Context-aware workflow management in eHealth applications

Inauguraldissertation

zur
Erlangung der Würde eines Doktors der Philosophie
vorgelegt der
Philosophisch-Naturwissenschaftlichen Fakultät
der Universität Basel
von

NADINE FRÖHLICH
aus Wolfen, Deutschland

Basel, 2016

Genehmigt von der Philosophisch-Naturwissenschaftlichen Fakultät
auf Antrag von

Prof. Dr. Heiko Schuldt, Universität Basel, Dissertationsleiter
Prof. Dr. Andreas Meier, Universität Freiburg, Korreferent

Basel, den 16. September 2014

Prof. Dr. Jörg Schibler, Dekan

Namensnennung-Keine kommerzielle Nutzung-Keine Bearbeitung 3.0 Schweiz
(CC BY-NC-ND 3.0 CH)

Sie dürfen: **Teilen** — den Inhalt kopieren, verbreiten und zugänglich machen

Unter den folgenden Bedingungen:

Namensnennung — Sie müssen den Namen des Autors/Rechteinhabers in der von ihm festgelegten Weise nennen.

Keine kommerzielle Nutzung — Sie dürfen diesen Inhalt nicht für kommerzielle Zwecke nutzen.

Keine Bearbeitung erlaubt — Sie dürfen diesen Inhalt nicht bearbeiten, abwandeln oder in anderer Weise verändern.

Wobei gilt:

- **Verzichtserklärung —** Jede der vorgenannten Bedingungen kann **aufgehoben** werden, sofern Sie die ausdrückliche Einwilligung des Rechteinhabers dazu erhalten.
- **Public Domain (gemeinfreie oder nicht-schützbare Inhalte) —** Soweit das Werk, der Inhalt oder irgendein Teil davon zur Public Domain der jeweiligen Rechtsordnung gehört, wird dieser Status von der Lizenz in keiner Weise berührt.
- **Sonstige Rechte —** Die Lizenz hat keinerlei Einfluss auf die folgenden Rechte:
    - Die Rechte, die jedermann wegen der Schranken des Urheberrechts oder aufgrund gesetzlicher Erlaubnisse zustehen (in einigen Ländern als grundsätzliche Doktrin des **fair use** bekannt);
    - Die **Persönlichkeitsrechte** des Urhebers;
    - Rechte anderer Personen, entweder am Lizenzgegenstand selber oder bezüglich seiner Verwendung, zum Beispiel für **Werbung** oder Privatsphärenschutz.
- **Hinweis —** Bei jeder Nutzung oder Verbreitung müssen Sie anderen alle Lizenzbedingungen mitteilen, die für diesen Inhalt gelten. Am einfachsten ist es, an entsprechender Stelle einen Link auf diese Seite einzubinden.

Quelle: http://creativecommons.org/licenses/by-nc-nd/3.0/ch/ Datum: 12.11.2013

Contents

Abstract

Workflows are a technology to structure work in functional, non-overlapping steps. They define not only the order of execution of the steps, and describe whether steps are executed in parallel, they also specify who or what tool has to fulfill which step. Workflows offer the possibility to automate work, to increase the understandability of processes, and they ease the control of process execution. The tools to manage workflows, so called workflow management systems (WfMSs), are traditionally rigid as they separate workflow definition done at build time from workflow execution done at run time. This makes them ill-suited for managing flexible and unstructured workflows. In this thesis, we focus on the support of flexible processes in eHealth, which are affected by more foreseen than unforeseen events. To bridge the gap between rigid WfMSs and flexible workflows, we developed a concept for dynamic and context-aware workflow management called *Flexwoman*. Although our focus lies on flexible eHealth processes, Flexwoman is a generic approach that can be applied to several different application domains. Flexwoman supports the usage of context information to adapt processes automatically at run time to foreseen events. Processes can also be manually adapted to handle unforeseen events. To achieve this flexibility, context information from different sensors is unified and thus can be analyzed in the same way. The analysis and adaptation of workflows is executed with a rule engine. A rule engine can store, reason about and apply knowledge automatically and efficiently. Rules and application logic are separated, thus, rules can be changed during run time without affecting application logic or process description. Workflows are internally described by Hierarchical Colored Petri nets (HCPNs) and executed by a HCPN execution engine. HCPNs allow for a deterministic execution of workflows and can represent workflows on different levels of detail. In summary, in Flexwoman, significant context changes (events) trigger automated adaptations that replace parts of the workflow by sub workflows, which can in turn be adapted. The adaptations and the rules for context-aware adaptation are saved in the organizational memory for later reuse. Flexwoman's event based behavior facilitates proactive adaptations instead of only allowing for adaptations while entering or leaving a task. Replacements are not bound to special places defined at build time but each part of the workflow, which has not been executed yet, can be replaced at run time. We implemented and evaluated the concept. The evaluations show i) that all required functionality is available, ii) that the system scales with a growing number of rules, and iii) that the system correctly handles failure situations.

Zusammenfassung

Workflows strukturieren einen Arbeitsablauf in funktional nicht überlappende Aktivitäten. Dabei definieren sie nicht nur die Ausführungsreihenfolge dieser Schritte, sondern beschreiben zusätzlich, ob diese Schritte nebenläufig ausgeführt werden und wer oder welches Werkzeug eine bestimmte Aktivität ausführt. Workflows ermöglichen eine Automatisierung des Arbeitsablaufs, verbessern die Verständlichkeit von Prozessen und erleichtern die Überwachung der Prozessausführung. Workflow-Management-Systeme (WfMS) sind eine Software zur Definition, Ausführung und Überwachung von Workflows. Herkömmliche WfMS sind starr, denn sie trennen die Workflow-Definition, die zur Definitionszeit erstellt wird, strikt von der Ausführung der Workflows zur Laufzeit. Dadurch sind diese Systeme ungeeignet für das Management flexibler und unstrukturierter Workflows. In dieser Arbeit wollen wir flexible eHealth-Prozesse unterstützen, die von mehr vorhersehbaren als unvorhersehbaren Ereignissen betroffen sind. Um die Lücke zwischen starren WfMS und flexiblen Workflows zu schliessen, haben wir mit Flexwoman ein Konzept zum dynamischen und kontextabhängigen Workflow-Management entwickelt. Flexwoman wird in der Arbeit zwar auf eHealth-Prozesse angewandt, ist aber ein generischer Ansatz, der in vielen unterschiedlichen Anwendungsdomänen eingesetzt werden kann. Flexwoman benutzt Kontextinformationen zur automatischen Laufzeitanpassung von Prozessen an vorhergesehene Ereignisse. Flexwoman erlaubt zusätzlich eine manuelle Anpassung von Workflows an unvorhergesehene Ereignisse. Um eine solche Flexibilität zu erreichen, werden die Kontextinformationen, die von verschiedenen Sensoren kommen, intern vereinheitlicht. Somit können sie auch auf einheitliche Weise analysiert werden. Die Analyse der Kontextinformationen und die Anpassung der Workflows werden mit einer Rule-Engine ausgeführt, die vorhandenes Wissen nicht nur speichern kann, sondern auch automatisch und effizient über dieses Wissen schliessen kann. Da Regeln und Anwendungslogik separiert werden, können Regeln zur Laufzeit geändert werden, ohne die Anwendungslogik oder die Prozessbeschreibung zu beeinflussen. In Flexwoman werden Workflows intern durch hierarchische gefärbte Petrinetze (HCPN) beschrieben. Sie werden von einer HCPN Execution-Engine ausgeführt. Wir haben HCPN für die interne Darstellung gewählt, weil sie eine deterministische Ausführung von Workflows erlauben und Workflows auf verschiedenen Detailebenen repräsentieren können. Zusammenfassend stossen signifikante Kontextänderungen (Ereignisse) automatische Anpassungen der Workflows an. Dabei werden Teile des Workflows durch Subworkflows ersetzt, die

wiederum angepasst werden können. Die Anpassungen (Subworkflows) und die Regeln für diese kontextabhängigen Anpassungen stellen Lösungen und Expertenwissen dar, die für die spätere Wiederverwendung dokumentieren werden. Flexwoman ermöglicht eine vorausschauende Anpassung von Workflows und führt Anpassungen nicht nur beim Starten oder Beenden von Aktivitäten durch. Anpassungen sind nicht an bestimmte Stellen des Workflows gebunden, die zur Laufzeit festgelegt worden sind, sondern Anpassungen sind zur Laufzeit an jeder Stelle des Workflows möglich, vorausgesetzt, sie ist noch nicht ausgeführt worden. Wir haben das Konzept implementiert und in einer Evaluation konnten wir zeigen, dass die benötigte Funktionalität im Prototypen vorhanden ist und das System auch bei einer wachsenden Anzahl von Regeln skaliert.

Acknowledgement

It was a long way to finish this work. I want to thank the people who supported me by encouragement, friendship or love on this journey.

First, I want to thank my supervisor Prof. Heiko Schuldt for his guidance and his support, for encouraging and pushing me to finish my work and for the family friendly atmosphere he established in the DBIS group. I would like to thank my co-supervisor Prof. Andreas Meier for taking the time to review this work. Additionally, I want to thank Prof. Georg Paul from the Otto von Guericke University Magdeburg who gave me the opportunity to start a Ph.D. and believed in my abilities as a researcher.

My gratitude goes to Marcel Lüthi, Christoph Langguth and Thorsten Möller for proof reading and helpful comments. I would like to thank the former DBIS team for the discussions and collaborations but also for the social time we shared together. My special thanks goes to Thorsten Möller, Christoph Langguth, and Laura Voicu. I wish to thank our project partners from the University Fribourg for the great collaboration in the LoCa project.

I want to thank Anita Lerch, Julia Vogt, Melina Inderbizin, Roman Troyer, Elke Schlote and Susanna Parikka-Hug for their friendship and their moral support.

Finally, I want to thank my family. I thank my parents for always believing in my abilities. I want to thank Marcel Lüthi for always supporting and encouraging me. I thank Nathalie and Maggie for showing me that there are other things in life than this thesis.

Chapter 1

Introduction

Our world changes frequently and fast. Increased competition especially on the global market results in technological and procedural advances and consequently in processes that underlie continuing revolution. Traditionally, processes that make use of computerized support are stable, structured and often repeated. Consequentially, tools, so called workflow management systems (WfMSs), developed to handle such processes fit perfectly the requirements of these processes. However, processes that are dynamic and have to deal with frequent or unforeseen changes, unstructured or less structured processes as well as rarely executed processes are not or insufficiently supported by traditional WfMSs.

One typical domain affected by these problems is healthcare. Exceptions such as emergencies, unexpected reactions on treatments, new insights that require new treatment methods, are day-to-day business. Thus, many of the processes in healthcare are characterized by dynamic changes, often they are only partly structured to deal with the dynamics. Also for dynamic processes it is advantageous, when work is modeled with WfMSs as well as coordinated and controlled by WfMSs. An important advantage of this approach is that common procedures and their execution are documented and the expertise about these processes is captured. This allows an easy understanding and reused of expertise, avoids errors in treatments and eases work. But conventional WfMSs cannot manage and execute dynamic processes as they occur in real life. These WfMSs lack the possibility to react on unforeseen events as well as to handle not completely specified workflows. When using conventional WfMSs for handling dynamic processes, dynamic processes have to be defined as static. For handling dynamic changes that characterize dynamic processes, conventional WfMSs have to be bypassed. Consequently, dynamic processes and their process executions are not completely documented when modeled using conventional WfMSs. For processes that are only partly structured it is similar. When using conventional WfMSs for their management, they have to be designed as structured ones. The option to use unstructured process parts to react flexible to exception is lost.

A specific kind of software, which was developed to overcome the above described problems, are dynamic workflow management systems. The concepts of dynamic workflow management

are diverse and have different foci. Some concepts aim at supporting changes of the workflow definition, others at supporting changes of the workflow instance. Some concepts require a definition of the complete workflow at build time and apply changes at run time. Other concepts require only a partial definition of the workflow at build time. In the latter case, process parts known at build time and place holders for the at build time unknown process parts have to be defined. At run time unknown process parts have to be specified. All approaches of dynamic workflow management have in common that adaptations do not cause interruptions of the workflow execution. Existing dynamic workflow management tools often require manual input to adapt the workflow. In some application domains, such as healthcare, mistakes can be fatal. In these domains adaptations that are not known in advance can only be applied manually by an expert. Nevertheless, in healthcare there are often adaptations that are routine jobs that are known in advance. For these adaptations automated adaptations are beneficial as they save time and are less error prone than manual adaptations.

Automatic adaptation requires the usage of additional knowledge. It requires knowledge about the situation and environment of the user, the so called user context. Examples of context are location, time, or physical parameters of a user such as blood pressure. The usage of context does not come for free. Context data has to be captured and analyzed before it can be used for workflow adaptation. This requires time and resources. The time needed for context capturing and extraction should not be higher than the time needed for manual input, otherwise the application will not be accepted. Thus, a big challenge, which will be addressed, in this work is the automatic and frequent acquisition and processing of context data and context data changes in a way that hides needed expenditure of time and resources from the user. This is even more challenging in mobile settings where context changes more often. Such settings are typical in eHealth as software meaningful for physicians often runs on mobile devices. Fortunately, today's mobile devices are powerful enough to acquire and use context, in contrast to mobile devices some years ago.

