been received. To further check if and what part of the global state is inconsistent, applications instances exchange checksums of data that allow for exact comparison of
the replicated state. If an inconsistency has been identified, all change messages that are missing have to be retransmitted in order to get a consistent state for every instance.

This comparison of checksums and the retransmission of change messages however are very expensive when considering the amount of messages that have to be sent over the network. An optimization is therefore introduced which tries to avoid this. A so called “current site” is introduced. This site is one instance of the application which periodically sends the latest change messages that it has received. The goal of this redundancy is that instances receive change messages that might have been lost on the network and the need for retransmission occurs less often.

5. Discussion

This paper classified interactive systems that allow several users to simultaneously work on a single document. These systems are characterized as synchronous Groupware systems. To better satisfy the interactivity requirement the replicated architecture variant for those systems supports the Loose Consistency Approach. This approach intentionally ignores the full consistency between all copies of the replicated state of the application instances. Therefore strategies like Application Data Units have to be applied that resolve inconsistencies of the state. These strategies are implemented by synchronization protocols.

Due to the nature of the synchronization strategies, which are very specific for each type of application, no general protocol exists that provides sufficient synchronization. This is why the implementation of synchronization strategies in interactive Groupware systems has to be done for every application. For some applications it might not be possible to find suitable implementations at all.

References

3. Handley, M., Crowcroft, J.: Network Text Editor. USC Information Sciences Institute and University College London
5. Fundamental Modeling Concepts Web Site
   http://fmc.hpi.uni-potsdam.de
6. Microsoft Netmeeting
   http://www.microsoft.com/windows/netmeeting/
7. Network Text Editor at the University College London
   http://www-mice.cs.ucl.ac.uk/multimedia/software/nte
A New Approach to Business Process Support: Case Handling

Jakob Magiera
Hasso-Plattner-Institut für Softwaresystemtechnik
Prof.-Dr.-Helmert-Straße 2-3, 14482 Potsdam
jakob.magiera@hpi.uni-potsdam.de

Abstract. Despite the increasing spread of workflow management systems for business process support, workflow management systems and the workflow approach do not provide the required flexibility needed especially in information-intensive business processes. This document points out the weaknesses, most prominently inflexibility, of traditional workflow management and summarizes how the product-driven approach of case handling as implemented in the FLOWer [4] software can amend them. It also discusses the requirements and limitations of case handling, not promoting it as a panacea for business process support in general.

1 Introduction

In the 1980s, a trend developed that was aimed towards defining and optimizing the processes that could be observed in business organizations. The idea was to seek out those processes and capture them in a process definition model. These models could then become the basis for executing these processes in the organization. The procedure of defining a model and using it for business process execution is called workflow management. Just as industrial automation helped increase quality and throughput in industrial environments, so workflow management was expected to increase quality and throughput in information-intensive businesses. Workflow Management (WFM), as defined by the Workflow Management Coalition (WFMC), is "the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules." [1] This approach, as implemented by most WFM systems, has one central aspect: objects “are passed from one participant to another for action.” This is called routing and is based on an idea that dates back to the days of industrialization: the assembly line. On an assembly line, an unfinished product is transported from work station to work station where specialists perform a particular task on the product. At each station, the product is modified, parts are added or removed, or it is even divided into several components that can be worked on in parallel and later on put together again. This approach is still successfully applied in industrial automation where the stations can be both machines or humans. But also administrative environments can profit from this approach, if processes are characterized by a high degree of uniformity, little customer contact and very few
exceptions. This however, is seldomly the case. Especially in information-processing organizations like service providers, a lot of exceptions occur and may even be the rule. Moreover, the workers involved no longer carry out clearly separated specialist tasks that can be performed isolatedly. Many of them are generalist “knowledge-workers” whose qualifications and responsibilities overlap with those of others. Such people require a complete overview of the whole case being worked on instead of the limited view of a specialist. They need mechanisms that allow for a large number of exceptions to occur without having to explicitly model every possible exception in the process model. It must be possible to allow competent employees to process large parts of a case without having to “walk through” every step given by the process design. Knowledge-workers also need a flexible way to gather information that covers more than what the process design(er) thought necessary for a particular activity. Case handling [2] provides the mechanisms that support this approach.

Chapter 2 elaborates on the introduced weaknesses of WFM and point out why it does not provide the flexibility that is needed in information-intensive processes. Chapter 3 introduces the case handling approach as implemented by the FLOWer software and how it solves the problems of WFM. Chapter 4 summarizes the findings and discusses the requirements and limitations of case handling.

2 The WFM Approach

Typical WFM process definitions can be modelled as a sequence of activities with input and output data being passed from one activity to the next. Activities are associated with roles and can be carried out by all persons that have this role.

Fig. 1. Workflow Process Model

Fig. 1 shows how WFM is based on the assembly line approach. Each activity is a “station” where a specialist (a person with the associated role) works on a task. He
can receive input data from previous activities and possibly produces output data for following activities. The split after activity 2 means that activity 3 and 4 can be carried out in parallel.

2.1 Context Tunneling

The process model is executed using dynamic role resolution. Each user receives one or more roles and a so-called work tray. In a person’s work tray those activities appear that are due to be processed and match one of his roles. Only data from immediately preceding activities is available. If a user does not have a role of preceding or subsequent activities, he has no idea where the data came from or where it is going (activity sequence 2-3-5 in Fig. 1). Moreover, even if he has the roles needed to carry out several activities in succession, dynamic role resolution can assign consecutive activities to different users. This means that not only the data context but also the activity context is very limited. In short, one has no overview of the whole case. This is called context tunneling. A mechanism that allows a user to see the data of other preceding activities that relate to his work tray can significantly enhance the quality and speed of processing. Consider a medical process where one activity is to prescribe a medication to the patient. Imagine that a preceding activity included entering allergies of the patient. If this information is not available at the activity “select medication” a wrong decision could have serious consequences. In another process, missing context information might cause considerable delay because external documents have to be manually searched.

2.2 Domination of Control-Flow

WFM that is based on the assembly line paradigm is dominated by routing. Nothing can happen outside the definition of the activity sequences. Typically, if a subsequent activity has the same role as the current one, it will be assigned to the same person who carried out the current activity (activity sequence 2-4 in Fig. 1). In reality this means that activity after activity is “pushed” into a user’s work tray. This push-oriented perspective leaves users powerless and inflexible. Powerless in the sense that they cannot influence what goes into their work tray and inflexible in the sense that they are tied to the sequence (one activity after another) defined in the process model. Consequently, mechanisms are needed that provide the user with more ability to control what goes into their work tray and a process model definition that gives more freedom to the user in carrying out activities. One must keep in mind that in WFM exceptions must be modelled before the process is executed, i. e. at design time. This means that one must know beforehand which activities might need to be redone or skipped and include every possibility in the process model. This places a considerable burden on the process designers. If unexpected situations occur during process execution, one either has to go back and change the process model or there must be mechanisms that bypass the process design somehow. The first solution risks that the current process state is incompatible with the changed process model and
many steps might have to be redone. If one has to bypass the process model and thus go behind the WFM system’s back very often it makes it more of a liability than an asset.

3 Case Handling

Case handling does not use the assembly line paradigm as its basis. It does not view the process as a series of isolated activities but instead provides the means to see the process (or case) as a whole. The sections in this chapter describe how the key concepts of case handling, context provision, flexible execution, and a data-driven process execution, help to support this view.

3.1 Context Provision

The most important idea the term case handling implies is the availability of context and thus avoiding of context tunneling. It suggests that a user always has the whole case, e.g. a motor claim, in mind instead of only an activity like “fill out opposite party data”. In order to think that way, the user must be provided with data that relates to more than just a singular activity. Fig. 2 is an Entity-Relationship-Diagram according to the FMC notation [5] that illustrates how data is bound to the sequence of activities in WFM.

![Fig. 2. Data context in WFM](image)

The precedence relation between activities represents the order of activities that is modelled in the process design. One activity produces data output for its subsequent
Each user receives one or more so-called work profiles from which his work tray can be filled. It consists of an authorization profile and a queue template. The authorization profile consists of roles. This collection of roles serves to restrict which activities a user can carry out. The queue template is a collection of data objects that can be used for filters. When the data objects specified in the queue template are given data values, a queue filter is defined (is_used_for-relation). This filter selects all cases and activities that match the defined filter, thus filling the work tray. Additionally, a user can create generic queries with any data value defined in the process definition. Such queries could be used for a service hotline to quickly gather all cases that relate to a particular customer, but also to gather statistical data. One could, for instance, create a query that selected all cases that have a value of more than € 100,000 and are handled in the North region. These mechanisms demonstrate the shift from a push-oriented to a pull-oriented approach. With case handling, the user has much more freedom to select which cases he wants to work on currently.

<table>
<thead>
<tr>
<th>queue template</th>
<th>queue filter</th>
</tr>
</thead>
<tbody>
<tr>
<td>region</td>
<td>“North”</td>
</tr>
<tr>
<td>order amount</td>
<td>“&gt;1000”</td>
</tr>
<tr>
<td>customer name</td>
<td>“* Smith”</td>
</tr>
<tr>
<td>product ID</td>
<td>“1403”</td>
</tr>
</tbody>
</table>

Table 1. Queue Template Example

As an example, imagine a work profile that contained the roles “Regional Manager”, “UNIX Admin”, and “Accountant”. A queue template could contain the data objects “region”, “order amount”, “customer name”, and “product ID”. A user
activity which receives it as input. These are the data objects that are tied to this precedence relation. Many data objects can be associated with one precedence, but data is only available in the context of two adjacent activities. Case handling takes a much less limiting approach. Data is seen within process context, i.e. a data object can potentially be seen at every step within a case. This is shown by the n:m relation in Fig. 3 that specifies which data objects are relevant for an activity. How this relation is determined depends on the case handling system. FLOWer provides filters that allow a user to gather context information on a very fine-grained level.

As mentioned in section 2.1, WFM not only limits data context but also activity context. Only the roles determine what a user can carry out and see in his work tray. Case handling with FLOWer has powerful mechanisms that give a user a better overview over the whole case. Fig. 4 shows how the content of a work tray is determined using FLOWer.
with this profile could then receive a work tray based on a queue filter that attached the values “North”, “>1000”, “* Smith”, and “1403” to the queue template. This would mean he would receive all cases of the North region that had more than 1000 products of the product 1403 ordered by a customer with the last name Smith. His role as regional manager might allow him to confirm a bill due to be sent for these orders, while his role as accountant could allow him to enter the bill in the businesses’ accounting software.

3.3 Flexible Execution

As mentioned in section 2.2 the domination of control-flow in WFM leads to a very rigid process model that makes exceptions very difficult to handle. In fact, any kind of exceptions have to be put into the process model at design time. Fig. 5 shows a simple process model that allows for a very limited flexibility.

![Fig. 5. Skipping and Redoing in WFM](image)

The graph shows an activity sequence (1, 2, 3, 4) and two additional control-flow paths (3,1) and (2,4). The path (3,1) means that it is possible to redo activity 1, 2 and 3. In addition, activity 3 can be skipped. If one arrives at activity 3 and an exception occurs that makes it necessary to redo activity 3 only, this process model forces one to redo activities 1, 2 and 3. What is more, since activity 2 can only be carried out by a person with role B, a person with role A has to wait for the person with role B until they can finally redo activity 3. Things would get even worse if all persons with role B were unavailable. A WFM system administrator might have to be consulted to override the process model in order to go past activity 2. This shows how a rigid process model can make process execution extremely inflexible and time-consuming.

Together with role hierarchies, case handling with FLOWer provides a very flexible process design. Fig. 6 illustrates what a simple role hierarchy could look like.
Role A is higher than roles B and C, which means that whatever role B or C are allowed to do, role A can also do. Roles B and C, in turn, are not allowed to do what role A can do. Fig. 7 shows what a case handling process model can look like that allows roles not only to carry out (execute) but also to skip or redo an activity. Consider a person that has role A in the role hierarchy above. With this process model, this person could carry out activities 1, 2, 3, and 4 sequentially. If activities 1 and 2 became obsolete for some reason, the skip permissions at these activities would allow him to start with activity 3 also. Moreover, he could redo activities 2, 3 and 4 if necessary.

All possibilities this process model holds can be seen in Fig. 8. This also illustrates how easily a process model can become complicated if many exceptions have to be incorporated. One must bear in mind that this is a very simple example, and that actual business processes are much more complex. It is obvious that such process models can quickly become unmanageable.
3.2 Data-driven Control-Flow

In traditional WFM, control-flow determines the progress of the process. This was previously described as the domination of control-flow. Case handling takes a radically new approach to how a case is processed. The value of data objects can now determine the progress of a process. Fig. 9 depicts an excerpt from the meta model given in [3] transformed to FMC notation. Again, we see the difference between the WFM approach in Fig. 2.

A data object is not tied to the precedence relation between activities but can be related to activities in three ways (only two depicted). The first type of relation is “restricted”. Data objects that are restricted to an activity are similar to the way it is WFM, i.e., the data is necessary for an activity to be completed and it can only be entered if the activity is the next due to be processed. Mandatory data objects are also necessary for an activity to be completed but can also be entered if the activity is not
The next due to be processed. All other data objects are “free” (not depicted), i.e. they can be entered at any point in the process.

The following rule determines when an activity is completed. An activity instance is completed if and only if

- all previous activities have been completed (or skipped),
- all mandatory/restricted data objects have a value,
- and the so-called completion condition of an activity is true.

Since the completion condition is mostly set to “true”, the completion of an activity actually only depends on the values of data objects. FLOWer uses so-called forms to allow the user to enter data. The cardinalities in the relation between activities and forms imply that many activities can be handled by one form. Depending on the role a user has, it is therefore possible to complete many activities using just one form. Since forms can also exist independently of activities (0:1 relation), data can also be entered without carrying out any activities. This adds to the provision of case context and the detachment from control-flow discussed above.

4 Conclusion

Chapters 2 and 3 have demonstrated how traditional WFM falls short of the needs for flexibility and context knowledge needed in information-intensive processes. With the concepts of context provision, data-driven control-flow and flexible execution, case handling provides the mechanisms to support such processes adequately without forcing knowledge-workers into a straight-jacket defined by the process model. In the author’s opinion, it is a successful attempt to combine data-centered approaches like Groupware and the control-oriented workflow approach.

But even case handling does not provide a panacea for business process support. The flexibility provided by case handling may not be necessary in all areas, and where processes are characterized by standard specialist tasks that contain few or well-defined exceptions, the traditional WFM approach is more appropriate. In such environments, processes are characterized by separate well-defined activities that can be worked on without necessitating case context. The assembly line paradigm is appropriate here. One must also bear in mind that the flexibility of case handling comes with a price: more time and resources must go into the design of the process, the roles and the filters. Since information is so readily available to the users, it is also essential that they are competent and knowledgeable, i.e. they want the case overview that case handling provides. Otherwise, information overkill may be the result. As always, with freedom comes responsibility. Applied to the wrong domain or given into less capable hands, case handling could become more of a curse than a blessing. The author suggests that a thorough analysis of the types of processes occurring in the organization and the people working on them be made before making the decision for
case handling. In the right hands, it can be a powerful method to support business processes and bring them to a new level of quality and throughput.

References


Using AI Planning Algorithms To Support Business Process Modeling

Harald Meyer

Chair of Business Process Technology
Hasso Plattner Institute for Software Systems Engineering
at University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
D-14480 Potsdam, Germany
hmeyer@cs.uni-potsdam.de

Abstract. The purpose of this paper is to elaborate the using of well known planning algorithms from the AI domain for modeling business processes. The development of business process models is very labor-intensive. Planning algorithms can reduce the work amount by automating the ordering of the activities in the process. The work, that has to be done, will be limited to the specification of initial and goal states and process activities with their preconditions and effects. The main focus will lie on the selection of an appropriate algorithm.

1 Introduction

Business process modeling is very important. It leads to a better understanding of and an improved communication about the processes in a business. A model of the processes is the basis for improving the businesses working by changing its processes. This is known as business process reengineering (for example [21]). Besides modeling existing processes, creating models of new processes before their deployment is equally important. The processes can be tested and simulated and errors and inefficiencies can be found very early. The problem with business process modeling is that it is very labor-intensive. This paper demonstrates one possible automation for business process modeling using AI planning algorithms.

Business process modeling involves the observation of real processes in a business. This can be very difficult because a good level of abstraction has to be found. If it is too abstract the people executing the processes do not know what they have to do. If it is on the other hand too specific, it is hard to overview the complete process. Besides for people executing the process their labor will be very boring and inflexible because they are told what exactly they have to do. If a good level of abstraction is found, the activities, which are atomic for this level, have to be found and differentiated against each other. This is difficult because there may be no exact boundaries between the activities.

Beside the difficulty of identifying atomic activities, business process modeling additionally consists of their correct ordering. This can be very labor-intensive, too. But in contrast to the identification of activities it can be automated. Instead of devel-
opposing new algorithms for this task it seems obvious to reuse algorithms from a well-
known domain – AI planning. Those algorithms are for example used to control ro-
bots, which have to execute certain actions with preconditions and effects. The com-
plete ordering of the actions is not important. Instead a partial order of the actions for 
a given initial state, a goal and a set of possible actions is imposed. Except for the 
different terms action and activity, this situation seems similar to the one we have in 
business process modeling. If we can reuse AI planning algorithms, business process 
modeling can be reduced the identification of activities, the determination of an initia-
lar state and the selection of a goal. The main focus of the rest of this paper will be on 
finding an appropriate algorithm with the needed features.

In the next section related works are presented and the differences to this approach 
are elaborated. Then in section 3 AI planning and business process modeling are in-
troduced in general. Section 4 then demonstrates how planning algorithms can be 
used for business process modeling. It starts with the determination of the essential 
features of a planning algorithm for business process modeling. Afterwards the best 
algorithms are selected and a business process example is introduced and imple-
mented. The paper ends in section 5 with a conclusion which summarizes the work 
that has been done and assess it and gives an outlook over possible future work.

2 Related Work

Adapting AI techniques for various business process technologies like modeling or 
workflow management is not completely new. For example several knowledge-based 
approaches to workflow management exist. Although most research in this area is still 
done at an academic-level (for example the MILOS project[9] at the University of 
Kaiseraltern, researches at the University of Edinburgh [18,19]) some steps for using 
AI techniques in commercial workflow management systems have been done. One 
example is a project at the BT Research Laboratories which tries to include AI plan-
ing in a legacy workflow management system[26]. Another interesting project is the 
SHAMASH tool which is developed in a research project[1] with the same name. It is 
a tool for business process reengineering which uses AI techniques for the representa-
tion of knowledge and optimization of process models.

The main distinctions of these works from to the work that is presented in this pa-
ter, is that they are mainly focused on improving the automated execution of business 
processes inside a workflow management system. And they are using various AI 
techniques like scheduling, knowledge representation and planning whereas I am 
focusing on AI planning.

While the second distinction is less important, the first one shows a completely dif-
frent approach. We want to model new or existing processes to get a better under-
standing about the processes, to improve and verify them. Automated execution is not 
the main goal but a benefit one can get after the processes are modeled. So the plan-
ing algorithm is not a part of the workflow management system, but will be used in 
an earlier stage of the modeling process. Fig. 6 will show how the modeling process 
should look like.
3 Preliminaries

Before planning algorithms can be used to support business process modeling, planning and business process modeling have to be introduced. This will be done in this section. First planning is introduced together with a non-business process example. Then a short introduction into business process modeling is given. Its purpose is to introduce some of the terms I am using and to support the motivation. It is not an introduction into business process modeling in general. If an introduction into business process modeling and workflow management is needed, [17] and [21] are a good starting point.

3.1 Introduction To Planning Algorithms

The shortest definition of planning is that it is problem solving. A planning algorithm takes a planning problem which consists of an initial state, one or more goals and a set of possible actions with their preconditions and effects (also called domain theory) and tries to determine a (partial) ordering of actions which leads from the initial state to one of the goals. This ordering is then called the plan. If several possible plans exist planning algorithms try to find the optimal one concerning the length of the plan.

3.1.1 Planning Example

![Blocksworld problem](image)

Fig. 1. Blocksworld problem.

To simplify the demonstration of features of planning algorithms an example is introduced here and used throughout this paper. It is the classical planning problem called blocksworld. It consists of a number of blocks and a table. The problem is to get from one configuration of the blocks to another one. One instance of blocksworld can be seen in Fig. 1.

As we have already learned a planning problem consists of a description of the initial state, the description of the goal and a domain theory. Let us identify those elements in this blocksworld problem. The initial state is the configuration of blocks on the left side. Using first-order calculus it may be described as: on(A,B), on(B, Table), on(c, Table). The goal is just a different configuration of the blocks: on(A, Table), on(B, A), on(C, B). Finally the domain theory consists of just one element. The only possible action is to move one block from one position to another position. The pre-
condition for this action is that no other block is on top of the block, that should be moved, and that the new position is not occupied. A position is occupied if it is a position on a block and there is already another block on this block. For example it is not possible to move block C onto block B in the initial condition. The effect of the move action is that the block is no longer at the old position but at the new one.

![Diagram of the blocksworld problem](image)

**Fig. 2.** Solution for the blocksworld problem from Fig. 1.

One solution is shown in Fig. 2. First block A is moved onto the table. Then block B is moved on top of A and finally block C is moved on top of B. Beside this solution, which is optimal concerning the plan length, an infinite amount of other solutions exists.

### 3.1.2 What Makes Planning Complex?

Looking at above example planning seems to be pretty simple. But in reality it is very complex. Even with only one action there is already an infinite amount of possible solutions. Of course only very few of them make sense\(^1\) and even fewer are optimal.

But if you have got a complex planning problem with tenths of entities and actions, finding an optimal solution by hand is nearly impossible. Even for the computer it is not easy. As researches have shown general planning with first-order is semi-decidable \([7,10]\). This means that for a given planning problem it is possible to prove that a solution exists. But proving that no solution exists is not possible. By limiting language for describing planning problems decidability can be reached.

The most simple limitation is using propositional calculus instead of first-order calculus. Tom Bylander showed in \([4,5]\) that propositional planning is decidable if only a STRIPS-like language is used. This is basically a propositional language with

---

\(^1\) Move(C, A) and Move(C, Table) in the initial state does not lead to the goal
only ground formulas\(^2\). The initial state is a finite set of ground atomic formulas; the goal state and the preconditions and effects of the actions are ground literals.

Limiting the planning problem language to propositional calculus is not practical as it is very labor-intensive and sometimes impossible to specify real-world problems using only propositional calculus. Fortunately first-order STRIPS planning can be made decidable, too. As shown in [6, 10] it is decidable if no functions are allowed\(^3\). This is called datalog STRIPS planning. A detailed discussion of propositional and first-order calculus will be given when the features of planning algorithms are assessed in section 4.2.

Besides decidability, it is also interesting to know how long it takes to calculate a solution. As one may have guessed even decidable planning is very complex. [4,5,6,10] show that even simple propositional STRIPS planning is in the complexity class PSPACE, which means that we can not make statements about the run-time of a planning algorithm. We can just say that the space the algorithm needs to solve a problem grows only in a polynomial way: If we have two planning problems p and q and we know that q is twice as complex as p, we can be sure that space(p) = n \(\times\) space(q). This is a very important statement, because if we could not say that the algorithm uses only polynomial space, it would be practically unsolvable: For the same planning problems p and q and the space function space(x) = 10\(^4\), which returns the number of bytes used by the algorithm, p has the complexity 10\(^6\), q has the complexity 20. While p is solvable using nearly 10 gigabyte of space, q needs more than 90 million terabyte and is practically unsolvable.