In summary, this work makes use of context to adapt processes automatically and flexibly to frequently changing user context. We focus on users of mobile devices in a healthcare environment which are faced with frequent context changes. We developed a concept for context-aware workflow management. For realization of this concept we developed an architecture and implement it. We call the resulting system **Flexwoman**, which stands for **flex**ible **wo**rkflow **man**agement. Flexwoman is qualitatively evaluated to check whether the required functionality is correctly implemented. Additionally, Flexwoman is quantitatively evaluated to prove that the system scales with a growing number of rules in the rule base and correctly handles failure situations.

The following sections will present the main contributions of this thesis and introduce our application scenario that is located in the healthcare sector.

1.1 Major contribution

The major contribution of this thesis is the development of a general concept and system, called Flexwoman that makes it possible to use context to adapt (mobile) workflows automatically to their environment. Flexwoman focuses at the adaptation of workflows and the handling of not only foreseen, but also unforeseen context changes. The management of unforeseen changes differentiates Flexwoman from conventional WfMSs. The option of pre-planning of adaptations that are known at build time in combination with the possibility to execute workflow adaptation at every point of the running workflow that is not executed differentiates Flexwoman from current dynamic WfMSs.

We made the following detailed contributions to realize the development of Flexwoman:

- We developed a general concept for context-aware and dynamic workflow management.
- We analyzed existing approaches for dynamic and context-aware workflow management.
- We developed an architecture that realizes our concept based on the well known workflow reference model. We extended this model with components that enable the dynamic and context-aware adaptation of workflows. Our concept of dynamics is partly based on the open-point approach, an approach for dynamic workflow management. It retains the advantages of the open-point approach and improves its weaknesses.
- We analyzed which methods are convenient to realize our architecture and designed a system based on Colored Petri nets (CPNs) and Rule-based systems (RBSs). CPNs are the base for workflow execution and run time adaptation of workflows. RBSs are used for analyzing the relevance of context changes and trigger the adaptation of the workflows by using rules.
- We formally described the main concepts of Flexwoman using Colored Petri nets.
- We prototypically implemented Flexwoman based on two kinds of components (i) components that are developed from the scratch and (ii) components that reuse and extend tools we evaluated before for their suitability.
- We successfully evaluated the implementation of our concept with a qualitative and a quantitative evaluation. The qualitative evaluation showed that all required system functionality is available within Flexwoman. The quantitative evaluation proves that the system scales when the number of used rules grows.

1.2 Application scenario

As described before we develop in this work a general concept for the adaptation of workflows in dynamic environments. This concept can be applied in various domains. It is especially of benefit in domains where information has to be adapted fast and correct to a frequently changing situation, such as fire fighting and emergency management. The concept can also support users with information about the nearest shopping mall or museum.

For the illustration of our concepts, we use an application scenario from the healthcare sector or more precisely, we use an eHealth scenario[1]. It describes the daily life of a person suffering from diabetes who applies the Intensified Conventional Therapy (ICT). In this scenario we model the most frequently occurring exceptional circumstances that can happen in the daily life of a diabetic. We have taken this scenario because it is a manageable and realistic scenario that makes use of context[2] and it shows our idea of adaptable software based on Flexwoman.

We shortly explain the most important concepts and terms needed in our scenario: *Diabetes mellitus* is a metabolic disease. A person suffering from this disease has too much glucose in the blood plasma (hyperglycemia). There are several therapies for the different types of this disease. Our application scenario is a simplified detail of the daily routine of a diabetic who applies the Intensified Conventional Therapy. This therapy imitates the insulin secretion of a healthy pancreas. This allows the diabetic to live a nearly normal life with few constraints. It is the recommended therapy for diabetics suffering from *diabetes mellitus* type 1.

We will consider the following scenario: Bob is a 30 year old diabetic suffering from diabetes mellitus type 1. Before breakfast he measures his blood sugar level. Then he thinks about what he wants to eat and determines how many carbohydrates the meal has. By using the value of the blood sugar and the carbohydrates he calculates the dosage of insulin to inject. Thereby, he uses an application running as a Flexwoman process. Then he injects short acting insulin and has breakfast. For lunch and dinner Bob follows the same procedure. Additionally, Bob injects long acting insulin in the morning before breakfast and in the evening before lunch. The reminder for that comes from the application running on Flexwoman. This application shows him also the dosage he needs to inject.
One summer day, Bob does not feel well. He feels dizzy and sweats, he is thirsty and has a blurred vision. Thus, he measures his blood sugar - 8.9 mmol/L, and realizes that it is too high. The application advises him to inject 1/5 of his daily dosage of insulin and to drink water. After two hours he does not feel better. He asks his wife to meter his blood sugar level. The blood sugar level is 17 mmol/L. Now the application informs Bob's physician and calls automatically an ambulance. In the hospital his blood sugar level is stabilized.

[1] EHealth is the use of information and communication technology to support healthcare [BCF+08].

[2] The facts used in our application scenario are taken from http://de.wikipedia.org/wiki/Insulintherapie, http://www.diabetologieportal.de/Therapieformen/Intensivierte-konventionelle-Insulin-Therapie-(ICT).htm?ID=42, http://en.wikipedia.org/wiki/Hypoglycemia

One year later he feels tired, cannot concentrate and shivers. He measures his blood sugar. It is too low: 2.4 mmol/L. The application advises him again what to do. He takes 15g of carbohydrate. Slowly he feels better and recovers completely during the day.

1.3 Outline

In Chapter 2 we describe the fundamentals of this work. We develop a context-aware WfMS, thus we introduce the concepts context and context-awareness, as well as we describe and compare conventional and dynamic workflow management. As context-aware workflow management is a special kind of dynamic workflow management, we examine formal models of dynamic workflow management, to find out whether they can be used for Flexwoman. The focus of the examination lays on the different kinds of Petri nets. In our work we use rules to react on context changes with adaptations of workflows. Rule based systems are also presented in this chapter.
In Chapter 3, we describe the motivation and concept of dynamic context-aware workflow management with our system Flexwoman. The realization of Flexwoman, that is the architecture with its components and the implementation, are described in Chapter 4. In Chapter 5 we evaluate the overall system qualitatively as well as quantitatively. Chapter 6 introduces related work and compares systems related with our work – Flexwoman. Chapter 7 concludes with conclusion and outlook on future work directions.

Chapter 2

Fundamentals

The goal of this work is to develop an approach that allows an adaptation of the structure of workflows, (i) manually for unknown events (context changes), and (ii) to the greatest possible extent automatically, based on context information for known events. Our focus lies in the support of mobile environments where context changes frequently but individually for every mobile user. That is, we will care about structural changes of workflow instances and not about workflow definitions. This chapter describes the fundamentals needed as foundation of this work and it is structured as follows:

Firstly, we clarify the terms *context* and *context-awareness* in the area of application development in Section 2.1 and 2.2. Afterwards, we take a look at workflows in general and we will review structural adaptations to workflows, so called flexible or dynamic adaptations. The adaptation techniques usually do not act on the assumption of automatic adaptation. They assume that manual interaction is necessary. Context-aware workflows aim to overcome the dependency on manual input by using automatically acquired context instead of manual input. Section 2.3 will discuss general workflow topics. Section 2.4 will introduce dynamic workflow management. A discussion of Petri nets in Section 2.5 will conclude this chapter. We decided to take Petri nets as the description language for our approach because Petri nets have a strong mathematical foundation, and are well documented. Models can be analyzed by simulation and formal analysis methods. On the other hand, Petri nets can easily be mapped to workflows and they offer a graphical notation to describe processes, which eases the understanding of Petri nets considerably.

2.1 Context

This section will explain our point of view on the usage of context in computer applications and discuss the relationship of context and ubiquitous computing. We will introduce some important definitions of context in computer applications and we will explain the aspects of context related

to our work.

The idea of using context in computer applications is related to the idea of ubiquitous computing. Ubiquitous computing is a technology that makes many computers available throughout the physical environment but effectively these computers are invisible to the users [Wei93]. In [Wei93] Weiser complains, that the computer often remains the focus of user attention instead of being a working tool, disappearing from the user's awareness. In the following text we will explain how context helps to improve this situation and which further advantages the usage of context has.

According to the The American Heritage Dictionary [Dic09], *Context are the circumstances in which an event occurs; a setting.* Or in slightly different words [LLC]: *Context is the set of circumstances or facts that surround a particular event, situation, etc.*.

A particular situation can be a talk of two persons. The relationship of these people is context that influences their talk essentially. For instance, the relationship of the people will influence the topics and the directness of their talk. A person, suffering from diabetes, will talk about urgent problems regarding this disease with the physician in charge and not with unknown persons he meets in the coffee break of business meeting. With foreign persons he will do small talk and not discuss current health states. The human to human interaction is heavily affected by the context of the interaction (see also [DA00]) and it is of high importance for humans to react on context in the right way. Therefore, humans are well trained in using context and calculate the position usually extremely well.

Also in human computer interaction (HCI) it is of great advantage when computers can use context to influence the application in a way valuable for the user. For instance, it can be useful when the application shows information which is important for the user in the current context or when the application adapts itself to the context. An example for context use in HCI is an application that gets blood sugar values of a patient via WIFI or bluetooth. This application could visualize the value or it could automatically compute whether the patient is hyperglycemic or hypoglycemic. The application could propose actions to take by the patient such as sending a text message (SMS) to the physician in charge or call the physician. Furthermore, it could execute these actions in critical situations automatically without human interventions.

To achieve such a behavior, computers have to detect context, to choose important context, to process the context and to react in a convenient way on the processed context information. When the computer fulfills these tasks as far as possible automatically the user is freed from doing these tasks himself/herself. The user can concentrate on his/her core tasks respectively real goals. The attention of the user is no longer focused to the computer, just like Weiser demands. Therefore, the automated usage of context in applications is a means to come closer to ubiquitous computing. Based on that fact, the idea of context quickly spread to other application areas.

However, when people started to think about applications that use context, and tried to imple-

ment them, they realized that they could not describe what context exactly is. The problem was that context is a moving target, every situation has its own context and for different applications used in the same situation different kinds of context can be important. But without a clear and applicable definition of context in the application area of human computer interaction it was difficult to apply the concept. People started by describing context by synonyms and by examples [DA00]. Examples of context are only of limited usefulness. They can give an idea what context can be, but no more. As a side effect, location, which was identified as an example for context, was extensively used in context-aware applications. Schmitt et al. [SBG99] described this as a shortcoming and presented a variety of further examples of context. But they also did not give a definition of context. Afterwards, many attempts were made to find a general definition of context. Dey and Abowd [DA00] formulated one of the most commonly used and most convincing definition of context:

"Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves."

In other words, context is something that characterizes the situation in which user-application-interaction happens. Therefore, it is only a part of the whole situation. Winograd [Win01] criticizes the above definition as including too much information which is irrelevant for an application developer. He limits context to that part of the environment, that is effectively used by the application.

"something is context because of the way it is used in interpretation not due to its inherent properties [...] Features of the world become context through their use."

Consider the example in which a person suffering from diabetes measures the blood sugar level. Context, according to Deys and Abowds definition, can be the time of measurement, circumstances of measurement, such as, was the person exercising before, or is the person traveling at the moment. But also persons nearby are context. In reality, persons nearby during measurement are irrelevant. The irrelevant context data are excluded by Winograd. In practice, both definitions are relevant. The first definition explains what can be used as context data and the second definition describes that only a part of this data should be used for application development. This exclusion of irrelevant information reduces the amount of context data that has to be acquired and stored. This is especially important in conjunction with the usage of context in resource limited mobile environments. Thereby, it is not always trivial for the application designer to decide which part of the context is relevant for an application. This kind of context relevant to an application will be called *relevant context* in the following text.

Another problem the application developer is faced with is that the acquisition of relevant context is sometimes not possible, extremely difficult (complex) or extremely expensive. For instance, the blood sugar value of a diabetic and the carbohydrates he/she wants to eat decide on the

insulin dose he/she has to take[1]. The estimation of carbohydrate from a picture of the meal is technically difficult. The more exact estimation from tables with meal components and belonging carbohydrates is time consuming. Also the application based version of this proceeding is time consuming as it requires a manual selection of all components of the meal. Therefore, the diabetics usually estimate the carbohydrates based on experiences and input the estimated value directly to the application. Another example is the acquisition of emotions. Emotions, especially emotions leading to mental pressure, influence the blood sugar level. To acquire emotions, to find reasons for blood sugar variability, can be done by capturing and analyzing physical values such as blood pressure or heart rate. A frequent monitoring of these values to capture emotions is usually too expensive.

When the cost of acquisition slows down the execution of an application, or the acquisition requires that many resources that the target device cannot handle the application in a user friendly way, the application developer has to decide about the necessity of that value for the application. Is this value highly relevant, alternative ways for acquisition have to be found, for instance manual input, or the application can not be implemented. Is the value not highly relevant for the application, the context value can be omitted. That is, the application should run smoothly on the target devices of the application, e.g., mobile devices. The context that is really used in an application is called *used context* in the reminder of the text.

A formal description of our understanding of context, explained in detail before, is inscribed in the following definitions:

Definition 1 (Context [DA00]) . Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. □

Definition 2 (Relevant Context) . Relevant Context is a part of the context of a specific application. Relevant context includes only that part of the application context, that can be meaningful used to reach the application goal. □

Definition 3 (Used Context) . Used context is that part of relevant context that a specific application uses to reach its goal. □

Figure 2.1 illustrates our understanding of context inferred by the definitions described above. The circle, named *context*, is the concept described in the definition of [DA00]. *Relevant context* is the concept described as context by [Win01]. It is a smaller subset of our context definition. *Used context* is that part of *relevant* context an application developer is able to use.[2] Context data are symbolized as points in Figure 2.1. At this point we want to stress the fact that context is

[1] This is a simplified point of view.

[2] The circles for used and relevant context do not have such a big difference in size as the circles of context and relevant context. This is because of the fact that a very high percentage of relevant context is recognized by the developer as relevant and is additionally technically usable and therefore used as application context. Context as a

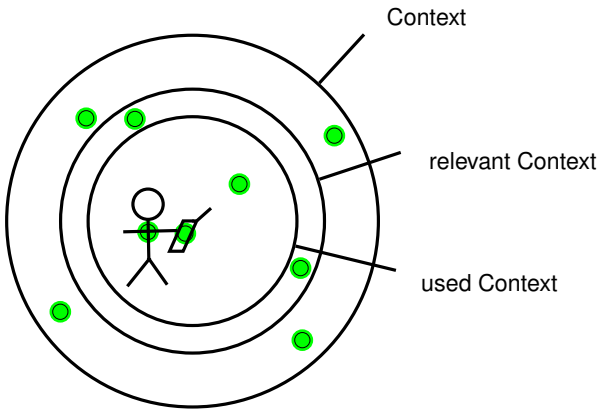


Figure 2.1 – Context in a user-application-interaction (The green dots symbolize context data.)

usually a combination of several context data. It is the combination that can help to get a better understanding of the current situation [CK00] and therefore a better adaptation of the application to the current situation. For example, when the physician enters a sick room we can combine his/her profile data as well as his/her time and location data to get only information about the patients he/she is in charge of. Additionally, the above example shows that it is possible to find context of entities by using context of other entities. For example, by knowing the location of the physician we can explore the context of this location to find, for instance, the patients in the sick room the physician is in charge of.

For different applications acting in the same context that are used by the one or more users usually the *relevant context* differs because relevancy of context depends on the goals of the application. This is also true for *used context* as a sub set of relevant context(see also Figure 2.2).

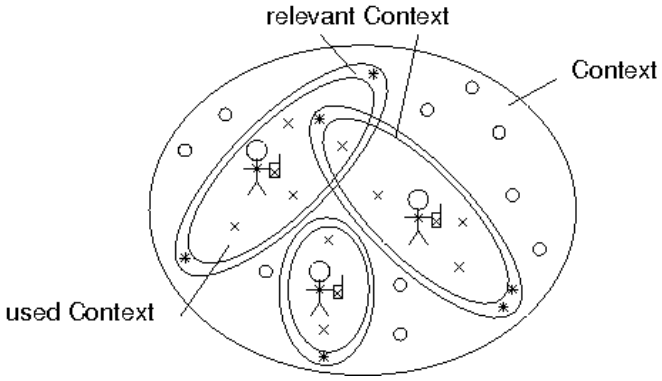


Figure 2.2 – Potential relation of used and relevant context for three parties in the same situation

Beside the introduced classification, Soylu et al. propose in [SCD09] a categorization into low

whole is extremely large compared to the relevant context and therefore the circle for context is much larger. But the sizes of the circles have only symbolic character and do not base on numbers.

level and high level context. They describe low level context as raw context usually sensed or given as manual input and not processed. In contrast, high level context is derived from low level context, by operations such as conversion, for instance from mg/dL to mmol/L[3], abstraction/generalization, for instance from a blood sugar value to a rough scale such as ok, hyperglycemic, hypoglycemic, or cleaning, for instance using the statistical quantities such as mean or standard deviation to find erroneous values and reset them e.g. to an average. Especially a cleaning of data can be important because context is, as pointed out in [SCD09], imperfect. The data can be imprecise dependent on the measurement equipment, incomplete when consisting of more than one value as GPS data, or ambiguous when computed from values measured by more than one measurement instrument.

For practical reasons, it is of value to notice that the definitions show that context is a heterogeneous concept. The term context includes frequently updated and never or rarely changed context data. Grossmann et al. [GBH+05] introduced the following classification in dynamic and static context, which can be useful in supporting efficient context data management:
Frequently updated context, so called dynamic context, is usually automatically acquired by sensors. This context is affected by more updates than queries. Examples for this context class are blood sugar, time, location. In contrast, rarely changed context, such as name, and date of birth, is manually acquired or semi-automatically acquired for instance by user profiling. This so called static context is affected by more queries than updates.

2.2 Context-awareness

Context-awareness is a term that describes the relationship of context and application. There is also a multitude of different definitions of the term *context-aware*. We will use the term synonymously to the terms *context-dependent* and *context-sensitive* as it is often done.

Dey and Abowd [DA00] state that:

"A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task."

They do not include detection and interpretation of context in the definition but only require the response to context as necessary property of a context-aware system. Thereby, they stress the fact that only relevant information/services are provided to the user. They do not further discuss relevance of context.

Chen and Kotz [CK00] have a different view on context-awareness and do not limit the definition to a reaction on context changes. They describe context-aware computing as:

"a mobile computing paradigm in which applications can discover and take advantage of con-

[3] That are two different units of measurement for blood sugar, needed in different countries. mg/dL is an old unit of measurement, mmol/L is SI-conform.

text."

This definition is also very general. It states that context-awareness means both recognition and usage of context. It includes applications that only show information to the user as well as applications that adapt automatically to context changes. This is similar to the definition of Dey and Abowd. But Chen and Kotz explicitly include the discovery of context. The interpretation or processing of context is not explicitly mentioned as important part of a context-aware system. Additionally, the term context-awareness is limited to mobile systems.

Ceri [CDMF07] describes context-awareness as an

"evolution of personalization (user identity, preferences), that includes user environment".

Ceri's definition implies that personalization is a subset of context-awareness. Hence, context-awareness is not only of advantage in mobile environments but also in traditional desktop environments as it evolves from personalization. This statement is partly contrary to the above described definition of Kotz and Chen [CK00]. It is clear that the impact of context exists in desktop environments. The customization of application by using user preferences, for instance, can be of great advantage for the user. However, the impact is larger in mobile environments where context changes more rapidly and manual adaptations because of context changes are no longer manageable.

In our work the term context-awareness will be used as follows:

Definition 4 (Context-awareness) . Context-awareness is the property of a system, that:

discovers context and context changes The discovery can be an automatic process of sensing context (including the retrieval of previously stored context data or log data) or a manual process, where the user inputs relevant data. Context is a moving target. Also relatively stable kinds of context such as family names can change. Therefore, context changes have to be discovered in convenient time periods. The convenience of time periods is application dependent.

processes context data The system stores, eventually cleans, aggregates and generalize the context data. Thus, the system can easily analyze context and take advantage of context information.

takes advantage of context New and updated context can be presented to the user (*passive context-awareness*) or applications can be automatically adapted to context by changing the application's behavior (*active context-awareness*). Passive context-awareness requires user activity to adapt the application, for instance, by selection of context from a selection lists presented by the system. The selection triggers the adaptation of the application. In active context-aware systems the user stays passive in the adaptation process (see also [CK00]).

□

After giving a definition of context and context-awareness it is necessary to state that the implementation of a context-aware application is still a big challenge. The decision which context to use is of big importance and can limit the applicability of an application. It would be better to adapt the human to human interaction by a learning component. Such a learning component would learn to capture only that context important to the application and it would learn how to react to context changes in a proper way. Furthermore, such an approach would imply the usage of different kinds of context in different situations of an application. But this is an extremely challenging approach that requires interdisciplinary knowledge of, for instance, human perception and machine learning. Such a learning approach is out of the scope of this work.

2.3 Workflow management

The workflow concept is the central concept in this thesis. Workflows in general are flows of progress or work done by a company, industry, department or person [Dic09]. The term workflow, as used in this thesis, is the computerized flow of work. That is, not a person or a group of persons takes over the responsibility for workflow management, but a software. Workflows realize two main principles:

i) structuring of work and

ii) automation of work.

Workflows structure work in functional non-overlapping steps or building blocks. They define a (partial) order of these steps and specify who (person, role) or what tool has to fulfill which step. This structuring eases the coordination effort as processes are better understandable — load can be better shared among resources and personnel, a better utilized capacity can be achieved, personnel can be assigned to that step they are best suited for. Additionally, the structuring of work produces intermediate results after every step of work. This eases controlling of process execution.

A workflow automates work, as it is a computerized flow of work. That is, a software takes over workflow management. For its work the software needs as input a (partial) description of the workflow and executes according to that plan. This software is called workflow management system (WfMS) and offers tools for the definition, execution and monitoring of workflows. Thereby, WfMSs are not concerned with the execution of single tasks but handle the transition between the tasks. The work is delegated to external applications or persons. The execution state of a workflow virtually represents the actual state of work. Therefore, the automation does not only ease coordination and frees people from routine tasks, but it is also the precondition for an automated controlling — monitoring, error detection, error signaling, error resolving. The tools for automated controlling are also offered by WfMSs. Thus, workflows and WfMSs ease the coordination and controlling of work and are means to improve efficiency of work execution.

This make them especially beneficial for large and long running processes with many people involved in their execution. For these processes the coordination and controlling effort is especially high and costs for expensive and resource consuming WfMSs can be fast amortized.

Furthermore, the way workflows are handled by WfMSs — structuring of work in building blocks with a predefined (partially) order of execution, description and execution according to that description — make WfMSs especially suitable for the management of highly structured and often repeated workflows that are rarely subject to changes. For highly structured workflows it is a good way to predefine the order of execution because checks for correctness of workflows can be done in advance and will not disturb the maybe critical execution. For these workflows it is not only no problem but a demand that the workflow execution is rigid and cannot deviate from the workflow definition because such an approach avoids errors. Today's WfMSs are suitable for often repeated workflows because the time-consuming definition of workflows especially pays off when the workflow is executed several times. Additionally, today's WfMSs are appropriate for managing rarely changed workflows as modeling concepts for the modeling of expected changes exist, but with an increasing number of exceptions the workflows loses clarity. Unexpected changes cannot be handled.

Despite the advantages of workflows and WfMSs, the functionality, today's WfMSs offer — coordination and controlling of work — is no longer sufficient to support WfMS users appropriately. Laws and prescription are subject to changes. Companies try to improve their processes frequently. Therefore, workflows have to be changed frequently and WfMSs have to be able to support these frequent changes. Additionally, there is a trend to individual product solutions, product customization and flexible reactions to changes of the mobile environment. WfMSs have to handle rarely repeated workflows and have to support the flexible reaction to customer wishes and context changes. A WfMS has to offer strategies to manage exceptions that were not expected, that is, exceptions that were not pre-defined. And there is the request for WfMSs that are able to handle less structured workflows. Thus, today WfMSs have to handle more flexible, unstructured and rarely repeated workflows. To handle these kinds of workflows is the objective of this thesis. A challenge of this work is the fact that the need for controlling is essential and remains. Unfortunately, controlling and flexibility are conflicting goals [Nar04]. Thus, while focusing in this work on flexibility of workflows, we have to deal with balancing controlling and flexibility.

In this section we aim to explain the notion of workflows as well as terms relevant to this topic, in detail. We will explain how workflows are handled in traditional WfMSs. Additionally, the strength and weaknesses of the conventional WfMSs are shown, and it is explained why conventional WfMSs fail to meet today's requirements of dynamic workflow management. Subsequently, we introduce several different approaches of dynamic workflow management that aim to overcome the weaknesses of traditional WfMSs. The main focus of this section is on dynamic workflow management.

2.3.1 Workflows

Workflows are abstractions of real world processes. They define what has to be done in which order by whom or by which application. Workflows abstract from details not needed for process execution. WfMSs offer all tools needed for the handling and automated execution of workflows: tools for the definition, execution and monitoring of workflows. That is, workflows and WfMSs are means to structure and automate work and therefore improve work efficiency.

Workflows and their management are the core topics of this thesis. To discuss the problems and current challenges of workflow management, we have to use a precise and unambiguous base of workflow terminology and definitions. Unfortunately, although workflow technology is not a new technology, workflow terminology and definitions lacks a commonly used base and are often used conflictingly, that is, terms are homonymously and synonymously used. Therefore, in this section we introduce the workflow terminology used in this thesis in detail. Additionally, we give a detailed and formal introduction to workflows. We refer to the work of the Workflow Management Coalition (WfMC)[4]. The WfMC aimed to provide a standardized basis for workflow terminology and definitions. But we differ in naming when their naming is not commonly used and we aim to give more detailed and formal definitions than they do.

The most important term that has to be defined in this section is the term *workflow*. This term is strongly related with the term *business process* since workflows automate business processes.

Definition 5 (Business Process (WfMC)) . A business process represents a real world process. Work to be done by this process is structured into logical steps. The linkage of the steps represents the order of work execution. The steps are realized to achieve a common objective. The outputs after process completion are defined. □

Business processes are real world processes. An example for such a process is the *Insulin injection of a diabetic before a meal* (see Figure 2.3), a business process with the goal to find the right dose for an insulin injection before a meal. Business processes are structured as sequences of logical steps of work, that are executed one after the other or in parallel to reach a common goal. The steps are called tasks or activities.

Definition 6 (Task) . A Task is a logical step of work in a process. This step can require manual intervention for completion of process execution or it is fully automated. □

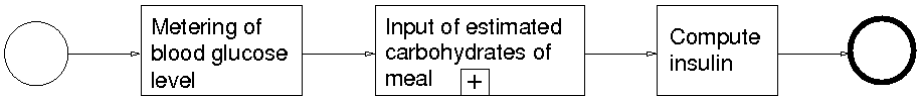
Tasks are units of work. Tasks are manually executed by humans or automated by computer programs. For example, the business process *Compute insulin for injection before each meal* (see Figure 2.3(a)) consists of the tasks (i) *Metering of blood glucose level*, (ii) *Input of estimated carbohydrates of a meal* and (iii) *Compute insulin*. Tasks are atomic units such as task (i) and (iii)

[4]The WfMC is an organization of companies and research groups engaged in workflows and business process management (BPM), founded in 1993. The WfMC defines standards and reference models in the area of workflow management and BPM.