While we do not have this problem for space, we still have it for time used by the algorithm. Ideally we like to have an algorithm which is in P. This would mean that we could solve the problem using polynomial time. Sadly this not feasible without severe restrictions on the expressiveness of the planning language[5]. Even membership in NP, is only reachable with some restrictions: If one does not search for any plan, but only for plans smaller than a given plan length, planning is solvable in non-polynomial time [5,10]. As we will later see this is used by some algorithms.

3.1.3 Planning Algorithm Types
The previous section should have made clear that planning can be very time-consuming. But fortunately the complexity class only says something about the worst-case. For the average-case run-time may be much lower. The purpose of a good algorithm is to limit the run-time for the average-case and ensure that only very few planning problems are worst-case scenarios.

It is important that good algorithms are used for planning, because otherwise no results would be delivered in an acceptable time frame. Therefore a large amount of different types of algorithms has been developed. A coarse classification distinguishes just between general and planning-specific algorithm types. The first class contains algorithms which are adapted from different fields of computer science. They are either adjusted for planning or planning problems are translated into problems which

\(^2\) No variables
\(^3\) Simplified results of [10]. Under certain restrictions even planning with functions is decidable
\(^4\) This number has no meaning, it was just chosen for demonstration.
can be solved by them. The second class consists of algorithms which were developed and exactly tailored for planning problems.

Historically the first algorithms belonged to the second class. Algorithms like STRIPS[11] or C-Buridan[16] were not using very elaborate techniques to reduce run-time. So techniques from other fields were adopted to improve the yet immature planning domain. For example very fast algorithms for solving satisfiability problems for propositional logic exist. Some planning algorithms use them by transforming a planning problem into a satisfiability problem. Recently planning-specific algorithm types have caught up. They use for example the fact, that creating a completely new plan is sometimes more complex than adjusting a given plan.

3.1.3.1 Graph-based Planner

Fig. 3. Graph built by the planner.

Graph-based planner work by successively building a directed graph. Fig. 3 shows how such a graph will look like. It consists of two different node types: Round ones represent propositions and square ones actions. An edge between a proposition and an action means that this proposition is a precondition for the action. An edge leading from an action to a proposition means that the proposition is an effect of the action. The grey edges between propositions are maintenance actions which represent the inertia in the graph: If a proposition is not explicitly changed it stays the same.

A graph-based planner consists of two phases: First the graph is expanded adding two new levels (one proposition and one action level) and then it is tried to extract a solution. A solution can be extracted if everything that has to be true in a goal state is true in the proposition level without being threatened by another proposition or action.

5 Feature-wise Buridan is pretty modern, but it has very bad run-time performance (see [2] p. 8)
3.1.3.2 SAT-based Planner

Solving satisfiability problems for propositional formula can be done very quick if the right algorithms are used. As Henry Kautz and Bart Selman demonstrate in [14] looking at planning as satisfiability instead of following a deductive approach makes sense. Although satisfiability is in theory at least equally complex as traditional deductive approaches, it is much faster in reality because very fast algorithms like GSAT[27], exist.

The problem with SAT-based planners is that encoding a planning problem into a satisfiability problem is harder than the normal approach of describing initial and goal state and the domain theory. Therefore they use a compiler which takes a planning problem and encodes it into a propositional formula. Then a solver tries to satisfy it and finally a decoder decodes the satisfying variable assignment into a plan. Fig. 4 shows how a complete SAT-based planner looks like. Besides the compiler, solver and decoder an additional element, the simplifier exists. It takes a propositional formula in conjunctive normal form and simplifies it.

Another interesting point about the figure is, that the solver does not always output a satisfying assignment. If it does not output one, it does not mean that no plan exists: The compiler created a propositional formula which represents the problem “Find a plan of length n”. As we have learned solving this problem is easier than “Find a plan” and leads to a simpler satisfiability problem. Therefore if no satisfying assignment is found the compiler gets the chance to generate a new formula for an increased plan length.

3.1.3.3 Planning as Heuristic Search

Another approach is to see planning as search. Instead of transforming the planning problem into another problem or deducing possible next states, the state space is simply searched. Of course the problem is that the search space is very large. A way to prune the search space has to be found. A good approach is the use of heuristic functions. They do not only find solutions but are very quick.

Maybe the most famous heuristic function is A*. It is used to find the shortest path in a graph. Take for example the graph out of Fig. 5. A path is searched from A to E. Instead of trying every possible path, a heuristic function f(n) is used to determine the shortest path. It consists of the function h(n) which is an estimation of the length of the path from the starting point and a function g(n) which represents the length of path from the starting node to the actual node. f(A) may be f(A) = g(A) + h(A) = 0 + 5 = 0. Then the two possible following nodes are inspected. f(B) may be f(B) = 5 + 4 = 9 and f(C) may be f(C) = 2 + 3 = 5. Because f(C) is smaller it is selected. This is repeated until the goal is found. If at any point f(n) > 9, the path over B is used. If h(n)
is an underestimate, e.g. there is no path from this node to the goal which is shorter than the estimate, A* is guaranteed to find an optimal solution.

![Fig. 5. Graph with five nodes.](image)

A* can not be used here as a heuristic function here. Fast-Forward uses a simplified version of a graph-based planner as a heuristic function. It is much faster as a complete graph-based planner, but it finds only possible solutions by pruning the search space. Fast-Forward must then use other mechanisms for finding real and optimal solutions.

3.1.3.4 Other Types of Algorithms

Besides the three algorithm types described in detail above, several different types exist. Four of the more interesting ones will be depicted now. Like graph- and SAT-based planner OBDD-based reuse a technique from another field of computer science. They use ordered binary decision diagrams to encode planning problems. This allows to implement a very fast planner. Ordered binary decision diagrams are a way to represent Boolean functions. They can be used very well for model checking where behavior of a system is modeled using a finite state automaton. A similar approach can be performed for planning problems[20].

Plan reuse by adjusting a given plan is also used in some algorithms. These are called case-based planner. Successful previous plans are stored and when similar problems arrive in the future they are reused. But as [22, 23] show this does not improve performance per se. Actually modifying a given plan is for worst-case scenarios at least as complex as creating a new plan.

A completely different kind of algorithms are the ones that are based on hierarchical task networks (HTN). They see the goal as a network of tasks. This network is then transformed by substituting non-primitive task into primitive ones until only primitive tasks exist. These primitive tasks are mapped to actions. So during execution a very sparse network is transformed into a dense one. One implementation of HTN-planning is O-Plan[8].

The final planning algorithm type that I will describe is on the boundary between planning and scheduling. The problem with traditional planners is that they only allow situations which are described in the generated plan. If an unknown situation occurs, a completely new plan, based on the new situation, has to be generated. A reactive plan-
planner does not have a complete plan in advance, but adjusts its plan during run-
time. So replanning is done during every step.

3.2 Business Process Modeling (BPM)

Basically two situations exist in which one wants to model business processes. Either existing processes in a business should be modeled or new processes should be mod-
eled before their deployment.

The benefit of having a model is that the processes can be seen in their entirety. They make understanding the whole process and communicating about it possible. This is the basis for process improvement. Processes can be simulated, verified and if needed, changed. Additionally the processes can be managed. They can for example be stored in a database and new processes can be constructed out of existing. Finally processes can be (partially) automated. They can be executed in a workflow manage-
ment system, where some parts of them are executed by specific applications and oth-
ers are executed by humans who are guided by the system.

Business process modeling is a very complex task. As we have already seen, it consists of an observation and an optional process improvement part. Of course be-
tween those lies the modeling part, where the observations are transformed into an initial model.

The observation leads to a set of activities with preconditions and effects. Finding preconditions and effects is difficult as they can be very subtle. Often the decomposi-
tion of a process into activities is equivocal, because the decision whether an activity is atomic or not, depends on the right observation of preconditions and effects. One example is the processing of an order in a call-center. If the activity of recording the personal information of the customers is not observed as a single activity, but is seen as a part of the completion of the order, many optimizations of the process are no longer possible: It may be wise to record the customers information before doing any-
thing else. The call-center agent can see useful information about the customer and if the process is interrupted the call-center agent may recall the customer. It should be obvi-
ous that automating the observation is nearly impossible. Most of the activities are done by humans and the information that is created, modified and exchanged be-
tween activities is normally not specified. If a call-center agent does not know the answer to a customer’s question, he may just ask a colleague. This conversation is never recorded nor does it follow any protocol. Observation has to be an active proc-
ессі where the system ask the persons, that participate in a process, certain question. Such a system would be very complex and would need domain knowledge to work correctly. It would be too expensive to give automated observation an advantage.

For process improvement the information from the observation is taken. A more efficient ordering of the activities is created. This is something that is - especially if an optimal solution is wanted - very difficult to do for humans. It is similar to plan-
ning problems which are semi-decidable. But we can easily see that it is even more difficult for humans. If you have three possible activities a, b and c and you only want to test the plans with the length 3, you already have 27 different plans (aaa, aab, …). For problems with more activities and for longer plans this number rises very fast.

---

6 This counts for human observers, too.
Actually you have \( a^n \) possible plans, when \( a \) is the number of possible activities and \( n \) is the exact plan length. Of course preconditions and effects of activities limit the amount of plans. But testing for them makes the problem even more difficult.

We have seen now, that some parts (process improvement) of business process modeling are difficult for humans while they are relatively easy for computers. For other parts (process observation) it is the other way around. Instead of automating business process modeling completely or not, it seems to make sense to automate the parts that are difficult for humans.

### 4 BPM Using Planning Algorithms

![Phases of the modeling process.](image)

We have seen now, that it makes sense to automate the ordering of activities during business process modeling. Fig. 6 shows how the process of business process modeling should look like after parts of it are automated. In the beginning either an existing process is observed or a new one is created. Then the results of the observation or creation are modeled. This modeling does no longer involve the creation of a complete business process model, but is the description of the process as a planning problem with an initial and goal state and a domain theory. The planning problem is then transformed by the planning algorithm into a plan which is equivalent to the process model. It is then verified by humans and then either executed in a workflow management system or brought back into the normal process execution in the business.

#### 4.2 Features Of Planning Algorithms

A large amount of different planning algorithms exist. The database of PLANET\(^7\) [25] lists round about 40 different algorithms. They do not only differ in the type of algorithm they realize, but also in the features they provide. To select one algorithm a feature-based selection process has to be established. Therefore the features that characterize planning algorithms are explained in the following.

\(^7\) European Network of Excellence in AI Planning.
4.2.1 Standardized Language

Actually there are two languages involved in a planning algorithm. You have the input language which describes your planning problem and you have the output language in which the generated plan is formulated. Having standardized languages for both of them is important: If your input language is standardized you can easily use your planning problem with different algorithms and compare their results. If you have got a standardized output language you can easily use your generated plan in applications or you can write a converter which translates it into the input language for a workflow management system or into an UML-diagram using XMI.

While a standardized output language does not exist, a standardized language for the description of planning problems called PDDL\[24\] exists. It was initially developed for the AIPS-98 planning contest for the comparison of performance and capabilities of different algorithms. Most modern algorithms support it. Its main advantage is that it is very flexible. At the beginning of the description of a planning problem the capabilities needed by an algorithm to understand the problem are stated. So PDDL can be used for algorithms ranging from very simple STRIPS planners to complex planners using uncertainty and sensing.

4.2.2 Expressiveness of the Used Logic

Limiting the planning problem language to propositional calculus is not practical as it is very labor-intensive to specify real-world problems using only propositional calculus. If you want to describe for example the type of hair every dog has, you have to describe it for every instance:

lassie → longHair.
struppi → shortHair.