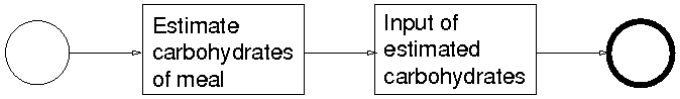
or in turn processes, so called sub processes, such as task (ii). Sub processes are processes that are initiated by another process.

Definition 7 (Sub process) . A sub processes is a business process that is part of a larger business process, the initiating process. The initiating process calls the sub process during process execution. Sub processes can act as initiating processes of other sub processes. □

Sub processes are described as business processes that belong to or are part of larger business processes, the so called initiating processes. Sub processes, described separately from the definition of the initiating process, are reusable components and may be applied by other initiating business processes. In contrast, embedded sub processes are dependent of the larger process and cannot be reused.



(a) Process - Compute insulin for injection before each meal. This process is the initiating process for (b). The +-symbol of the second step denotes a sub process.



(b) Sub process - Input of estimated carbohydrates of a meal. This is the sub process of process (a).

Figure 2.3 – Process - Injection of insulin before each meal for a simplified diabetes example (according to Intensified Conventional Therapy (ICT)). Process (a) and sub process (b) are depicted.

During life time a business processes is several times enacted. A single enactment is called process instance. The initiation of new process instances is triggered by conditions, such as the wish of a diabetic to eat something. The outputs after process completion are defined, e.g., the dose of insulin to inject. Thereby, the output value can differ for every executed process instance. When business processes are automated, they are called workflows:

Definition 8 (Workflow) . Workflows are the computerized facilitation or automation of a business process, in whole or part [Hol95]. A workflow W is defined as $W = (T, <)$, where:

- T is a finite set of tasks.
- $<$ is a partial order on T (control flow).

□

Definition 9 (Workflow graph) . A Workflow (W) can be represented by a directed graph, the workflow graph (WFG). The WFG of workflow W, $WFG(W)$, consist of nodes (N) and directed edges (E) that connect the nodes. There are following kinds of nodes [Nar04]:

- *work nodes* (N_{Work}) that represent the tasks of a workflow, that is: f: $N_{Work} \rightarrow W.T$ is a bijection (one-to-one correspondence). Therefore, the cardinality of the work nodes is equal to the cardinality of the tasks of a workflow ($card(N_{Work}) = card(W.T)$).
- *route notes* (N_{Route}) that evaluate boolean conditions to decide about the direction of control flow to a specific work node or route node.
- one *start node* (n_{Start}) that represents the start of a workflow, that is, the start node has no predecessor ($\nexists n \in N \setminus \{n_{Start}\} : (n, n_{Start}) \in E$) and
- one *end node* (n_{End}) that represents the end of a workflow, that is, the end node has no successor ($\nexists n \in N \setminus \{n_{End}\} : (n_{End}, n) \in E$) .

Thus, a workflow graph is a two tuple $WFG(W) = (N, E)$, where:

- N is a finite set of nodes, with $N = N_{Work} \cup N_{Route} \cup n_{Start} \cup n_{End}$.
- E is a finite set of edges, where $E \subseteq (N \times N)$.
  - In case of a *sequence of nodes* the partial order $<$ over the elements of $W.T$ (and therefore N_{Work}) is represented by an edge E that connects two work nodes. That is:
    $t_i < t_k \Leftrightarrow (n_{Work_i}, n_{Work_k})\ in\ E$, whereby t_i is represented by n_{Work_i} and t_k is represented by n_{Work_k}.
  - In case of *non sequential routing* the partial order $<$ over the elements of $W.T$ (and therefore N_{Work}) is represented by edges E incoming and outgoing to a route node n_{Route_r}. Thereby, more than two work nodes are connected via edges incoming and outgoing to a route node. To describe the partial order in case of non sequential routing it is sufficient to consider pairs of two work nodes, one node is connected via an incoming edge with the route node and the other is connected via an outgoing edge with the route node. That is:
    $t_i < t_k \Leftrightarrow \exists\ n_{Route_r} \in N_{Route} : (n_{Work_i}, n_{Route_r}) \in E \ \wedge\ (n_{Route_r}, n_{Work_k}) \in E$, whereby t_i is represented by n_{Work_i} and t_k is represented by n_{Work_k}.

There are two conditions a workflow graph has to obey [Nar04]:

- There is a path from the start node n_{Start} to every other node in the graph.
- There is a path to the end node n_{End} from every other node in the graph.

□

Workflows are designed to automate business processes. For workflow specification workflow languages are used. Workflow languages are computer understandable languages. The

workflow description in such a language is the precondition for automated process execution. Naturally, the concepts underlying business processes are strongly related to that concepts underlying workflows and both concepts follow the same basic idea. Workflows are also represented by a set of partially ordered tasks. The tasks represent logical units of work, that have to be executed to reach a common goal. Tasks are executed manually by users or automated by applications.

Figure 2.3(a) shows a graphical representation of a workflow. This figure is, as all figures in this section, depicted in the Business Process Model and Notation Version 2.0 (BPMN2).[5] In Figure 2.3(a) we see a directed graph that represents a workflow, with nodes that represent tasks, and directed edges that represent the order in which the tasks are executed. Every workflow starts with a start node and finishes with an end node. Start node and end node are nodes without equivalent in the workflow definition. They have been added in the workflow graph to signal or visual mark start and end. On the other hand, these nodes can be important for (distributed) workflows that have to synchronize their execution (e.g. synchronized initialization or end).

The order of task execution, also known as control flow of workflows, can be described by Control Flow Patterns delineated by the Workflow Patterns Initiative [vdAtHR11], [vdAtHKB03], [RHM06][6]. The so called Basic Control Flow Patterns are fundamental building blocks of workflows that capture elementary aspects of process control. These basic patterns are available in all current WfMSs. [vdAtHKB03] Because we use the Basic Control Flow Patterns frequently in our figures, we show an overview of them in Figure 2.4 and describe them briefly[7]:

Sequence (see Figure 2.4(a)) is a pattern which describes that a task is enabled after the completion of the preceding task. Tasks are executed one after the other.

Parallel Split (see Figure 2.4(b)) is a pattern that describes the divergence of a branch into parallel branches whereby each of them executes concurrently. This pattern depicts the concurrent execution of tasks.

Exclusive choice (see Figure 2.4(c)) is the basic pattern describing the divergence of a branch into multiple branches. Once the incoming branch is enabled, the thread of control is immediately passed to one of the outgoing branches based on a selection mechanism. This pattern is also known as conditional routing. Variables determine the routing.

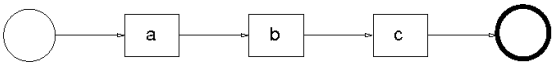
[5] We use BPMN2, maintained by the Object Management Group (OMG), for several reasons: BPMN2 is (i) a standard for business process modeling, (ii) it is a visual modeling language and (iii) it aims to be a notation that is understandable by all business users — business analysts, technical developers, business people. BPMN2 bridges the gap between the business process design and process implementation [OMG11].

[6] The Workflow Patterns Initiative describes recurring fundamental requirements, that arise during business process modeling, in an imperative way as patterns. The patterns are called Workflow Patterns and cover besides control flow patterns, workflow data patterns, workflow resource patterns and exception handling patterns.

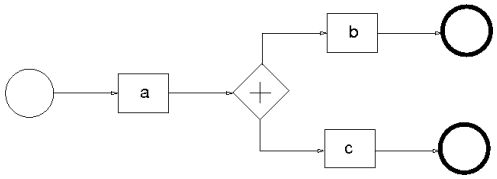
[7] The WfMC also describes these patterns and label them differently. We decided to refer to the presentation of the Workflow Patterns Initiative because of the more precise description.

Synchronization (see Figure 2.4(d)) is a pattern, where multiple branches converge into a single subsequent branch. The thread of control is passed to the subsequent branch when all input branches have been enabled.

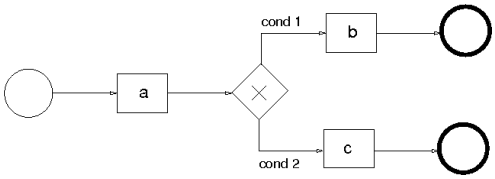
Simple merge (see Figure 2.4(e)) is a pattern where multiple branches converge into a single subsequent branch. Each activation of an incoming branch results in the thread of control being passed to the subsequent branch.



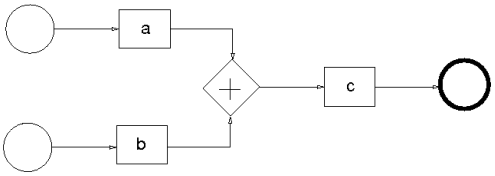
(a) Sequence: a task is enabled after the completion of a preceding task.



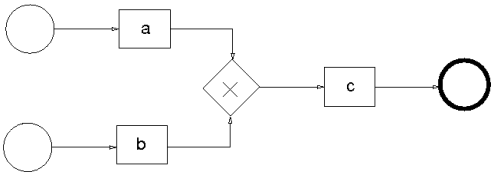
(b) Parallel split: split of a branch into parallel branches which execute concurrently.



(c) Exclusive choice: split of a branch into multiple branches. Is the incoming branch enabled, the thread of control is immediately passed to one of the outgoing branches based on a selection mechanism.



(d) Synchronization: convergence of multiple branches into a single subsequent branch. The thread of control is passed to the subsequent branch when all input branches have been enabled.



(e) Simple merge: convergence of multiple branches into a single subsequent branch. Each activation of an incoming branch results in the thread of control being passed to the subsequent branch.

Figure 2.4 – Basic Control Flow Patterns [RHM06].

Beside the Basic Control Flow Patterns, the Workflow patterns initiative describes Advanced Branching and Synchronization Patterns. As the name implies, the advanced patterns describe more complex branching and synchronization concepts. The by these patterns described concepts are in practice commonly used but not always directly supported by WfMSs. The explanation of that pattern is out of scope of this thesis. For further reading see [vdAtHR11].

2.3.2 Managing static workflows

WfMSs provide tools for computerized definition, execution, and controlling of workflows. One of the most important characteristic of a WfMS is, that its run time components (Workflow Enactment Service and Admin & Monitoring Tools) does not focus on the execution of individual tasks, but focus on the process itself — the flow of work through an organization, its monitoring and execution. The responsibility of the execution of a task is delegated to applications or people. Work is decoupled, and pieces of work to be done become independent of the workflow and independent of each other (see [vdA98]). Therefore, programming of workflows can be seen as *programming in the large*. The term programming in the large was coined by DeRemer and Kron in [DK76]. They proposed to make a distinction between the programming of components — programming in the small — and the programming with components — programming in the large. Programming in the small describes the traditional way of (structured) programming. Programming in large describes the structuring and integration of large collections of reusable components to compose a larger system. Workflows integrate different applications which share the same application type or belong to different classes of applications, such as Desktop Applications or Web Services. Each of these applications implements a task of the workflow.

Definition 10 (workflow management system) . A workflow management system is the software that is concerned with the definition, controlling and execution of workflows. □

The WfMC [Hol95] described a reference model of WfMSs. This reference model is a specification for the implementation of WfMSs and should enable the interoperability between WfMSs of different vendors as well as it aims to improve the ability of workflow applications to interact with other applications such as email and document management [Hol95]. In the following sections we will discuss two views on the reference model. Firstly, we will have a look at the components and interfaces of the workflow reference model. Afterwards, we will give a detailed description of the functionality a WfMS offers to build and execute workflows.

Workflow reference model: Components and interfaces a WfMS has to offer

According to the workflow reference model a WfMS should offer well-defined interfaces that allow interoperability between the components that constitute the WfMS. Furthermore, a WfMS should offer well-defined interfaces between the component responsible for the enactment of the workflows and components outside the scope of the WfMS, that is, the enactment components of other WfMSs, applications invoked by the enactment service to execute tasks, and workflow client software. Such an architecture enables the compatibility of components. Consequentially it offers the possibility to freely combine components of different vendors to customize the WfMS. Therefore, WfMSs that follow the reference architecture defined by the WfMC are vendor independent.

In the following section, we will have a detailed view on the components defined in the workflow reference model. Additionally, Figure 2.5 shows the workflow reference model and distinguishes internal modules (Definition, Enactment, Admin & Monitoring), from external modules outside the control of the WfMSs (Client Applications, Invoked Applications, Enactment of other WfMSs).

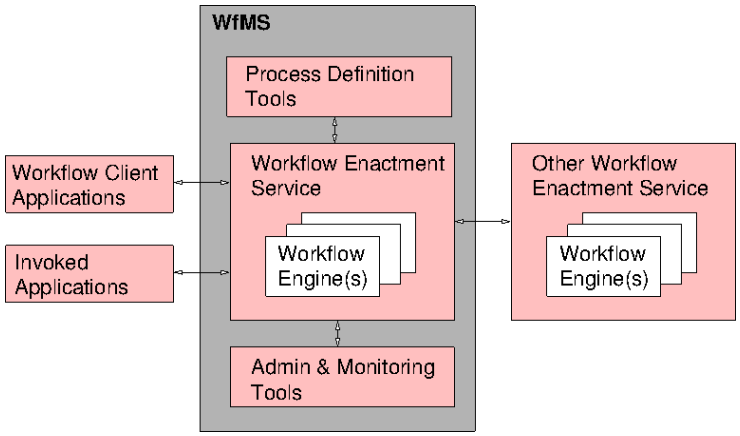


Figure 2.5 – WfMS components and interfaces [Hol95]

Process Definition Tools allow users to define workflows or more precise: to create process definitions. WfMSs often offer graphical modeling tools for process definition since WfMS users are usually domain experts with no programming experience. The graphical tools are more intuitive and help to avoid modeling errors.

The *Workflow Enactment Service* is the core of the WfMS. It contains one or more Workflow Engines that execute the workflow definitions, specified with the Process Definition Tools at run time. This module offers well-defined interfaces to all other modules of the reference model.

Admin and Monitoring tools control workflow execution conducted by the Workflow Enactment Service. While the Administration tools attend to security, control and authorization, the monitoring tools view the status of work flowing through the organization.

Workflow client applications are applications, that work on the workflows managed by the Workflow Enactment Service. These applications offer an interface for end users to interact with the WfMS when manual interaction is required to perform tasks.

Invoked applications are applications, that are invoked by the WfMS to execute tasks of the workflow. The communication between invoked application and WfMS covers functionality such as notification of task completion by the application and data passing to and from the application.

Other workflow enactment services are workflow enactment services of other WfMSs. The interface between enactment services of different WfMSs is intended to allow a work passing

from one WfMS to the other WfMS. This work passing ranges from simple task passing to a full workflow application interoperability with a complete interchange of process definition, workflow relevant data and a common look and feel.

Workflow reference model: Functionality a WfMS has to offer

In addition to the component view described before, we depict a view of the functionality of WfMSs to understand the design proposed by the reference model and to understand the interplay of the components. A WFMS works according to the workflow reference model at three levels (see Figure 2.6):

Process Design: At build time the workflow is designed and defined. The design phase starts with the analysis of the real world business process. The analysis serves as the basis for the process design. WFMS usually offer graphical user interfaces to support process design. The graphically designed business process is translated into a formal, computer processable definition [Hol95]. The WfMC [Hol95] describes that WfMSs may allow changes to the workflow definition from the run time operational environment. But this functionality is rarely realized in WfMSs.

Process Instantiation and Control: The workflow definition is interpreted at run time. A start event triggers the creation of a workflow instance by the workflow enactment service. More precisely, the workflow enactment service consists of one or more workflow engines that are concerned with the creation of workflow instances, and additionally, manage and execute workflow instances [Hol95]. This includes task scheduling within an operational process and interaction with application tools and human resources. Workflow engines have to deal with processes, operation over a wide geographical basis and are therefore often distributed across a number of computer platforms.

Interaction: The control software of WfMSs, the workflow engines, interacts at run time with human users and IT applications to process task instances. The interaction covers, among others, passing control between tasks instances, determination of the process status, invocation of application tools and passing of appropriate data [Hol95]. Users interacting with the WfMS maybe use other software tools to process their tasks.

The description of the functionality of the WfMS shows, that an important concept applied by WfMSs is the concept of build time modeling and run time execution. At build time the workflow definition is modeled, can be analyzed and simulated [EKR95]. At run time several workflow instances can be created, executed and monitored. It is important to point out that the workflow definition is also known in literature as workflow specification, workflow model, or workflow schema. The terms workflow and workflow definition are used synonymously. The workflow instance is often referred to as case.

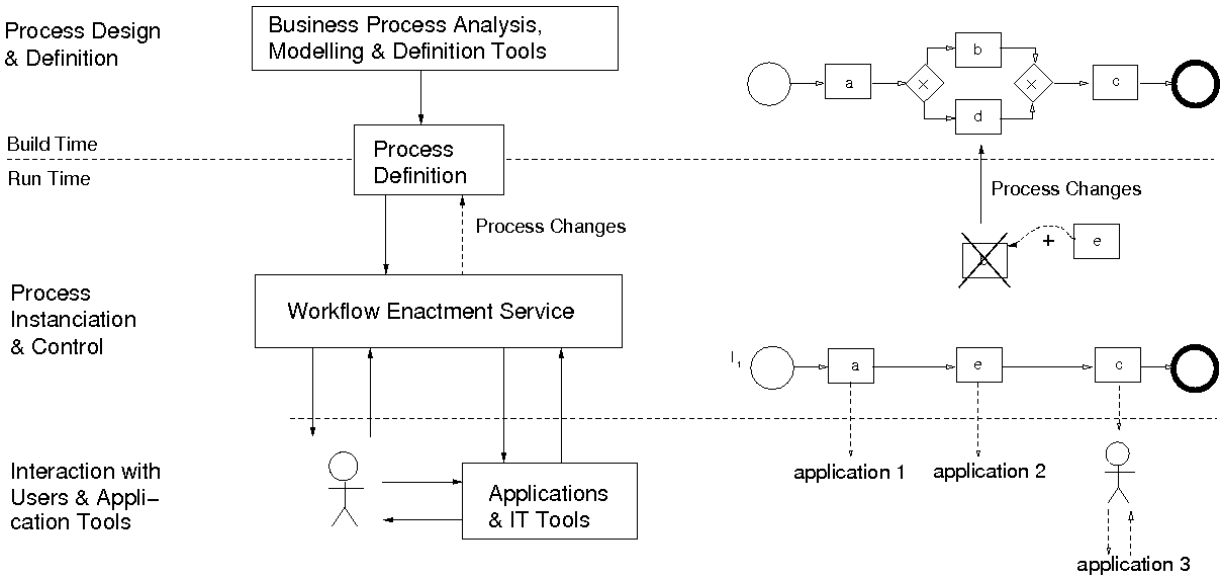


Figure 2.6 – WfMS characteristics as described in [Hol95] decorated by an example process: At build time the workflow process is defined. At run time the process is instantiated and controlled. Some WfMSs may allow dynamic changes to the process definition (replacement of task b by task e in the example). During enactment, WfMSs interact with users or applications that execute various tasks.

Definition 11 (Workflow definition) . The definition of a workflow (see Definition 8) is created at build time. Thereby, the tasks of the process, information about these tasks, such as associated data, start and termination criteria of the process as well as the relationships of the tasks are specified. The workflow definition describes all possible execution paths a workflow instance can take at run time. A workflow is usually represented by a workflow graph (see Definition 9). □

Definition 12 (Workflow instance) . A Workflow instance that is based on the workflow definition, is created, executed and eventually terminated by one (or more) workflow engines at run time. Each workflow instance uses its own data and has an individual behavior. Task instances represent tasks at run time. In workflow definitions the pattern exclusive choice is used to define alternative paths of routing a instance can take at run time. At run time a workflow instance always follows exactly one of these execution paths. During execution, a workflow instance has a defined state. The state is characterized by already executed task instances, just executing task instances and task instances still to be executed. According to [Hol95] there are following basic states of a workflow instance:

- *initiated*: the workflow instance is created but not yet started.
- *running*: the workflow instance has started execution and any of its tasks may be started.
- *active*: one or more of the tasks of the instance have been started (one or more task

instances exist).

- *suspended*: the workflow instance is quiescent. No tasks will be started until the workflow gets a resume command and returns to the state running.
- *completed*: the workflow instance is completed and will be destroyed.
- *terminated*: the workflow instance stopped before its normal completion and is destroyed.

□

Figure 2.7 visualizes the execution states of two different workflow instances (I_1, I_2) that share a common workflow definition. Both workflow instances have been initiated, and are neither suspended nor terminated or finished. I_1 has executed task a and is executing task b, I_2 is executing task a. Thus, the state of both is active and not only running.

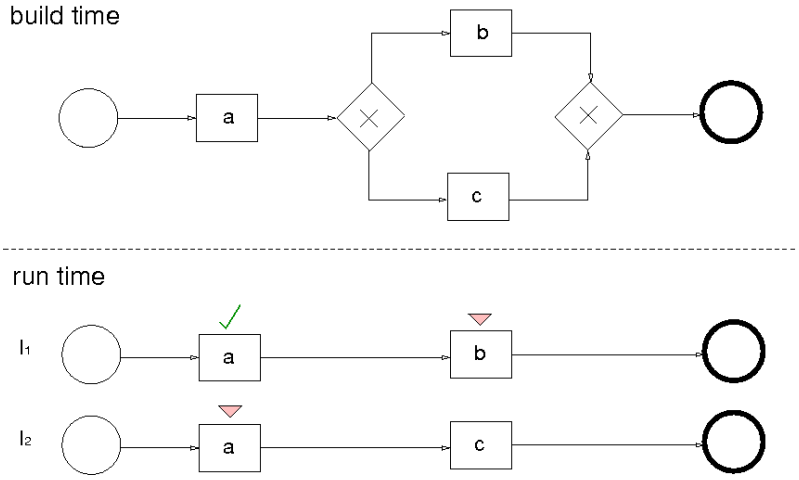


Figure 2.7 – At build time different execution paths are modeled. At run time workflow instances (I_1, I_2) follow exactly one path. I_1 executes b, a is already executed. I_2 executes a, c is still to be executed.

2.3.3 Strengths of the usage of workflows in WfMSs

As described before, workflows are (partly) automated business processes. Before we describe the advantages of workflows managed by WfMSs, we will point out, that already the specifying of a business process has many advantages. A process specification defines which work is executed in which order by whom. That is, it captures process knowledge, it ensures that the process matches the process requirements and it documents the process. Knowledge about the process is disclosed and no longer only available in the head of some experts. Knowledge capturing and documentation can trigger business process optimization and business process

improvement activities. Working with the process, visualizing and thinking about the process are preconditions for the detection of errors and conflicts.

Managing workflows by a WfMS enables process automation. Thereby, the management system automates the process itself and passes the work to people or computer applications. The advantages of automated over non automated business process management can be described as follows:

- *Computer aided process specification:* The WfMS allows for a specification of a workflow in a computer understandable form. This allows for a computer aided simulation, execution, administration and monitoring of the workflow.
- *Computer aided process execution:* The workflow execution by a WfMS according to a predefined workflow definition avoids that process steps are forgotten, that process steps are executed in a wrong order, and that the same work is done several times. It ensures that work is executed in the right order by the right person at the right time.
- *Computer aided process monitoring:* Problems while process execution are detected and can be easily identified by the monitoring tools. It is no longer possible, that deadlines are exceeded because the agent in charge is sick. Additionally, some systems have the ability to take measures to solve the problems.

Beside the advantages over non automated business process management, workflow management proceeded by a WfMS has advantages over workflow management done by means of scripts or done directly in the application code. Important advantages of workflow management by WfMSs are:

- *Process logic and task logic are decoupled.* This decoupling of process and task makes the execution of the tasks independent of the process logic and it makes tasks independent of each other.
- *Data flow is separated from control flow.* State and location of data can be determined during workflow execution.
- *Build time modeling is separated from run time execution.* This allows for a more efficient run time execution.
- *Monitoring/Tracking of process execution of process instances.* The separation of concerns described before allow for monitoring of process execution.
- *Integration and coordination of heterogeneous applications as well as manual and automated activities.* Existing software can be reused and has not to be rewritten. For the users a holistic view on business processes is offered.

- *Modeling tools and languages are convenient for domain experts.* Workflow management systems usually offer easy to understand workflow modeling languages and often graphical modeling tools. Hence, modeling is supported in a way suitable for domain experts with no software programming skills.
- *Workflow management offers a lot of additional functionality.* The software offers not only the modeling, execution and tracking of workflows but also provides functionality such as error handling and automated enactment of workflows.

2.3.4 Weaknesses of the usage of workflows in WfMSs

Despite the large number of advantages, the usage of workflow technology offers, there are some disadvantages with a high impact on their usability:

- *Large run time system:* The large run time system enlarges applications that build up on workflows. An application developer has to balance whether the underlying process logic justifies the use of a workflow system. This consideration is especially important for mobile applications, where application size and footprint are critical parameters.
- *Inflexibility:* Traditionally, workflows are rigid plans predefined at build time and executed as predefined at run time. This is done to allow for a more efficient execution (see Section 2.3.3). This strategy is problematic when unexpected events occur that require changes to workflow instances. Unexpected events that require changes to all workflow instances of a workflow definition, entail a change of the workflow definition. When the system has no tools to handle these changes flexibly, running workflow instances are suspended or aborted to avoid undesirable side effects. Afterwards changes to the workflow definition can be applied. [EKR95] The handling of unexpected events that require changes of only some workflow instances is not foreseen in traditional WfMSs. Thus, users try to bypass the system for the affected workflow instances. The exception handling is not documented and cannot easily be reused. Inconsistencies of the workflow instance caused by such an exception handling cannot be detected.
- *Confusing presentation:* When a workflow has a lot of known alternative execution paths (expected changes), the presentation of the workflow becomes more confusing and less readable with a growing number of alternatives. The workflow definition explodes [KLB+08].