For such an simple example it is still pretty easy. But if you have more than one property and different types of objects (dogs, cats, …) and you have complex queries like: “Give me all the dogs that have long hair and are 50cm tall” propositional calculus becomes not only inappropriate but unusable, because even if can formulate such a complex query\(^8\) it could not return all the dogs matching this query. It would just answer yes or no.

With first-order calculus you have got predicates, which define relations between objects, functions and an two quantors: all-quantor and existence-quantor. While the all-quantor defines that something has be true for all objects, the existence-quantor says that at least one object with the given property has to exist. Defining the dog problem in first-order calculus is much simpler:

\(^8\) Which would look like: \(?\text{dog} ∧ \text{longHair} ∧ \text{height50cm} \).
dog(lassie).
dog(struppi).
hair(lassie,long).
hair(struppi,short).
height(lassie,50).
height(struppi,30).
\(?\, \text{dog}(X), \text{hair}(X, \text{long}), \text{height}(X,50)\)
lassie

By the way: The blocksworld problem above was also defined using first-order calculus. If you want to, you can try to model it using propositional calculus. It should be possible but you would need much more actions than one. For every possible start and goal position and every block you would need an own action.

4.2.3 Conditional Effects
Normally the effects of an action occur every time the action is executed. Sometimes one wants to have actions whose effect only occur if certain conditions hold. Take for example a modified version of blocksworld: Instead of having just one hand which can only move one block at a time, we have a second hand and we can hold blocks in both hands. The simple move action is then decomposed into the actions grab and put. The action “grab” grabs one block with one hand if no other block is in it. The block is then in the hand, with which it was gripped. The action “put” puts a block which is in one hand onto another block or the table.

Without conditional effects four actions have to be described: Grab and put for each hand. Having conditional effects only two action descriptions are needed. The precondition for the grab action is that at least one hand is free. The effect is that, if the right hand is free the block is in the right hand. Otherwise it is in the left hand. Putting a block down can then be done if the block is in one of the hands. The effect is conditional, too: The hand in which the block was is free.

This example demonstrated that conditional effects do not increase the expressiveness of the language; They just simplify the description of the planning problem. In particular conditional effects do not allow uncertainty. The effects of an action are still predetermined by the state in which the action was executed.

4.2.4 Uncertainty, Sensing and Probabilistics
Uncertainty is a way to overcome the limitation of complete predetermination. Using uncertainty it is possible to describe an initial state which is only partly known and actions whose effects do not happen every time the action is executed.

An uncertain initial state for the blocksworld example maybe, that block A is either on block B or on the table. Uncertain effects may be used to describe a move action which is not always successful. If a block is put on another block, it may fall down if it was not released properly.

Two possible solutions to resolve uncertainty exist. The first one is sensing. Using it the real initial state and the effects of actions can be determined during run-time. In the blocksworld domain one could check whether a move action was successful or not by sensing the state after the action was executed. Sensing can also be used to model actions which involve exogenous events. A business process example is credit rating. It may involve an activity which performs the check of an external database. The out-
come of this activity can not be known in advance. This activity is mapped to an action whose outcome is uncertain. But as its effect is needed, sensing is performed to determine the correct value.

The second solution to resolve uncertainty is, to not only describe that either that or that effect occurs, but to also assign probabilities to all possible effects. No sensing is then performed, but a plan is generated for which the probability to reach to goal state is as high as needed. In the blocksworld domain one would describe, that the effect of moving a block onto another block is that it is there in 90% of the cases. In the other 10% it is on the table. A plan may be one in which it is tried to put as few blocks on other blocks as possible to reach the goal.

Probabilistics can be used for easy process improvement. It just involves the observation of the probabilities of the outcomes of actions. If for example initially an activity benefits the reaching of the goal in 90% of the cases and in a later observation only in 20%, it may be no longer part of the generated plan. While this seems to be a nice feature, it is not really needed and generates more work. On the other hand uncertainty and sensing are essential features for business process modeling. If everything is known in advance, no process is needed. One would just need a activity with sub-activities which are executed one after another.

4.2.5 Arithmetic Functions and Metrics
The support of arithmetic functions should be standard. It should be possible in every planner to work with numbers and add or multiply them. Sadly only very few algorithms support it. Normally if an algorithm supports arithmetic functions it does support metrics, too. Metrics[13] are a way to define optimization criteria for the generated plan. While the plan normally is optimized concerning plan length, it may make sense to define another function (e.g. a cost function) whose value should be as low or high as possible.

A cost function could be used by assigning costs to each possible activity. When the goal is specified it is also said that the cost function should be as low as possible. Instead of searching for the shortest plan, a plan is searched which generates the lowest costs. Both arithmetic functions and metrics are needed to do business process modeling. But they are not essential because many business processes can be modeled without using them.

4.2.6 Temporal Planning
One of the abstractions of planning from the real world is atomic time. Each action takes as long as each other. With temporal planning actions have a duration and can overlap. As temporal planning is not important for business process modeling, because it uses atomic time of activities, I will not describe temporal planning in detail. An example for an implementation can be seen in [28].

4.2.7 Multi-Agent Planning
I will not describe multi-agent planning in detail, too, because agents or actors are not important during business process modeling. Of course roles exist and certain activities can only be executed by persons with certain roles. But the mapping of roles to actors is not done during business process modeling. While this would have the ad-
vantage, that parallelism and synchronization could be defined in the process model, it would involve using the complete organizational structure. This is too complex for human business process modelers. It is also too complex for planning algorithms, because it would add another dimension with hundreds of possible actors for each action. Multi-agent planners are only designed for round about ten agents. Such a low number is not useful for business process modeling. Therefore this feature is not used.

4.2.8 Traceability
The last feature I will describe is different from the first seven ones. While these defined the expressiveness of the language, traceability is used to draw connections between a generated plan and the planning problem. Traceability is original a term from requirements engineering and described by Gotel [12] as: “The ability to describe and follow the life of a requirement in both forwards and backwards direction”.

Using traceability it is possible to determine why the algorithm imposed a certain ordering of the activities. It should be possible to answer questions like: “Why should I do X, I would rather do Y?” or “Why do we need to do Z, it seems useless?”: If the form of such questions is limited, the understanding should not be difficult. The hard task is to find correct answers to the questions and present them in convenient form. Such abilities are known as expert systems. They have to be deeply integrated into the algorithm. As complex as this seems, traceability is essential for business process modeling as it helps to build the trust in the generated plan.

4.2 Discussion Of Important Planning Algorithms

<table>
<thead>
<tr>
<th>Importance</th>
<th>Graphplan</th>
<th>SGP</th>
<th>Metric-FF</th>
<th>SATPLAN</th>
<th>Blackbox</th>
<th>Burkidan</th>
<th>C-Buridan</th>
<th>UMOP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithm type</td>
<td>-</td>
<td>Graph</td>
<td>Graph</td>
<td>Heuristic Search</td>
<td>SAT</td>
<td>Graph+SAT</td>
<td>own type</td>
<td>own type</td>
</tr>
<tr>
<td>Standardized Language</td>
<td>++</td>
<td>In: PDDL</td>
<td>In: PDDL</td>
<td>In: PDDL</td>
<td>In: PDDL</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Calculus</td>
<td>+++</td>
<td>First-Order</td>
<td>First-Order</td>
<td>First-Order</td>
<td>First-Order</td>
<td>Propositional</td>
<td>Propositional</td>
<td>Propositional</td>
</tr>
<tr>
<td>Conditional effects</td>
<td>*</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Uncertainty</td>
<td>+++</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Sensing</td>
<td>**</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Probabilistic</td>
<td>*</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Arithmetic functions</td>
<td>**</td>
<td>no</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Temporal Planning</td>
<td>-</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Multi-Agent</td>
<td>-</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Traceability</td>
<td>***</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 1. Assessment of the requirements.

Table 1 gives an overview of the different features and their importance. A “-” means that a feature is not important, a “+” means that it is nice to have, a “+++” means that the feature is needed but not essential and a “++++” describes an essential feature.

It is obvious that none of the algorithms fulfills all the features that are needed for business process modeling. It is easy to eliminate Graphplan[3], SATPLAN[14], Blackbox[15], Buridan and C-Buridan[16] from the list of possible algorithms because of their limited feature set. Three algorithms are left for which an implementation of business processes seems possible: SGP[2], Metric-FF[13] and UMOP[20]. SGP has the disadvantage that it only works with CMU-Lisp, which is not available under Windows. Therefore the example process was not implemented in SGP. But
this does not mean that SGP does not fit the requirements. It is just a limitation imposed by the author.

4.3 Implementation Of The Example Process

Now that we know the two algorithms with which we want to implement business process we will try it with a very simple example. The example is the leasing of a flat. This process starts with renting request. Then the tenants and the landlords data have to be fetched and the rent has to be calculated. These three elements are the parts of the lease data which is combined in the next step. Finally the lease is concluded. Although the activities were mentioned one after another, no ordering is needed at the moment. Of course certain activities depend on each other. It is for example needed that tenant and landlord data and the calculated rent are available before the lease data can be combined.

We will first implement the process using Metric-FF. The planning problem consists of two separate files: One containing the domain theory and one containing the problem. I will not show the complete example here, but only demonstrate some features and problems of Metric-FF. Lets first look at the way an activity is described in Metric-FF using PDDL:

```pddl
(:action RentingRequest
   :parameters (?t - tenant ?f - flat)
   :precondition (forall (?ot - tenant)
    (not (isRented ?f ?ot)))
   :effect (requestedFlat ?t ?f)
)
```

This is the description of the renting request. Besides preconditions and effects it has parameters which are needed to execute the activity. Here they are the possible tenant and the flat that he wants to rent. The precondition states that the flat may not be rented by another tenant and the effect is that the tenant made a renting request.