2.3.5 Discussion of static workflows

The separation in build time definition and run time execution allows a process execution in line with the workflow definition and prohibits unwanted changes in workflow instances. To react on events that require structural adaptations of the workflow, traditional WfMSs offer conditional

routing. This control flow pattern allows different routing of work for one workflow definition. That is, every workflow instance follows its own execution path, depending on different conditions (see Figure 2.7). The approach of conditional routing is not flexible enough to handle events that require changes to workflow instances and are unforeseen or foreseen but the time of their occurrence is not known. Originally, workflow management systems were developed to enable process automation of business processes. Business processes were traditionally highly structured and long living. Once designed, these processes were valid for a long time span. That is, a small number of workflow definitions were defined and a high number of instantiations were created of them. Thereby, the specification was made in a very prescriptive way. There was little scope left to the users and it was tried to capture every possible execution path [KLB+08]. Traditional WfMSs work well for these processes, they are designed for. But today, business processes are subject to a lot of changes resulting from the necessity to improve products and business processes to stay competitive on the market. Therefore, using traditional WfMSs, a high number of workflow definitions is created and only a view times instantiated. Additionally, business processes are less structured because of business trends like business-on-demand and in general the trend to offer an increasing product and service variability.

In summary, workflows have to be much more flexible today, as they are applied to less structured and less predictable processes. For traditional WfMSs with their rigidity caused in build time modeling and run time execution, it is difficult to reach the needed degree of flexibility. With this work we aim to develop an approach that overcomes the inflexibility of traditional WfMSs and makes WfMSs more flexible regarding the structural adaption to foreseen or unforeseen events. A way to deal with this problem is to apply dynamic workflows instead of conventional workflows. Additionally, this kind of workflows avoid confusing presentations as they do not apply conditional routing to offer a higher flexibility. But they also usually need a large run time system. Dynamic workflows are described in detail in the following section.

2.4 Managing dynamic Workflows

As explained before, dynamic workflows offer the possibility to deal with evolutionary and temporary structural changes of workflows. This capability of dynamic workflows is getting more and more important as business processes are subject of frequent modifications and workflows are used for less structured scenarios today, characterized by foreseen or unforeseen events that lead to changes of workflows.

There are different kinds of dynamic workflows. Sadiq et al. [SSO01] classify dynamic workflows in dynamic, adaptive, and flexible workflows. In our opinion, this denotation is not intuitive as the words dynamic and flexible can be used synonymously and all dynamic changes are adaptations to occurring events. Han et al. [SHB98] classify the same concepts in meta-model approaches (dynamic and adaptive workflows for Sadiq) and open-point approaches (flexible

workflows for Sadiq). The term meta-model approaches combines two completely different classes of dynamic workflows in one class. For a better understanding these classes should not be combined. In contrast, the term open-point approach is easy to understand and clear. Weber et al. [WRR07] classify flexibility in process-aware systems (PAIS)[8] and use the terms process adaptation (dynamic and adaptive workflows) and built-in flexibility (flexible workflows). Additionally, they denote dynamic workflows as schema evolution and adaptive workflows as ad-hoc changes. The definition of built-in flexibility is a superset of the concepts covered by flexible workflows. As also built-in flexibility is an adaptation of the process to occurring events, the term process adaptation is in our opinion not sufficiently self-describing. For us it is important to point out that the three levels described in the subsection about the functionality of the WFMS (see Section 2.3.2) guide us in making a WFMSs flexible. A WFMS can be made flexible on all three levels: At the workflow definition level, the workflow instance level and the interaction level. Whereby, in the strict sense, making the execution of tasks flexible is out of the scope of a WFMS.
Because the above introduced terminology for the types of workflow flexibility is not always clear, we defined our own one. Essentially, with this terminology we aim to be precise and intuitive. Thus, we decided to use the terms *schema evolution* (dynamic workflows), *ad-hoc changes* (adaptive workflows), and *partly defined workflows* (flexible workflows plus concepts introduced additionally in built-in flexibility). In the following we describe our understanding of this classification and refer to the explanations of [SSO01], [SHB98], and [WRR07].

2.4.1 Schema evolution

Schema evolution are evolutionary changes of the workflow definition[9] resulting from real world changes such as process improvements, innovations, and reengineering [SSO01]. These modifications are long term changes, that do not affect individual cases but all process instances of a process definition (see Figure 2.8). Therefore, the modifications have to be applied on process type/process definition level [SSO01].

According to Casati et al. [CCPP96] schema evolution consists of two parts: i) the modification of the workflow definition, and ii) the management of running instances. The WfMS has to provide tools for a refinement of the workflow definition without having to rewrite the workflow from the scratch and it must guarantee the syntactically correctness of the new version of the workflow. A more difficult issue is the management of running instances. Casati et al. describe three main policies for handling the changes of running instances:

- *Abort* of all workflow instances following the old workflow definition and start all new workflow instances according to the new workflow definition. This implies a loss of useful work since existing instances are aborted.

[8] Process-aware systems are a super class of workflow management systems.
[9] As mentioned before, workflow schema is another name for workflow definition.

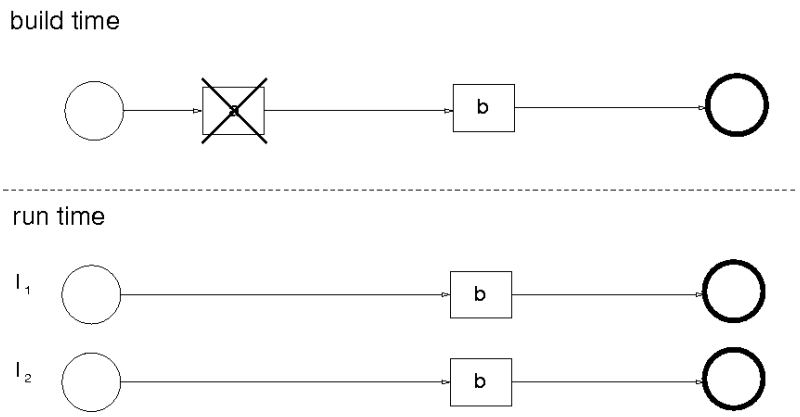


Figure 2.8 – Schema evolution: build time changes (removal of task a) affect all instances. Two instances that were started after schema modification are shown. What happens to running instances depends on the change policy.

- *Flush* means that it is waited until all existing instances belonging to the old workflow definition are terminated. Only after the termination of all running instances, new instances, following a new definition, can be started. This policy does not imply a loss of work but of productivity because of the waiting period. In some cases this policy cannot be applied, e.g., when a law is changed, offenses against the new law are not tolerated.

- *Progressive* is a policy that allows for making different decisions for different instances. The decision is made according to instance state or history. For instance, such policies allow for concurrent completions of workflow instances acting according to different workflow versions, which avoids the latency of the flush policy. Another example is the migration of running instances to the new workflow version. Whether a migration of the workflow instance is useful depends on the compliance of the versions of the workflow definition and the state of the workflow instance execution. For further readings see [CCPP96].

2.4.2 Ad-hoc changes

Ad-hoc changes are short term changes, that are needed to react fast and convenient on known and unknown exceptions or events, e.g., a diabetic is allergic to insulin or concomitant substances, or a female diabetic conceives a child. The specific events effect only one or maybe few instances of a workflow definition. Thus, the changes have to be done on instance level. If the same ad-hoc change has to be applied several times to instances of the same workflow definition, this can be an advice that this change is a evolutionary change. Such changes have to be handled by a schema evolution.

Typical change operations are adding, removing and swapping tasks. Thereby, it is important that the guarantees, checked at build time, are not violated and the system stays consistent. This can be done by pre- and postconditions for change operations. Additionally, the complexity

of adaptations of workflow instances such as re-mapping of input and output parameters should be hidden to a large degree to the users. [RRD03]

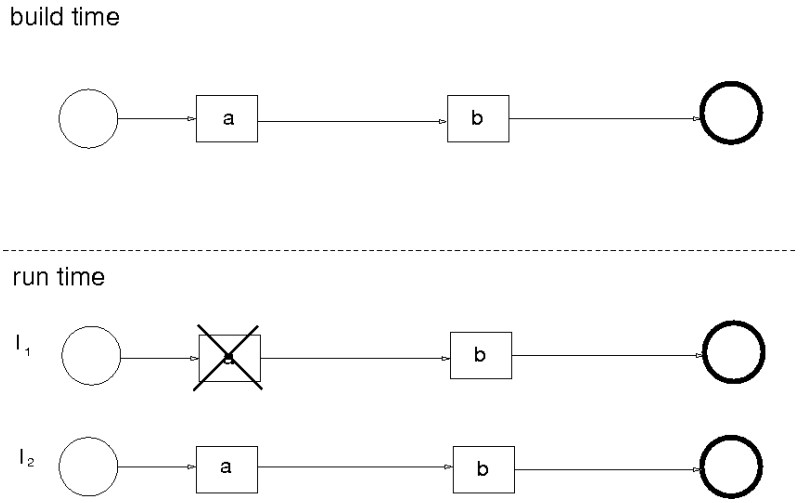


Figure 2.9 – Ad-hoc changes: a change of instance I_1 (removal of task a) does not influence other instances such as I_2 and the workflow definition is also not touched.

2.4.3 Partly defined workflows

The term partly defined workflows covers the (i) open-point approach, and (ii) further very different ideas that aim to avoid a complete specification of the workflow at build time to reach a higher flexibility of workflows at run time. That is, the term refers to underspecified workflows [KLB+08].

Open-point approach

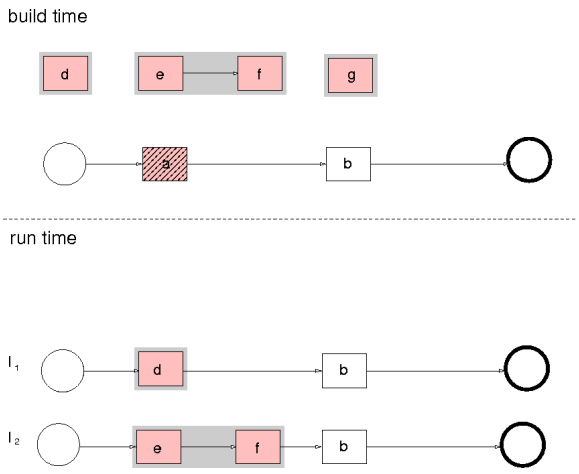
Open-point approaches offer the ability to execute workflow instances basing on partly defined workflow definitions, e.g., the patient visit in hospitals follows a routine but the treatment cannot be exactly pre-defined as it depends on too many different parameters (patient record, physical parameters, disease). At build time the workflow is defined as follows: Parts of the workflow unknown at build time are defined as open points (abstract tasks), parts known at build time are defined as normal tasks. At run time the model is completely defined and the open points are specified by including predefined (or at run time defined) fragments. Fragments are atomic tasks or sub processes composed of more than one task. Maybe there are no similar workflow instances of a workflow definition. [SSO01]

Weber et al. identified in [WRR07] four patterns for predefined changes. Two of them fit in the category open-point approach:

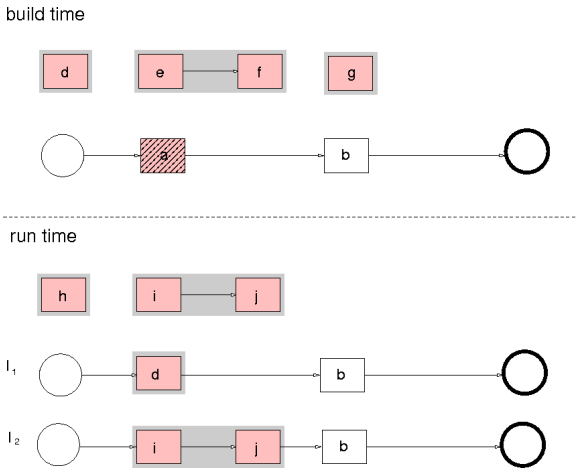
- *Late Selection of Process Fragments* is an approach, where at build time placeholders

(abstract tasks) mark points in the workflow definition for which several alternative implementations exist. All alternatives are known at build time and implemented as fragments. For the selection of the convenient implementation at run time, information which is only available at run time, is required. When the workflows are automatically executed predefined rules are used to select the implementation (see Figure 2.10(a)).

- *Late Modeling of Process Fragments* is an approach similar to Late Selection. Late modeling offers more flexibility compared to Late Selection. This approach is used when it is not possible to specify all alternative implementations in advance at build time. Late modeling allows for the modeling of fragments at run time. The only constraint is that a fragment has to be modeled before it can be used as choice for an open-point implementation (see Figure 2.10(b)).



(a) Late Selection: At build time predefined fragments (d, e-f, g) can replace the abstract task a at run time. The instances I_1 and I_2 depict two possible workflow instances.



(b) Late Modeling: At build time predefined fragments (d, e-f, g) and at run time defined fragments (h, i-j) can replace the abstract task a at run time. The instances I_1 and I_2 depict two possible workflow instances.

Figure 2.10 – Two variants of open-point approaches

Further approaches to improve workflow dynamic

In this section we describe approaches that base on the idea of incompletely defined workflows and that are not covered under the term open-point approach. Multi Instance Activity and Late Composition are two patterns for predefined changes (see [WRR07]) that do not fit into the category open-point approach. Generic Interface is an approach noted in [SHB98]. Multi Instance Activity is a very limited approach while Late Composition and Generic Interface are complex and powerful approaches.

- *Multi Instance Activity* is an approach used when a task has to be executed more than once and at build time it is not clear how often it has to be executed. Thus, at run time the number of created task instances is determined [WRR07] (see Figure 2.11).

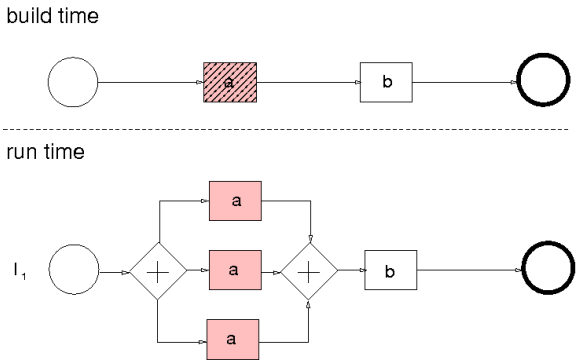


Figure 2.11 – Multi Instance Activity: several instances of a marked task (a) can be created at run time.

- *Late Composition of Process Fragments*[10] is an approach applied when there exist several possible compositions of process fragments. The approach is used to reduce the number of process definitions to be specified at build time. At build time process fragments are created. At run time the process fragments are dynamically selected and control dependencies are added on the fly to compose dynamically a process instance [WRR07] (see Figure 2.12). An automated late composition requires the formalization of process goals to

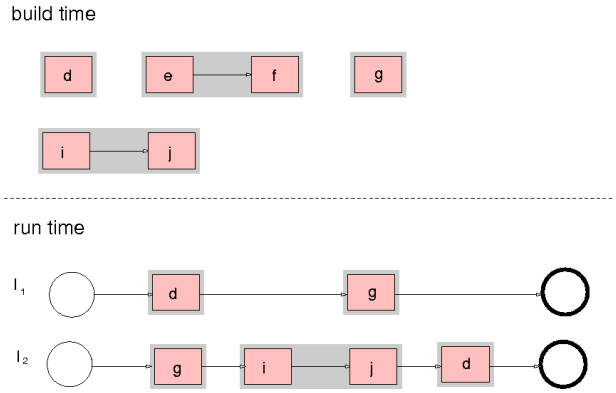


Figure 2.12 – Late Composition: At build time defined fragments are composed at run time. The instances I_1 and I_2 depict two possible workflow instances.

determine fragments and their execution order [GOR11]. Planning techniques are used to

[10] It is possible to see Late Composition as a special open-point approach where the partly defined workflow consists of start and end task and exclusively one open point (abstract task) and no other tasks.

implement this goal-based approach, that is, to dynamically adapt the workflow when the process goal changes or a failure occurs and the workflow has to be repaired [GOR11].

- *Generic Interface* is an approach that proposes to build for every task in a workflow a generic open interface to apply dynamic changes more flexible at individual tasks [SHB98] (see Figure 2.13).

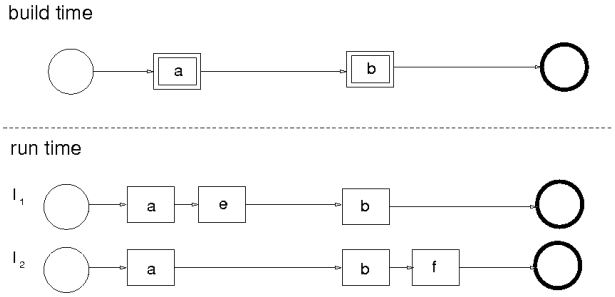


Figure 2.13 – Generic interface: for every task a generic open interface for applying changes is offered.

2.4.4 Discussion of dynamic workflows

Dynamic workflows are important concepts since assumptions made at build time can be wrong at run time [RM03]. Furthermore, there are processes that are too dynamic to be predictable at build time, that is, there is a lack of a priori knowledge [KLB+08]. Additionally, a high flexibility of processes can lead to too large and confusing workflow definitions.

Workflow management systems that handle dynamic changes have to cope with following challenges [RM03]: They have to:

- recognize changes relevant for workflows definitions and workflow instances they manage,
- support or automatically process dynamic adaptations, and
- verify the proposed adaptation.

We introduced the three main categories of dynamic workflows described in the literature. The approaches differ in the time when the adaptation is applied (build time or run time) and therefore their influence on workflow definition and workflow instances. Schema evolution is applied at build time and influences all workflow instances. Ad-hoc changes are applied at workflow instances at run time and influence only one instance. In the open-point approach the points of possible adaptation are fixed in advance in the workflow definition and can be filled individually at run time. That makes them inflexible to react to changes at tasks not foreseen for changes. Changes are applied to one instance. A short overview of the properties of dynamic workflows is shown in Table 2.1.

Approach	Schema evolution	Ad-hoc changes	Open-point approach	Multi instance activity	Late composition	Generic interface
time of adaptation	build time	run time	run time	run time	run time	run time
affected instances	all instances	one instance	one instance	one instance	one instance	one instance
location of adaptation	at every task of wf definition	at every task instance of wf instance	at abstract tasks, in instances; addition of fragments	at marked tasks, in instances	the whole instance	at every task instance
kind of adaptation	change of wf definition	change of wf instance	completion of wf instance	adaptation of wf instance	composition of wf instance	change of wf instance
convenient for kind of wfs	all kinds of wfs	rarely predictable wfs	partly structured wfs	wfs where number of repeated elements not known until run time	wfs with fixed set of tasks but until run time unknown order of tasks	partly structured wfs
challenges of adaptation	running instances	consistency check of changes at run time	changes of instances at not abstract tasks; find fitting fragment at run time	abort of iteration	find fitting fragment at run time	find fitting fragment at run time

Table 2.1 – Overview of dynamic workflow properties

2.5 A formal model for static and dynamic workflows

For dynamic adaptations of workflows that can be defined and executed in correct and consistent way we want to define a formal model. Thus, the formal model supports the definition, verification and execution of workflows.

There are many different frameworks and languages that support the specification of formal models. We are faced with the problem that frameworks and languages suited to specify formal

models for the definition of workflows and those suited to specify formal models for the verification and execution of workflows have conflicting goals.
For the definition of workflows, that is, the mapping of real-world processes into a workflow, the knowledge of domain experts is required. Domain experts need a language for workflow definition that is intuitive, easy to learn and easy to understand. Such a language needs many modeling constructs and few modeling restrictions. These language properties are not only important because it eases the correct definition of workflows, but also because the defined workflow models are used for discussions (e.g. with further domain experts, software developers or the management) resulting in problem detection and process improvements.
In contrast, frameworks and languages that are easy analyzable, to verify workflow models and check them for properties such as reachability, liveness, and boundness requires a restricted structure with few modeling constructs, and a strong mathematical basis. This is also true for process execution languages.

The graphical modeling language BPMN, that we use in this thesis for drawing workflows, is a de facto standard for process modeling. BPMN does not focus on technically realizing business processes, that is, it does not focus on process execution [DDDG08]. Business processes can be defined as well by languages such as the Web Service Choreography Description Language (WS-CDL), a non executable modeling language; the Business Execution Language (BPEL) and the XML Process Definition Language (XPDL) – both languages focus on process execution and offer a precise execution semantics. For both the BPMN standard defines a translation from BPMN. An alternative to BPEL and XPDL is Yet Another Workflow Language (YAWL) that offers, other than the former two, direct support for (almost) all Workflow Patterns described by the workflow patterns initiative [vdAADtH04]. This language also focuses on process execution and a mapping tool from BPMN to YAWL (BPMN2YAWL) exists [DDDG08]. The combination of graphical and user focused modeling languages with textual and execution focused modeling language is a solution implemented in many modern WfMS.

Execution focused languages need a precise execution semantics. Therefore, they are usually based on frameworks such as Petri nets [vdAADtH04] (YAWL), or the frameworks are directly used as modeling and/or execution language [vdA98]. In conjunction with the search for frameworks that allow a more flexible and dynamic process execution, frameworks such as process algebras e.g. π-Calculus [PW05] and monadic combinator libraries in functional programming languages [PAK+11] are proposed. Beside the usage of existing frameworks, the creation of proprietary frameworks, especially for handling the specialties of dynamic workflows, is proposed. Reichert and Dadam [RD98] suggest a formal model that represents the control flow as a directed, structured graph.

In our work we will use Petri nets as framework for process execution. Workflows are easy to map to Petri nets and Petri nets support a lot of control flow patterns directly [vdAADtH04]. On the other hand, there are many different kinds of Petri nets, some of them allow for a high flexibility during process execution. In the following we will describe different kinds of Petri nets

and we will have a look at their suitability for workflow execution and their flexibility regarding structural changes.

2.5.1 Petri nets

Petri nets were developed by Carl Adam Petri and they were introduced in Petri's thesis "Kommunikation mit Automaten" (Communication with automata) in 1962 [Pet62]. These nets are often used to model and analyze concurrent processes, which are typical for distributed systems. Petri nets are a visual language, a mathematical theory, and a formal language. There are many advantages of using Petri nets as a basis for workflow modeling, execution and controlling (see [vdA98]):

- Petri nets have a formal mathematical definition.
- Petri nets can be visualized by an intuitive graphical notation.
- Petri net elements can easily be mapped to workflow elements. Furthermore, there are kinds of Petri nets that support all routing constructs used by workflow systems.
- Petri nets are well researched and documented.
- There are a lot of analysis techniques for Petri nets, such as, techniques to find out whether a Petri net is bounded, live, deadlock free, or whether a marking M of a net N can be reached from the initial marking M_0 of N (for more information see [EN94]).
- Simulations can be used for model checks.
- Petri nets are vendor and tool independent.

Since the first publication a lot of different kinds of Petri nets were developed, such as Predicate Transition nets [GL81], Continuous Petri nets [DA93] and Colored Petri nets [Jen81, Jen89]. One motivation to develop new kinds of Petri nets was that simple forms of Petri nets tend to get very large when large processes are modeled. The size of processes can be reduced by using distinguishable tokens and by transferring aspects of the model into parameters [GL81], e.g. arc expressions. That is, such nets represent processes usually tighter, and better understandable than simple nets because of the usage of more abstract representations. The extended Petri nets are called *high level Petri nets*. Petri nets with indistinguishable tokens are called *low level Petri nets*.

We will start with the description of the classic and commonly used form of Petri nets - Condition-Event-Nets. This kind of Petri nets is easy to understand and suitable for an introduction.

2.5.2 Condition-Event-Nets

Condition-Event-Nets (CEN) are low level Petri nets, their tokens are not distinguishable from each other. A Condition-Event-Nets is defined as follows:

Definition 13 (Condition-Event-Net) . A Condition-Event-Net is a four tuple $CEN = (P, T, F, M_0)$, where:

- P is a finite set of places (conditions).
- T is a finite set of transitions (events), whereby $P \cap T = \emptyset$.
- F is a set of directed arcs, where $F \subseteq (P \times T) \cup (T \times P)$.
- M_0 is the start marking, where $M_0 : P \to \{0, 1\}$.

□

Places are passive components and represent states. They are graphically denoted by a cycle. Transitions are active components. They describe actions and are represented by a rectangle. Petri nets in general are bipartite graphs: directed arcs connect places and transitions and vice versa, but never places with places and transitions with transitions. Places, transitions and

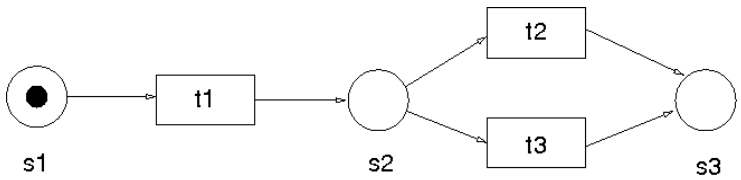


Figure 2.14 – Condition-Event-Net: places P = {s1,s2,s3}, transitions T = {t1,t2,t3}, connecting arcs e.g. f_1 = (s1,t1) and start marking $M_0(s1)$=1, $M_0(s2)$=0, $M_0(s3)$=0.

arcs constitute the static and non changeable part of Petri nets. Tokens, residing in places and representing objects, are the dynamic components of Petri nets. An example of a Condition-Event-Net is shown in Figure 2.14.

Tokens constituting the marking of a Petri net. In Condition-Event-Nets tokens are represented by black dots in places, so called anonymous tokens. Every place has at most one token, in other words: the places in a Condition-Event-Net have the capacity one. The initial distribution of tokens over the net is called start marking and denoted by M_0. By firing transitions, a new marking (a new distribution of tokens) is created.

Definition 14 (Firing rule) . Firing is the consumption of tokens from the input places of a transition and the production of tokens in the output places of the same transition. □

A place p is an input place to a transition t when it is connected with transition t by an arc from p to t. A place p is an output place to a transition t when it is connected with transition t by an arc from t to p. The precondition for firing is the enabling of transitions.

Definition 15 (Enabling rule) . A transition is said to be enabled, if:

i) the input places of a transition ${}^\bullet t := \{p \in P | (p,t) \in F\}$ have enough tokens for consumption by the transition t and

ii) the output places $t^\bullet := \{p \in P | (t,p) \in F\}$ have enough space for the production of tokens.

□

This means, in case of Condition-Event-Nets, that before firing every input place must have exactly one token and the output places must not have tokens. While firing the transition consumes the tokens of the input places and produces one token per output place. After enabling, a condition can but does not have to fire (immediately) — this is the case for all kinds of Petri nets in which tokens reside.

Definition 16 (Conflict) . Two (or more) transitions are in (forward) conflict if they have a common input place[11] and are enabled. The firing of one transition would cause the disabling of the other. □

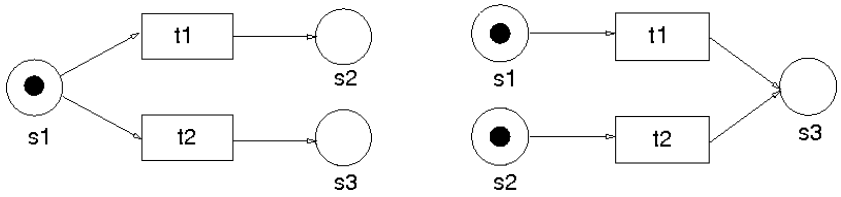
When two or more transitions with a common input place are enabled then a firing of one of them would cause that the remaining transition are no longer enabled as there is no longer a token in their input place (see [GT00, DA05]). This situation is called conflict and one example of a conflict is shown in 2.15(a).