The main problem with Metric-FF is that it does not allow sensing and uncertainty. This means that everything has to be described in the problem and that a new plan has to be generated for every instance of the business process. This makes Metric-FF practically useless. To overcome this problem a modeler could use Metric-FF to model common process instances and derive a general model out of the generated plans. Let us look at such a problem description:

```pddl
(define (problem newTenant)
 (:objects t1 – tenant
   ... Harald Meyer HoherBerg7
 )
 (:init
   (firstname t1 Harald)
   ...
 )
)
We see that a problem description consists of three different parts. First the objects are defined. Here we see another problem: Metric-FF does not know strings. This means that names and addresses have to be defined as the name of objects. Then we describe the initial situation where we for example assign the data to a tenant object. Finally comes the description of the goal which says here that at the end of the process the flat should be rented by the tenant t1.

It makes much more sense to realize the process using UMOP because it allows uncertainty and sensing. It even has a special application with which the plans can be executed. Lets look at the definition of an activity:

\[
\text{agt: tenant} \\
\text{RentingRequest} \\
\text{con: requestedFlat} \\
\text{pre: } \neg \text{isRented} \land \neg \text{requestedFlat} \\
\text{eff: requestedFlat}'
\]

We do again see the renting request activity. Besides the preconditions and effects which do express using propositional logic that to request a flat it may not be rented and no renting request may be made before and that afterwards a request has been done, an additional section called "con" exists. It describes the propositions which are changed by this activity. It is needed for synchronization because UMOP allows multiple agents which can perform activities in parallel. As the first line says this activity can be executed by the tenant agent. Lets finally look at the problem description, where we can see that disjunctive initial conditions are possible (either the rent has already been calculated and acknowledged by the landlord or not):

\[
\text{INITIALLY} \\
\neg \text{requestedFlat} \land \neg \text{isRented} \land \neg \text{tenantDataAvailable} \land \neg \text{leaseDataAvailable} \land \neg \text{landlordDataAvailable} \\
\left(\neg \text{rentAcknowledged} \land \neg \text{rentCalculated}\right) \lor \left(\text{rentAcknowledged} \land \text{rentCalculated}\right) \\
\text{GOAL} \\
\text{isRented}
\]

5 Conclusions and Outlook

Using planning algorithms for business process modeling is not only, but it also makes sense. It simplifies the human part of process modeling to the observation of processes and the modeling of the planning problem. The correct ordering of the activities is then done by the algorithm. It was also shown, that the approach of starting with business process modeling and not with improving workflow management systems using planning algorithm makes sense as the results of the planning algorithm can be verified and adjusted before they are used in a WFMS.
Besides this principle usefulness, existing planning algorithms make it difficult and sometimes impossible to model business processes correctly. But one problem would exist even if all needed features are implemented by an algorithm: Planning algorithms only look at the functional flow describing which activity has to be executed. The modeling of the data flow can only be rudimentarily simulated through preconditions and effects. It still has to be investigated whether it makes sense to embed the data flow inside the planning problem or if it is better to model it afterwards.

Further research is also needed for the successful implementation of automated planning for BPM. It seems obvious that this is not limited to choosing the right algorithm. Equally important is the employees’ acceptance for the generated plan. If they do not trust the automatically generated plans, the plans make no sense. Often a new plan means that old habits have to be thrown overboard. People will only do this, if they think it will give them advantages. This is even more important when interactive replanning is used. The employees are in a new, unexpected situation. Normally they would move back to known habits rather than asking a computer.

If they really let the computer generate a new plan, it has to give them as much trust as possible. One key to this is, that they see how the proposed activities lead to the goal. This may be in the form of a diagram showing the actual state, the goal and the activities that lead to the goal or by using traceability. Another way would be the development of an algorithm which allows the dynamic reconfiguration of the generated plan. If an employee thinks the process should look different, he just changes it. All dependencies are automatically resolved by the algorithm. This way the employee directly sees the outcome of the modification and can give more trust to the generated plan or use the modified plan instead.

Such an approach could also be used by a modeling tool. The business process modeler enters the initial and goal state and the activities with their preconditions and effects and the algorithm generates an initial plan in the background. The modeler can then change this plan, while the algorithm in the background ensures that the goal is still reached and that activities are only executed if it is possible.

References

24. PDDL – The Planning Domain Definition Language V 1.2.
25. PLANET Online Research Database. http://scom.hud.ac.uk/planet/repository/
Workflow in ERP Systems

Antje Rogotzki

Hasso-Plattner-Institut für Softwaresystemtechnik GmbH
at the University Potsdam, Germany
antje.rogotzki@hpi.uni-potsdam.de

Abstract. This paper surveys the concepts and usage of workflow realization in ERP systems. I specialize my considerations to the most wide spread ERP solution: SAP R/3. After a brief introduction to the concepts of the SAP architecture and SAP business workflow is explained. The second part concentrate on workflow modeling in R/3 and describes how the control flow and the data flow is defined. The next part deals with the workflow environment, which includes the techniques of Business Objects, events and organizational management. At the end, I illustrate an example workflow and its implementation in R/3.

1 Introduction

Office and administration organizations in many companies are in change today because competition factors like up market product quality, faster delivery times and intensive customer orientation require a flexible and process oriented organization structure. To develop this productivity capability, a reorganization of business processes as well as IT basic conditions have to be created to coordinate a flexible and application comprehensive control of business processes. Especially the automation of related working steps is very important so that long cycle times of business processes are avoided and costs are minimized. In recent years, there has been a strong shift of paradigm from individual software solutions to standard software systems, leading to the so-called enterprise resource planning (ERP systems). These systems allow a faster time to market than individual software by providing domain and corporate-independent functionality to manage and process data. A workflow management system, which is embedded in an ERP system is such an application comprehensive tool and enable monitoring and controlling of business processes. With the help of the workflow management system the integration of organization management and application components in business processes is possible.

This paper covers the subject of realizing workflows within ERP systems using as example the SAP R/3 system as the most common representative of ERP systems. The advantage of using workflows in SAP R/3 is among other things the personal participation. The workflow management system has the ability to manage the persons who are involved in a process. The goal is to use all personal resources as efficient as possible and compensate human faults. A very frequent execution of a
process results in a higher return on equity of the workflow modeling. An exception is the workflow modeling for emergency. An emergency process has to follow predefined paths and need execution as quick as possible. The amount of potential loss should be minimized here. Workflow systems help to saving cost through the reduction of time which is spent actively by a participant for the process, for example complex navigation proceedings are replaced by simple dialog based steps.

The first part of the paper introduce the SAP concept, which contains the SAP R/3 architecture and the role of the SAP Business Workflow. The next part deals with workflow modeling in R/3. Chapter 4 concentrate on the workflow environment that is provided by the R/3 system. In the last part I describe an example workflow.

2 The SAP Concept

Since Release 3.1 the R/3 standard software contains a workflow component, which is called the SAP Business Workflow (in some literature also SAP Webflow Engine). The SAP business workflow can be used to define business processes that are not yet mapped in the R/3 system.

2.1 Overview of the SAP R/3 Architecture

The R/3 System can be characterized as a multi-tier Client/Server middleware architecture. It consists of three hierarchical layers. The data management layer care about the data storage and data management. The application layer contains the business logic and the transaction management. The presentation layer represents the user interface, show the application data and receive user inputs. The layers are distributed over different computers depending on the client/server architecture.

SAP R/3 was developed as a modular system. It consists of the R/3 basic components and functional related modules (Fig. 1). The modules are optional and can be bought.
Workflow in ERP Systems

separately. They interact with each other. The Basic Components are responsible for the database connection and the communication between the modules. It contains functions for administration and customizing, programming interfaces, system monitoring etc. The workflow component takes care about analyzing, organization and control of business processes.

2.2 The SAP Business Workflow

The SAP Business Workflow is application comprehensive integrated in the R/3 system. It consists of a graphical environment for workflows. Furthermore, it contains a runtime system for monitoring and controlling of workflow execution. The SAP Business Workflow provides several workflow templates that map predefined business processes. These workflows do not require much implementation. In a wider sense the workflow system interact with the application component Human Resources, which defines the organizational plan of the company. Other central components of workflows are business objects. They are defined and implemented in the Business Object Repository.

The following terminology applies for the SAP Business Workflow (Fig. 2).

At definition time we have the workflow definition, which is the description of whole business process considering every decision possibilities and paths. The workflow definition is embedded in a multi step task. A multi step task can be characterized as a formal framework for a workflow definition. When you create a workflow the multi step task is created by the system automatically. A single step task is an abstract of the functional and organizational aspect of a business activity (see [6]). It defines what will be done and who is authorized to do this. Single step task can be background processing without user interaction or dialog execution with user.

Fig. 2. Terminology of the SAP Business Workflow
interaction. A single step task can also reference to a multi step task thereby a hierarchical nesting of workflows is possible. A single step task is assigned to possible agents who execute the tasks. For each task definition you have to define on which objects type the task operates and which methods it invokes. At runtime the workflow definition is represented by a workflow or sometimes called workflow instance. Work items are the separated steps within a workflow. Objects are instances of object types at runtime (see [4]).

3 Workflow Modeling in R/3

A workflow definition is an order of different business activities that are arranged sequentially or parallel. Different workflows of the same workflow definition distinguish between: the concrete objects, which are processed, the responsible agents of the single workitems if they are determining dynamically and the concrete path, which the workflow takes through its definition.

3.1 Control Flow Definition

In addition to the business activities it is necessary to integrate further steps in the workflow, which are used to manage the control flow. A branch in the control flow can be made automatically from the workflow system or with a user dialog. The SAP Business workflow provides the following step types:

**Fork:** The processing that follows takes place in parallel branches. You can define how many branches have to be processed for the fork to be successfully completed or define an end condition.

**User decision:** If the user should go into action in the workflow control, it can be made with a user decision. The user can decide in a dialog which steps are executed next. The choice possibilities have to be defined. For example, at the beginning of a workflow the agent have to decide if the client is a known customer or a new customer. Depending on the decision the workflow system branch in the according path.

**Condition:** Another possibility to manage the control flow is the condition. A condition can be compared with an If-Then-Else instruction. The system evaluates a defined condition and branches the control flow according to the evaluation result. For example, the condition step “Credit checking” is checked if a certain flag exists. This indicates that the client is a new customer. In this case the credit checking is necessary.

**Loops:** With this step a sequence of steps is repeated until the defined termination condition occurs (until loop) or as long as the defined comparison apply (while loop). For example, an application should be revised until the approver approves the application or the applicant withdraws the application. In this case the activities “approve application”, the user decision “Revise/Withdraw” and the activity “Revise Application” are embedded in an “until loop”.

Furthermore, the workflow system provides steps for internal control operations, process controlling and event creation.
Workflow in ERP Systems

With these step types the main workflow patterns like the exclusive choice pattern, the parallel split pattern etc. are supported (see [1]).

3.2 Data Flow Definition

As technical constructs for keeping of data from the environment of tasks, containers are used. They contain control information as object references or constants at runtime. A container is comparable with an internal table. It exists with a local validity for every workflow and every work item. The work item container contains all relevant information of the work item environment. This includes object references to the processing object, the results of methods, and the actual agent of the method and possible references to attachments. The workflow container contains all relevant information about the workflow context. That means, it includes object references to the processed objects in the work flow, an object reference to the workflow itself, which means administration data like the workflow initiator, creation date etc. Furthermore, it contains flags, counter or arbitrary local variables for intern control and administration.

The container definition defines at definition time which elements a container holds. The elements of a container are described with an ID, a data type reference and other properties. So data type verification is possible and consistent data flow between two containers is definable. The predefined elements in the workflow and work item container are filled automatically at run time. The exchanges of information between the components of the workflow system are made according to the mapping rules that are defined in the data flow definition. The tool, which supports the definition of mapping and assignment rules, is the data flow editor.

3.3 Monitoring Deadlines

The runtime system enables deadline monitoring for work item execution. For activities and user decisions you can define deadlines, which are monitored from the workflow system. There are different types of deadlines definable:

“Requested start” is the earliest time at which a work item can be executed. The latest start is the latest date or time for the recipient of a work item to have to start processing it. For example, with the processing of work item “Analyze problem report 12345” have to start not later than two hours after the workflow starts. The “requested end” is a deadline by which the processing of the work item should be terminated. The “latest end” is a point in time for the latest end of processing of a work item. For example, the processing of the work item for “Customer training invitation” have to finished two weeks before starting of the training.

Deadlines can not defined absolute but relatively to a reference in date/time. The behavior of the system in the case of an exceeded deadline is called escalation procedure. You can define reaction to a missed deadline. In this case the missed deadline terminate the work item and the reaction branch from the workflow definition is processed. Furthermore, the workflow system offers the standard
escalation to a missed deadline. In this case the relevant recipients for the missed deadlines are notified but the late work item won’t be completed.

4 Workflow Environment

4.1 Business Objects

In the R/3 System real objects, for example an employee or a customer order is mapped to so called Business Objects. These are built by the black-box principle, that means, structure and implementation details are hided. Business Objects are the interface between the workflow engine and application data and functions. Business Objects serve to encapsulate R/3 data and business processes so that external applications access it over a standardized platform independent interface (BAPI). In a R/3 workflow Business Objects like account, material or customer are usually processed in several steps. Object types consists of the following components:

- Methods are operations on an object. A work item operates always with a method on an object. The method parameters define the interface of the method invocation. Methods can also define exceptions, which will be executed in case of an error. The workflow system reads attributes of an object for control operations, for example, to decide for a branch at condition steps or to determine the responsible agent for a work item. Key fields are unique identifiers for an object. Events indicate state changing of objects. The workflow system reacts on events in starting tasks (triggering events) or terminating tasks (finalizing events). For example, an object “material” has the methods “create material”, the attributes “material name”, “material number” and the events “material deleted” or “material changed”.

- All SAP business object types and their methods are defined and described in the Business Object Repository. The Business Object Repository is a mandant comprehensive directory of every object type definition. It is already delivered with a lot of predefined object types. Object types correspond to classes in object oriented programming paradigm. The following relations are possible between object types: inheritance, association, composition and interfaces. The workflow system can interact out of the business logic of R/3 with arbitrary application at desktops or other systems, if their functions are encapsulated. Therefore, the object technology is elementary for SAP Business Workflow because the integration and interaction of software components of the SAP AG and other software producers is possible.

- SAP R/3 provides two tools to deal with Business Objects. The Business Object Repository Browser allows to finding an object type by using its position in the component hierarchy or by using its relationships to other object types. The Business Object Builder is used to display, create, test, delete or change of Business Objects.
4.2 Events

Events indicate a status change of an object, which is published throughout the system. Each event carries information from its creation context. This information is available to the receiver of the event and can be used for event-driven control and communication mechanism. An event can start, terminate, or continue tasks and workflows. For example, the event “Application created” starts the workflow for the release procedure. In this case the event is a triggering event for the workflow definition. Another example, the event “Purchase order released” can be a finalizing event for a task. The system waits for the event from the creation of the work item and at the arrival of the event the system ends the work item.

The event concept works according to the following procedure:

1. Register in Event-Consumer-Coupling-Table
2. Event produced & published
3. Scan Event-Consumer-Coupling-Table
4. WF system starts event consumer method

Fig. 3. The event registration procedure

At first the event consumer register itself in the Event-Consumer-Coupling-Table for this event. Then the system produces the event to publish the status change of the object throughout the system. The Event-Consumer-Coupling-Table is scanned to find the assignment between events and interested event consumer. Finally the workflow system starts the event consumer with the method, which is registered in the Coupling-Table.

4.4 Organizational Management

The goal is to link the single step tasks not static with certain employees but associate them dynamically on the basis of jobs, positions or organizational units. Therefore a flexible organization management is needed which you could be customizing fast and the organization forms have to be easily changeable. The precondition for linking single step tasks to possible employees is an organizational plan for a specific organization. The organizational plan consists of relations between the different organization objects, which can be presented in a hierarchical structure. The example in Fig. 4 shows an organizational plan for a company.
Fig. 4. Example of the organizational structure of a company

The organizational units represent a group of employees like financial accounting or material management. The organizational unit can be linked to positions. A position is a concrete activity and can be occupied by a holder, for example head of department for financial accounting or secretary in the marketing department. A position can be linked to a user. A user is equivalent to a user account of an employee in the organization. Furthermore, jobs are a general classification of functions in an enterprise, for example administrator or secretary. A job is defined by tasks and characteristics and can be linked to several positions. For example, it is wrong that the single step task “Check Account” is allowed to be execute from Mr. Smith. If Mr. Smith gets a new position in the company he would take along the task. Instead “Check Account” should be allow to execute from all employees, whose position is described with the job “administrator xx”. A single step task is linked to several similarly authorized agents, but can only be executed by exactly one agent.

4.5 Roles

A role is a characteristic or quality of a user from a functional oriented view. An employee can have one or more roles in addition to its organizational classification. Roles are used to specify an agent (or agents) for a task if the set of possible agents is too large, or not specific enough. An example for a role is, orders administrator for customer <customer> as of order amount <order amount> or superior of <user>. An employee holds a role only if the application object or attribute has a specific value. So, roles are used to find the responsible agents for activities or user decisions and associate them dynamically.

Role resolution determine at runtime, which responsible agents have a property described by a role. This improves the ability of workflows to get the right task to the right person at the right time. One procedure for role resolution is the generic
possibility. There the user has to program rules and a nearly arbitrary evaluation is possible. The other possibility uses a responsibility table. The system can read the responsible agents.

<table>
<thead>
<tr>
<th>Role Parameter</th>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer name</td>
<td>Definition of responsibility</td>
</tr>
<tr>
<td></td>
<td>A: Customer name: A-O</td>
</tr>
<tr>
<td></td>
<td>Order Value: until 1000,-</td>
</tr>
<tr>
<td></td>
<td>B: Customer name: P-Z</td>
</tr>
<tr>
<td></td>
<td>Order value: over 1000,-</td>
</tr>
</tbody>
</table>

Fig. 5. Example of a responsibility table

In the example above, two role parameters “Customer name” and “Order value” are defined. In the responsibility table there are two responsibilities A and B and the connected agents to each responsibility.

5 Example Modeling of a trip approval in R/3

To clarify the previous explanations to the topic of workflows I illustrate the implementation of an example workflow in this chapter.

5.1 The Scenario

The selected example references to the business process “trip approval” of a fictive organization unit “sales”. The scenario in this example begins with the completion of a leave request by an employee (requestor or creator of the notification of the trip). The completed form is then forwarded automatically to the head of department (employee’s superior). If the head of department approves the request, the employee receives a notification and the workflow is terminated. If the head of department rejects the request, the employee can decide to revise the request or withdraw it. If the employee decides to revise it, the request form is resubmitted to the head of department after the revision is made. Fig. 6 shows the order of the workflow.
5.2 The organizational structure

In order for the workflow system to establish the relationship between the requestor and their superior, I have created an organizational plan in the system. To keep the example as simple as possible, the organizational plan contains only one administrator and one head of department. The organizational plan consists of one organizational unit ("Sales"), which contains two positions, a head of department and an administrator. Each position is described by one job and each position is assigned one user as holder. The organizational unit can be created with the following transaction: Tools -> SAP Business Workflow -> Development -> Definition Tools -> Organizational Management -> Organizational Plan -> Create. Fig. 7 shows the completed organizational structure.
5.3 Task Definition

The notification of the trip is created in the first step of the workflow. A task is needed to execute this function. The task can be created separately from the workflow definition or directly from the Workflow Builder. This task uses as object type FORMABSENCE and execute the method CREATE from this object. The possible agents should be all employees in the enterprise. Therefore, the task is defined as a general task.

The next task that is incorporated into the workflow as the second step is “Check notification of the trip”. The object type used (FORMABSENCE) and the required method (APPROVE) are already defined and implemented in the Business Object Repository. The task is linked to the job of head of department.

Furthermore, we need the task “Revise notification of the trip” in the workflow definition. This task uses also the object type FORMABSENCE and the method UPDATE. The task is assigned as a general task so that all employees are allowed to execute the task.

5.4 Control Flow and Data Flow Definition

In addition to the task other steps have to be integrated in the workflow definition. The first is the user decision with that the requester can decide to revise or resubmit the notification of the trip if the head of department rejects it. For the user decision the alternative decisions Revise and Withdraw are defined.

If the requestor decides to revise and resubmit the notification of the trip to their superior, the step Check notification of the trip must be executed again. To solving this problem a container element is created in the workflow container that is used as a
flag. This flag contains different values, depending on the status of the notification of the trip.
- Approved
- Not approved and revised
- Not approved and not revised

The container element is used to define the condition in the Until Loop. The last part in the workflow definition is inserting a step in the workflow that sends a notification to the requester after the request has been approved. Fig. 8 shows the complete workflow model as it is displayed in the Work Flow Builder.

![Workflow model of the trip approval](image)

**5.5 Workflow Execution and Reporting**

A workflow can be started manually or with a triggering event. In the example workflow no triggering event is defined and the workflow is started manually. If the workflow initiator is a possible agent he gets the first step automatically for execution. Otherwise the work items are deposited in the Business Workplace Inbox to the according users. To analyze the execution of the workflow log provides different possibilities.
6 Conclusion

The SAP AG reacts with the introduction of their business workflow component to the changing business requirements and the more and more heterogeneous software environment in companies. Enterprises are able to implement next to available standard processes customer specific processes too and can integrate R/3 applications as well as external applications and desktop applications. The main goal is to link the right task to the right person at the right time and to achieve an acceleration of business process cycle. Furthermore, processes can be monitored and controlled department comprehensive and a whole point of view results.

The SAP Business Workflow doesn’t have to look isolated, but always in connection with SAP Business Object technology and the organizational management. The Business Workflow act as integration layer above business application logic and every advantages of the integrated R/3 standard software can be used further on.

To tap the full potential of process control, the companies have to create the organizational basic conditions by their own. That means, in addition to the permanent monitoring and adjustment of the implemented processes the analyzing and statistical examination of executed workflows belongs to it. Therewith a company is able to adjust their workflow processes evolutionary and flexible to new business factors and conditions and to preserve their competitive position.

References

Appendix

The SAP Business Workflow provided a number of tools for defining and analyzing workflows as well as for monitoring operation.

The graphical Workflow Builder

In the Workflow Builder the workflow is actual described. Here, you can insert activities, loops, conditions etc. With the help of the Workflow Builder you can also define the data flow definitions [5].

Fig. 9. Workflow Builder

The Business Workplace

The business workplace provides a standard working environment in which every R/3 user can carry out their share of the business and communication processes in the
Workflow in ERP Systems

enterprise. There, they receive all the work items that are assigned to them in the course of SAP Business Workflow and process the documents that were sent to them from people or from R/3 applications. Each user has a folder in which they can manage documents and work processes. A work item can be processed directly with a double click. The necessary applications are started immediately. One work item is displayed in the Business Workplace of several similar authorized agents, but can only be executed by one agent [5].

![Business Workplace](image)

**Fig. 10. Business Workplace**

### The Workflow Log

The workflow log is a record of a workflow, containing all workflow steps for which processing has been started so far. Any errors during the workflow are displayed in the workflow log and can be analyzed with the help of error messages shown [5].
Fig. 11. Workflow Log – Graphical View
Process Mining

Anne Rozinat

Chair of Business Process Technology
Hasso Plattner Institute for Software Systems Engineering at University of Potsdam
Prof.-Dr.-Helmert-Str. 2-3
D-14480 Potsdam, Germany
Anne.Rozinat@student.hpi.uni-potsdam.de

Abstract. The traditional approach of introducing workflow management to support an existing business process requires a time-consuming design phase for modelling this process before it can be realised and enacted. This leads to many problems such as subjectivity and incompleteness. Therefore process mining (also called workflow mining) can be used to objectively support design or redesign of workflow models by extracting information from log data of real executions. This paper presents a technique for process mining which uses log-based ordering relations to construct the workflow model. In respect of that approach the notion of complete logs and the rediscover ability is examined. Rather practical issues such as noise and timed logs are addressed as well and finally a mining tool is demonstrated.

1 Introduction

In the latest years workflow management systems have proven useful to support structured business processes. By shifting common tasks related to process management from the application level to a system offering general modelling and enactment functionality for business processes (sometimes they are referred to as a “business operating system”) a decoupling is gained similar to the management of data using databases.

Despite its promise, introducing workflow technology to manage complex business processes is far from trivial and there are some serious problems in particular related to the modelling of workflows. Nowadays the application of workflow management systems is driven by process definitions that need to be designed explicitly. The design phase requires the involvement of managers and workers to elicit the underlying process knowledge on the one hand and deep knowledge of the respective workflow language on the other hand. This leads to time consuming discussions and however the results remain subjective and incomplete.

In Fig. 1 the business process management cycle is depicted: After the workflow model has been constructed during the design phase there is an implementation and configuration phase in which the defined process is realised with respect to the limitations and particularities of the respective workflow management system. Afterwards the enactment phase is entered to handle cases according to the previously designed...
specification. Based on the running workflow instances diagnostic data can be collected to be analysed in the diagnosis phase that in turn can provide input for another design or redesign phase to complete the business process management life cycle. Unfortunately in the traditional approach there is little focus on the enactment phase and the diagnosis phase is usually missing.

The idea of process mining (also called workflow mining) is to reverse the process via collecting enactment data at runtime and inducing the underlying workflow model automatically. Note that often workflows are already there, running implicitly without any model specified but registering transaction logs, e.g. using an ERP system. When a workflow management system is introduced to guide these business processes, mining in contrast to modelling is objective as it is based on things that actually took place.

At the first glance it does not seem very useful to rediscover pre-specified workflow models but in fact, it can be very interesting to compare the specified model with the one actually carried out and thus establishing a feedback loop. Nowadays workflow models are getting more and more flexible, e.g. using exception handling. If an exception is carried out in almost every case this gives a hint to the necessity of redesigning the model. So process mining can be useful to support both design and redesign of workflows.

Process mining can be seen as a special form of data mining as data mining aims at the acquisition of knowledge out of huge sets of data [1]. In the case of process mining the knowledge of interest is the process knowledge that is exhibited through the knowledge of people participating in the respective business processes. The sets of data are the execution data of those business processes. So the term process mining is defined as the “method of distilling a structured process description from a set of real executions”[2].
The conceptual formulation towards a process mining method is driven by three main aspects [3]. Firstly, routing structures like sequence, loop, alternative and concurrency need to be detected in an explicit manner to model them appropriately. Furthermore, clearly the resulting model should reflect the behaviour as registered in the process log and finally be as simple as possible.

The remainder of this paper is organised as follows. In the next chapter one theoretical approach is introduced in more detail. In chapter 3 rather practical issues such as dealing with noise and timed logs are addressed. In chapter 4 a mining tool is presented and finally, there is a discussion related to other approaches and future work in chapter 5.

2 Approach of W.M.P. van der Aalst et. al

The approach that is presented in the following has been developed by W.M.P. van der Aalst and his research group [12] at Eindhoven University in the Netherlands. It is introduced in [4] and bases on a firm mathematical foundation.

As a starting point the character of log data process mining is based on needs to be specified. In recording the execution of processes it is assumed that

(i) each event refers to an activity,
(ii) each event refers to a case,
(iii) events are totally ordered [4].

This is not only provided by workflow management systems. Many information systems recording transactional data, e.g. case handling systems or ERP systems, fulfil these criteria as well.

Before moving on some terms from workflow management need to be clarified according to [5]. A workflow process definition specifies which tasks need to be executed and in which order. Cases are handled by executing tasks in a specific order and similar cases can follow the same workflow process definition (i.e. they can be regarded as workflow instances). An activity is a task that is executed for a specific case and by a specific resource. Furthermore the term event remains to be defined in this context: An event is the smallest item that can be recorded while executing business processes. The type of event determines the granularity of the recorded data as on the one hand an event could refer to the execution of a whole activity or, on the other hand, indicate in more detail that a certain activity has been started, aborted, resumed, suspended, …, withdrawn or completed.

Consider the schematic log data that is shown in Table 1. Each event has resulted in one log line which refers to an activity and a case. As in addition they are totally ordered all criteria previously required are fulfilled. Here activities are atomic (i.e. they have no duration) and so each event represents one executed activity. Table 1 is used throughout the remaining section to exemplify the process mining approach.
Table 1. Schematic log data.

<table>
<thead>
<tr>
<th>case identifier</th>
<th>task identifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>case 1</td>
<td>A</td>
</tr>
<tr>
<td>case 2</td>
<td>A</td>
</tr>
<tr>
<td>case 3</td>
<td>A</td>
</tr>
<tr>
<td>case 2</td>
<td>B</td>
</tr>
<tr>
<td>case 5</td>
<td>A</td>
</tr>
<tr>
<td>case 4</td>
<td>A</td>
</tr>
<tr>
<td>case 3</td>
<td>B</td>
</tr>
<tr>
<td>case 1</td>
<td>F</td>
</tr>
<tr>
<td>case 2</td>
<td>B</td>
</tr>
<tr>
<td>case 4</td>
<td>C</td>
</tr>
<tr>
<td>case 2</td>
<td>D</td>
</tr>
<tr>
<td>case 5</td>
<td>F</td>
</tr>
<tr>
<td>case 3</td>
<td>D</td>
</tr>
<tr>
<td>case 1</td>
<td>G</td>
</tr>
<tr>
<td>case 4</td>
<td>C</td>
</tr>
<tr>
<td>case 5</td>
<td>G</td>
</tr>
<tr>
<td>case 3</td>
<td>C</td>
</tr>
<tr>
<td>case 4</td>
<td>C</td>
</tr>
<tr>
<td>case 3</td>
<td>C</td>
</tr>
<tr>
<td>case 2</td>
<td>G</td>
</tr>
<tr>
<td>case 2</td>
<td>E</td>
</tr>
<tr>
<td>case 3</td>
<td>G</td>
</tr>
<tr>
<td>case 4</td>
<td>E</td>
</tr>
</tbody>
</table>

2.1 Preliminaries

Before presenting the process mining approach some preliminary terms and notations are needed. It is assumed that you are familiar with Petri nets and workflow nets, so refer to [4] for further information related to that, if necessary.

Let $T$ be a finite set of tasks. An example is the set of tasks that can be observed in the schematic log data of Table 1, which is $T = \{A, B, C, D, E, F, G\}$. Furthermore $T^*$ denotes the set of all sequences of arbitrary length over alphabet $T$. So the set of $T^*$ over the set of tasks recorded in the schematic log data contains all potential combinations of them like adumbrated in $T^* = \{\emptyset, A, B, C, \ldots, AB, AC, \ldots, AAB, ABA, \ldots\}$.

**Definition 2.1. (Workflow trace)** Let $T$ be a set of tasks. $\sigma \in T^*$ is a workflow trace.

**Definition 2.2. (Workflow log)** Let $T$ be a set of tasks. $W \subseteq T^*$ is a workflow log.

The set of $T^*$ can be seen as the set of all potential behaviours related to a certain set of tasks and in this context a trace is one instance of behaviour from this set. Finally a workflow log consists of a set of traces, i.e. is capable to specify the overall behaviour that was observed from a process.
Now this can be used to express the schematic log data from Table 1 using the notation of W. The reference that has been required to be present from each event to a case is needed here as the actual behaviour should be specified and every case corresponds to an instance of observed behaviour, i.e. can be assigned to a workflow trace. For case 1 and case 5 the workflow trace AFG, for case 2 and case 4 the trace ABCDEG and for case 3 the trace ABDCEG can be assigned. So the overall behaviour that was observed in Table 1 can be specified as a workflow log \( W = \{ \text{AFG, ABCDEG, ABDCEG} \} \). Nevertheless this is a unique but not a one-to-one mapping as W neither does attach its traces to cases nor takes their frequency into account to keep it simple. So it is abstracted here from cases yet they are needed to initially specify the workflow log.

**Definition 2.3. (Log-based ordering relations)** \( >_w, \rightarrow_w, \parallel_w \text{ and } \#_w \) are log-based ordering relations defined as follows. Let \( T \) be a set of tasks. Let \( W \) be a workflow log over \( T \). Let \( a, b \in T \):

- \( a >_w b \) if and only if \( a \) is directly followed by \( b \) in any trace \( \sigma \in W \).
- \( a \rightarrow_w b \) if and only if \( a >_w b \) and \( \text{not}(b >_w a) \).
- \( a \parallel_w b \) if and only if \( a >_w b \) and \( b >_w a \).
- \( a \#_w b \) if and only if \( \text{not}(a >_w b) \) and \( \text{not}(b >_w a) \).

The log-based ordering relations are used to acquire some knowledge with respect to the relation of two tasks. The direct successor relation \( >_w \) is the relation the others are based on. Regarding the example the following direct successor relations can be extracted: \( \text{A} >_w \text{B, A} >_w \text{F, B} >_w \text{C, B} >_w \text{D, C} >_w \text{D, C} >_w \text{E, D} >_w \text{C, D} >_w \text{E, E} >_w \text{G} \) and \( \text{F} >_w \text{G} \).

Two tasks are in causal relation, if one is directly followed by the other one at least once but never vice versa. If that condition holds, there seems to be a kind of necessity of the occurrence of the first task for an occurrence of the second task. Regarding the example the following causal relations can be extracted: \( \text{A} \rightarrow_w \text{B, A} \rightarrow_w \text{F, B} \rightarrow_w \text{C, B} \rightarrow_w \text{D, C} \rightarrow_w \text{E, D} \rightarrow_w \text{E, E} \rightarrow_w \text{G} \) and \( \text{F} \rightarrow_w \text{G} \).

Note that neither \( \text{C} \rightarrow_w \text{D} \) nor \( \text{D} \rightarrow_w \text{C} \) as \( \text{C} >_w \text{D} \) and \( \text{D} >_w \text{C} \). Instead they are in parallel relation: \( \text{C} \parallel_w \text{D} \). This results from the idea of parallelism or concurrency stating that the involved tasks can be executed in any order as they are independent of each other.

It can be proven that the relations \( \rightarrow_w, \leftarrow_w \) (just the inverse of \( \rightarrow_w \)), \( \parallel_w \) and \( \#_w \) are mutually exclusive and partition the set \( T \times T \) [4]. Thus the pairs of tasks not mentioned yet in any causal or parallel relation are in \( \#_w \) relation: \( \text{A} \#_w \text{C, A} \#_w \text{D, A} \#_w \text{E, A} \#_w \text{G, B} \#_w \text{E, B} \#_w \text{F, B} \#_w \text{G, C} \#_w \text{F, C} \#_w \text{G, D} \#_w \text{F, D} \#_w \text{G and E} \#_w \text{F} \).

**2.2 Mining algorithm**

The algorithm called \( \alpha \) uses workflow nets as a modelling formalism and is defined very precisely in a formal manner in [4] and [6]. In the following section it will be demonstrated and explained with the help of the example log data in Table 1.
The main steps to be performed can be summarised as follows:

1. Define the relationships $\rightarrow_w$, $\parallel_w$ and $\#_w$.
2. Link the source place $p_i$ with all $\text{first}(s)$.
3. Link the sink place $p_o$ with all $\text{last}(s)$.
4. Connect all activities that are in $\rightarrow_w$ relation by a place.
5. Merge places that have input or output transitions in $\#_w$ relation.

According to the example the first step has already been accomplished during the last section. The source place $p_i$ and the sink place $p_o$ mentioned in the second and third step stem from the definition of workflow nets [4] and refer to the well-defined creation and completion of a workflow instance. The set of $\text{first}(s)$ contains all tasks that are the first element in any trace $\sigma \in W$ and the set of $\text{last}(s)$ contains all tasks that are the last element in any trace $\sigma \in W$. According to the example $\text{first}(s) = \{A\}$ and $\text{last}(s) = \{G\}$. After performing step 2 and 3 the workflow net fragment looks like depicted in Fig. 2.

![Fig. 2. Workflow net fragment after step 3.](image)

According to step 4 there has to be a connecting place between two tasks that are in causal relation. This is proven in [4] but also complies with intuition. After performing step 4 the workflow net looks like depicted in Fig. 3.

![Fig. 3. Workflow net after step 4.](image)

Although the basic structure has already been established, up to now the workflow net contains only AND-splits/joins but no OR-splits/joins. They are not distinguished until performing step 5. The places within the dotted area suggest that B and F as well as E and F can be executed in parallel but according to the relations defined before only C and D are in $\parallel_w$ relation. So according to step 5 all places having non-concurrent input or output transitions need to be merged. Thus the places within the dotted area are merged in the next step as $B \#_w F$ and $E \#_w F$ (Fig. 4).
Fig. 4. Workflow net after step 5.

Fig. 4 depicts the final workflow net that could be discovered by merely inspecting the workflow log of Table 1. As required from a proper mining method in the beginning the resulting model reflects the behaviour as registered in the log and is as minimal as possible.

2.3 Notion of Completeness

Imagine the trace AFG was missing in the example workflow log W. Clearly the correct model could not be detected as not the whole behaviour has been exhibited. So it is crucial to know what level of completeness is required from the log, i.e. what information is needed to construct the right workflow model.

The simplest assumption would be to require every possible trace to be present in the log, as therefore all possible behaviour would absolutely be exhibited. But the number of traces increases dramatically with an increasing level of concurrency. If there are, e.g., 10 tasks executed in parallel the total number of interleaving are 10! = 3628800.

In case of using the $\alpha$ algorithm, it is fortunately sufficient to require the log being complete with respect to the direct successor relation $>_w$ as all other relations are derived from it. This means that every two tasks that potentially follow each other directly in fact do that in any trace [4]. Particularly in case of a high degree of concurrency this is a vast reduction in contrast to requiring all possible combinations.

Consider for example the workflow net depicted in Fig. 5. The workflow log containing every possible trace would be denoted as $W_1 = \{ABCDE, ABDCE, ACBDE, ACDBE, ADBCE, ADCBE\}$. In comparison a complete log with respect to $>_w$ could result in $W_2 = \{ABDCE, ACBDE, ADBCE, ADCBE\}$.

Fig. 5. Workflow net with 3 concurrent transitions.
2.4. Which Process Can Be Rediscovered?

Instead of defining properties with respect to the workflow log the focus is now changed to the underlying process model. Consider the entity relationship diagram in FMC notation [7] depicting the relations between the underlying workflow model (i.e., the one that is actually executed), the complete workflow log and the induced workflow model (i.e., the one that is discovered using the α mining algorithm) in Fig. 6. The rounded nodes are entities, the rectangular nodes are relations and the cardinalities are given in well-known (min, max) notation. It can be read as follows: Every underlying model can be associated with multiple complete logs and each of them should induce the same workflow model using the α mining algorithm.

If the underlying model is known before it is logged and mined it can be compared with the induced model to measure the mining quality. So it is possible to figure out, whether they are equal, behaviourally equivalent or to which degree they are similar to each other.

![Fig. 6. ERD depicting relations between models and log.](image)

In particular it would be interesting to know which processes can be mined correctly (indicated with the grey area in Fig. 7) and if there is a certain class for which can be guaranteed that every process belonging to it can be mined correctly (indicated with the innermost partition in Fig. 7). So the remaining part of this section deals with the question of which kind of processes can be rediscovered correctly.

![Fig. 7. Partition of the entity set of underlying models.](image)
Definitely there are processes that never can be rediscovered completely. One example for that are implicit places. Places in general are not represented explicitly within the log and are only detected because of the observed behaviour (for the same reason no place names can be rediscovered). Implicit places do not affect the behaviour of a workflow model and thus are not visible in the log. So the $\alpha$ mining algorithm is not able to detect an implicit place like depicted in Fig. 8.

Furthermore $\alpha$ cannot deal with constructs mixing synchronization and choice (or OR-join respectively) like depicted in Fig. 9. The greyly shaded area marks their overlap and so the place and transition within this area are involved in both synchronization and choice (or OR-join respectively). If such constructs were allowed, the direct successor relation $\succ$ would not contain enough information to rediscover the model. The left construct depicts a situation in which the choice between conflicting tasks may be influenced by the order in which preceding tasks are executed; i.e. it is not free at any time. Workflow nets containing non-free choices do not belong to the class of Free-choice WF-nets [5].

There is a class called Structured Workflow-nets (SWF-nets) defined formally in [4] that does not allow for:
(i) implicit places and
(ii) constructs like depicted in Fig. 9 (i.e., the elements of grey line colour are forbidden respectively).
These requirements lead to workflow models whose structure clearly reflects their behaviour.

At the first glance the class of SWF-nets seems to be quite restrictive but it allows for all routing structures encountered in practice: sequential, parallel, conditional and iterative routing (identified by the Workflow Management Coalition in [8]). Moreover, workflow nets not belonging to the class of SWF-nets are typically difficult to understand therefore should be avoided if possible (e.g. Non-free-choice WF-nets are often referred to as “confusion”). Finally, many workflow management systems only allow for the definition of processes that correspond to the class of SWF-nets.
Something else α cannot deal with are short loops, i.e. of length one and two (Fig. 10). In the workflow net on the left there is a loop of length one related to task B. So there should be a causal relation $B \rightarrow_w B$ but as the relation is only defined between two different tasks, loops of length one are not rediscovered by the α mining algorithm. In the workflow net in the middle there is a loop of length two between the tasks B and C. So there should be both a causal relation $B \rightarrow_w C$ and $C \rightarrow_w B$ but as a causal relation is only defined, if one task directly succeeds the other one and not vice versa, loops of length two are not rediscovered by the α mining algorithm. Fortunately loops of length three or more are no problem and can be rediscovered like indicated in the workflow net on the right.

Based on precise formal definitions it is proven in [4] that the α mining algorithm is able to rediscover sound SWF-nets without short loops (soundness is a basic requirement for the correctness of a model as explained in detail in [5]). So for this class of workflow models it can be guaranteed that, assuming the existence of a complete log, they are always rediscovered correctly. Further experiments have shown that even for models outside of this class α induces workflow nets that are either behaviourally equivalent or capture most of the behaviour, so that the mined model still contains a lot of useful information [4].
3 Further Challenges

Besides the theoretical approach presented in the last chapter there are further challenges related to process mining in practice. Two of them are briefly presented in the remainder of this chapter.

3.1. Noise and Incompleteness

So far there has been the assumption of absence of noise and completeness of logs. But one of the most important facts related to process mining in practice is that usually there is noise and there are incomplete logs present when registering execution data. Therefore the ability to deal with noise and incompleteness is essential for the applicability of process mining to real systems.

Noise means that something interferes with the registered log data (e.g. missing registration data or input errors), which may result in missing or interchanged events. Incomplete logs are caused by:

(i) too complex processes; i.e. not enough information can be registered as there are too many possible traces,

(ii) tasks with a very long execution time; i.e. certain traces potentially exhibited by the model never occur, and

(iii) paths of low probability; i.e. resulting from the unbalance of alternatives they might not be present in the log [3].

Without knowing the underlying model in advance of course it is not possible to determine whether a log is complete and free of noise or not. However, based on the experience with pre-specified models heuristic methods can be developed.

In case of noise, for instance, there could be two events referring to task A and B mixed up once within thousands of cases (e.g., all of them contribute to A > B and only one causes B ≥ A), which leads to the fact that because of this erroneously detected direct successor relation B > A the causal relation of A → B cannot be discovered. So obviously it is not feasible anymore to get by with these straight definitions of the log-based ordering relations. Instead it is crucial to expect some percentage of data to be erroneous, which leads to the redefinition of the direct successor and causal relationship. If direct successors in process logs can be discovered in the presence of noise and incomplete logs, it is possible to induce the respective workflow net similar to the α algorithm procedure.

In [10] and [11] these problems are addressed using heuristic methods and metrics. The notion of succession is differentiated such that

(i) a relation called succession relation, i.e. A > B (which corresponds to the direct successor relation from definition 2.3), denotes the number m of cases in which B is succeeding A directly with (A > B) = m, m ≥ 0, and

(ii) a relation called direct succession relation, i.e. A → B (which corresponds to an extended causal relation from definition 2.3), holds, if either

a. (A > B) > 0 and (B > A) = 0 or

b. (A > B) > 0 and (B > A) > 0 and ((A > B) − (B > A)) ≥ σ), σ > 0.
The problem is now to find out, whether two tasks are executed in parallel \(((A>B) > 0 \text{ and } (B>A) > 0)\) or whether they are in direct succession relation with noise \((A>B) > 0 \text{ and } (B>A) > 0\). For this purpose the threshold value \(\sigma\) defines a barrier between parallelism and direct succession with noise. In [11] it is attempted to find automatically for the respective model using a global learning approach (i.e., a logistic regression model is built).

3.2. Timed Logs

Often there are timestamps available in the logs of existing information systems. An evaluation of them can gain measurements like the probability of taking a specific path or the minimum, maximum and average flow time of the whole workflow. Therefor the frequency of traces needs to be taken into account, which leads to a bag of traces rather than a set of traces like the notion was before.

In [9] the \(\alpha\) mining algorithm is extended to analyse time information being attached to every log line. Here the timestamp is considered to correspond to the completion time of the respective activity. Often also the starting time of an activity is registered but assuming only the information about completion time firing a transition is again an atomic action and tokens spend time in places.

Besides the probability of taking a specific path and flow time two kinds of holding time (of a token in a certain place) can be measured, which are waiting time and synchronisation time. Synchronisation time is the time that is passed until a transition is enabled and since then waiting time is registered until the transition is fired.

Indicating real workflows with performance measures is something clearly not supported by contemporary workflow management systems but obviously would be useful to detect optimization possibilities with the help of waiting time measures, for instance. In this context it is important to realise that there is the imperative of paying attention to the respective national legislation regarding privacy and protection of personal data as if information can be traced back to employees, it could be used to systematically measure their performance [9].

4 EMI{T – A Mining Tool

Several mining tools have been developed by researchers to test their approaches. To reduce implementation efforts and facilitate access to their ideas in multiple contexts a tool independent XML format was developed for describing log data. The syntax of this format is specified by the following DTD [12] and is explained in detail in [1]:

```xml
<!ELEMENT WorkFlow_log (source?, process*)>
<!ELEMENT source EMPTY>
<!ATTLIST source
  program (staffware | inconcert | pnet | IBM_MQ | other) #REQUIRED
>
<!ELEMENT process (case*)>
<!ATTLIST process
  id ID #REQUIRED
>`
Here a WorkFlow_log element may contain several processes but regarding the example of Table 1 there was only one process executed. Concerning the process multiple cases can be registered and every case contains an arbitrary number of log lines. In Table 1 a log_line corresponds to a row within the table. So the example log is mapped to this workflow log format by specifying one process element containing five case elements that in turn contain the particular number of log line elements specifying the respective task name.

Fig. 11 shows on the one hand that there are different kinds of transactional systems like ERP systems, CRM systems or case handling systems registering events related to tasks for cases that can be mapped to the common XML format. On the other hand a number of mining tools share this input format as depicted in the lower block.

![Fig. 11. Common XML format [1].](image)

One of those tools is called EMiT (Enhanced Mining Tool), which is presented in [9] and implements the \( \alpha \) algorithm to induce the mined model as a workflow net. By converting log files from Staffware, InConcert and Pnet+ (so far) to the common
XML format it allows for mining workflow logs from real systems. Furthermore it is able to evaluate time stamps by replaying the timed traces in the discovered workflow net. The results can be both displayed like depicted in Fig. 12 and exported to a static or a hypertext-like report.

Fig. 12. Screenshot of EMiT [15].

5 Discussion

This paper introduced a quite intuitive but well-founded approach based on log-based ordering relations with the help of an example. It sketched a heuristic approach to detect direct successors and the idea of gaining performance metrics from timed logs. Finally the mining tool EMiT and a common XML format to describe log data were presented. The researchers want to extend the rediscover ability of the α mining algorithm to find short loops and to take behavioural equivalence into account in the near future [4] and they currently work on mining workflow logs in the presence of noise and incompleteness based on direct successors [11].

Related to process mining a number of different approaches are driven by different focal points [1]. For instance, there is an approach to measure the quality of a mined workflow model described in [1]. The approach of [13] is able to mine workflow models with non-unique task names and borrows ideas from machine learning and grammatical inference. Not mentioned here has been the approach of [14], which exploits the block structure of many processes using term rewriting techniques based on the specification of axioms for commutativity, distributivity etc. (forming an algebra) either.
In the end “the overall goal is to be able to analyse any workflow log without any knowledge of the underlying process and in the presence of noise” [9] and also some further developments seem conceivable related to this quite young and promising area. So the incorporation of case properties (i.e. context data) could be used to predict the routing behaviour of a single case [9]. Moreover the execution time of activities could be considered. And it would be interesting to examine the rediscover ability of certain mining methods from a rather practical point of view like, e.g., workflow patterns that can operate as building blocks in workflow modelling, or even – due to their expressiveness – workflow languages.

References

7. FMC research staff: FMC. http://fmc.hpi.uni-potsdam.de/.