Definition 17 (Backward conflict) . Two (or more) enabled transitions are in backward conflict (contact) when they have a common output place [Mat96]. □

In CENs the firing of two or more enabled transitions with a common output place at the same time would cause more than one tokens in the output place. That is not allowed for CENs (see Definition 13). Thus, one transition will fire and it is not in advance clear which. None of the remaining enabled transitions can fire as long as the output place has a token. This situation is called backward conflict and it is shown in 2.15(b).

Petri-Nets are *non-deterministic*. Therefore, in case of a conflict or backward conflict any of the transitions fires and it is not clear in advance which of them will be fired. In other words: in conflict situations there are not enough information to determine the subsequent control flow [Mat96].

[11] The situation of more than one transitions sharing a common input place is also called structural conflict [DA05].



(a) Conflict: t1 and t2 are in conflict. The firing of t1 would disable t2 and vise versa.

(b) Backward conflict: t1 and t2 are in backward conflict. In advance it is not clear whether t1 or t2 fires.

Figure 2.15 – (Forward) conflict and backward conflict situation in Condition-Event-Nets.

Mapping of Workflow concepts to Petri nets

The mapping of workflow management concepts to Petri nets, and hence CENs, is straightforward: *tasks* are mapped to *transitions*, *conditions* are mapped to *places* and *workflow instances* are modeled by (one or more) *tokens* [vdA98]. Figure 2.16 visualizes this mapping.

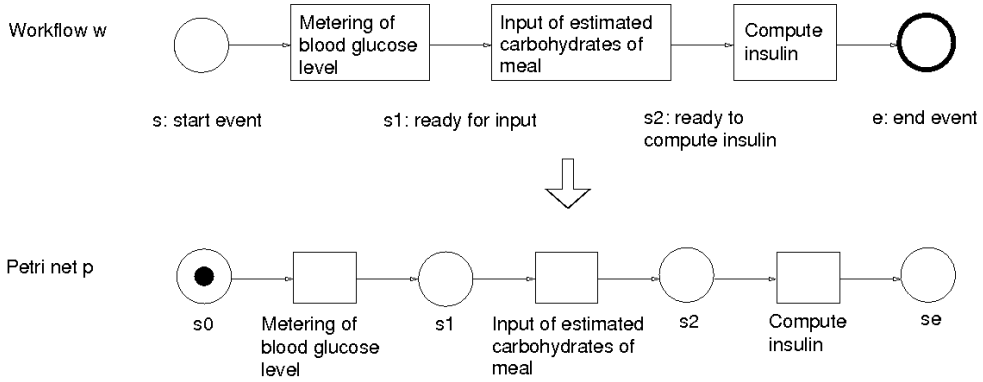
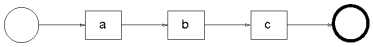
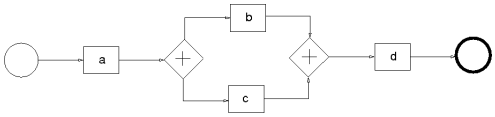


Figure 2.16 – Mapping from workflow w to Petri net p: tasks are mapped to transitions e.g. metering of blood glucose level; conditions are mapped to places e.g. ready for input to s1; an instance of workflow w is mapped to the token in p

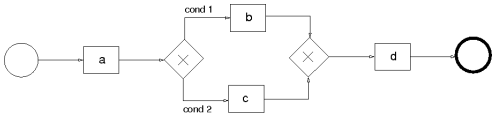
Of course, routing constructs of workflows have to be mapped to Petri nets. Workflow patterns (see Section 2.3.1) describe building blocks of workflows that are used to model routing. There are four main types of routing: (a) sequential — one after the other execution of task instances, (b) parallel — task instances can be executed simultaneously or in any order, (c) selective (or conditional) — when there is a choice (dependent on some variables) between tasks and (d) iterative — a task instance is executed more than once in a process instance until a condition is met [vdAvH04]. Because these four basic routing constructs have to be supported by workflows there has to be an adequate mapping of them to Petri nets. In the following text we will describe the mapping of that constructs according to [vdA98]. Figure 2.17 shows on the left hand side routing constructs of workflows. On the right hand side their mapping into Petri nets is depicted.



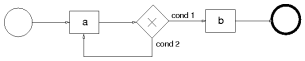
(a) Sequential routing: uses the sequence pattern



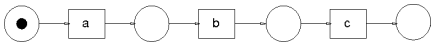
(b) Parallel routing: uses the patterns parallel split and synchronization



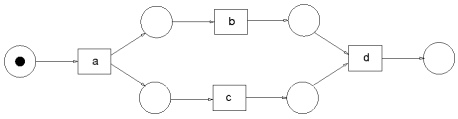
(c) Conditional routing: uses the patterns exclusive choice and simple merge



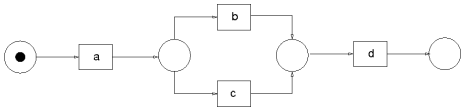
(d) Iterative routing: uses the pattern exclusive choice



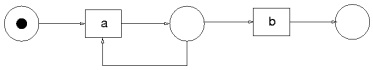
(e) Sequential routing: workflows can be correctly mapped to CENs



(f) Parallel routing: workflows can be correctly mapped to CENs



(g) Conditional routing: Non-deterministic choice — incorrect mapping!



(h) Iterative routing: Non-deterministic choice — incorrect mapping!

Figure 2.17 – Routing: on the left the routing constructs of workflows are shown and on the right their mapping into Petri nets according to [vdA98] is depicted.

Sequential routing As explained before, tasks are mapped to transitions and conditions are mapped to places (see Figure 2.16 as well as Figures 2.17(a) and 2.17(e)).

Parallel routing (Figures 2.17(a) and 2.17(e)) is a kind of routing where a branch of a workflow is split into several branches that are executed concurrently. A concurrent execution of branches means that the order of execution of the branches is not important or arbitrary, that is, it does not matter which branch is executed first or if the execution of these tasks really overlaps in time. The branches have to be synchronized before the thread of control is passed to the subsequent branch. Parallel routing of workflows is modeled by the basic control flow patterns *Parallel Split* and *Synchronization* (see Figure 2.4). Both patterns can be mapped to ordinary transitions of Petri nets. The execution of the 'Split'-transition enables the subsequent branches and the execution of the 'Synchronization'- transition synchronizes the branches.

Conditional routing (Figures 2.17(c) and 2.17(g)) is modeled if there is a choice between two workflow branches. The patterns exclusive choice and simple merge are used to model that kind of routing. The problem is that CENs are non-deterministic (see Figure 2.15). CENs can model a choice as a place that is a common input place of multiple following transitions

and a simple merge as a place that is a common output place of multiple transitions. Only one of the branches is executed. But because of the non-determinism an arbitrary branch is chosen for execution. It cannot be defined which branch is chosen, that is, CENs do not support deterministic choices. Later on we will show that high level Petri nets can solve this problem.

Iterative routing (Figures 2.17(d) and 2.17(h)) can be modeled by the pattern exclusive choice. Iterative routing has the same problem as conditional routing it is not possible to make deterministic choices with CENs.

Usually there is more than one workflow instance for a workflow definition. To each workflow instance belong one or more tokens. When tokens of more than one workflow instance reside in the same Petri net then token could get mixed. To avoid this, for every workflow instance a unique instance of the Petri net is used.

Are Condition-Event-Nets suitable for the definition and verification of static workflows?

As explained before, distributed static workflows with sequential routing, parallel routing and non-deterministic choices can be mapped to Condition-Event-Nets. Deterministic choices are not supported by CENs. Therefore, also iterations, that build on deterministic choices cannot mapped correctly. Beside this insufficiency in mapping workflow definitions, Condition-Event-Nets are getting large when they are used for the description of large systems. Such large nets are difficult to understand and to use. Thus, (non hierarchically) CENs are only suitable for the definition of a subclass of static workflows, namely small workflows without deterministic choices and iterations, that are not hierarchically organized.

Are Condition-Event-Nets suitable for the definition and verification of dynamic workflows?

Dynamic workflows are workflows that are created to facilitate changes to their structure not only at build time but also at run time. In Section 2.4 we described different ways to adapt workflows dynamically. In this thesis, we concentrate on the support of partly defined workflows. There are several different approaches to implement partly defined workflows. The approaches of this class of workflows have in common, that they consist of building blocks (tasks or sub workflows) that are defined at build time and assembled at run time. Condition-Event-Nets do not offer the possibility to assemble building blocks at run time. Therefore, CENs are not suitable to adequately support dynamic workflows.

2.5.3 Hierarchical Petri nets

When Petri nets are used to model real-world systems, the models tend to be complex and large [ZB97]. That makes it difficult to get an overview of the model [JK09] and hence it complicates both understanding and use. Thus, a mechanism is needed that allows for comprehending the model without studying the whole model. Hierarchical Petri nets are instruments to simplify models by structuring them in different levels of abstraction. That is, at the topmost level Hierarchical Petri nets will give an overview of the model, exact enough to get a rough understanding of the system. The level below will refine the topmost level and explain hidden details (see Figure 2.18). The level below the topmost level can also be refined by another level and so on. Every level can be further refined. The refinement, described here, is also called node refinement. Thereby, simple elements (places or transitions) are *replaced by* more detailed nets, so called subnets (static refinement), or simple elements are *associated with* subnets (dynamic refinement) [ZB97]. Dynamic refinement allows the reuse of subnets, that is, a subnet can be associated with more than one simple element. This view describes a top-down approach, starting with a general process that is stepwise refined. Besides, a bottom-up approach exists that starts with a large process that has to be generalized by identifying subnets and replacing them by simple elements. In summary, Hierarchical Petri nets offer the possibility: to inspect the model at different levels of abstraction, to visualize the refinement of selected parts of the net and, in case of dynamic refinement, to reuse parts of the model [Feh93].

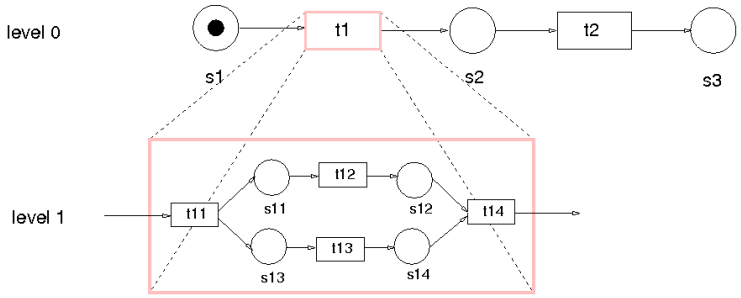


Figure 2.18 – Hierarchical Petri net with two levels: refinement of transition t1

Definition 18 (Subnet) . A net $N' = (P', T', F', M_0')$ is a subnet of a net $N = (P, T, F, M_0)$, when:

- $P' \subseteq P$ and
- $T' \subseteq T$ and
- $F' = F \cap ((P' \times T') \cup (T' \times P'))$ and
- $\forall p \in P' : M_0(p) = M_0'(p)$.

□

That is, a Petri net N' is a subnet of Petri net N when all elements of N' are elements of N. In this case, the initial marking of all places in N' is equal to the marking of the places in N [CMP08] (see Figure 2.19).

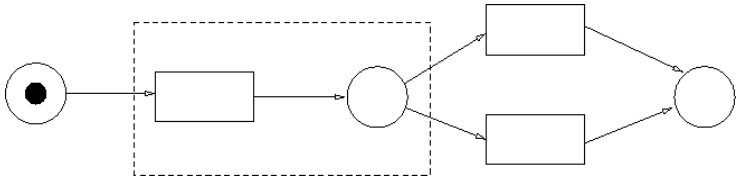


Figure 2.19 – Subnet: The dashed box marks a subnet of the shown net.

There are several approaches of refining of nodes in Petri nets. Different properties of Petri nets are preserved by different approaches. One common approach states that places of higher levels can be replaced by place bordered subnets, and transitions can be replaced by transition bordered subnets [Lak97, Lak00] (Figure 2.18 shows the replacement of a transition.) For the understanding of the concept *place/transition bordered subnet* we have to define the concepts: *preset*, *postset* and *border*:

Definition 19 (Preset [BGV91]) . In Petri nets, all elements $x(x \in P \cup T)$, with the exception of the start place, have a nonempty preset ${}^\bullet x$, where:

${}^\bullet x := \{y \in P \cup T | \ (y, x) \in F\}$. □

That is, the preset ${}^\bullet x$ of an element x is a set of elements y. For every element y there exists an arc $f \in F$ outgoing of y and incoming in x (see Figure 2.20).

Definition 20 (Postset [BGV91]) . All elements x, with the exception of the end element, have a non empty postset $x^\bullet$, where:

$x^\bullet := \{y \in P \cup T | \ (x, y) \in F\}$. □

The postset $x^\bullet$ of an element x is a set of elements y. For every element y there exists an arc $f \in F$ outgoing of x and incoming in y (see Figure 2.21).

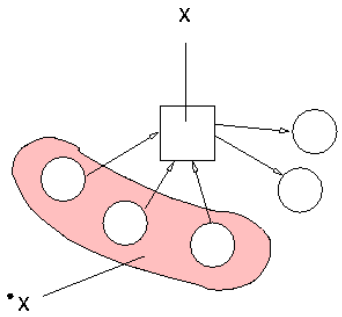


Figure 2.20 – Preset ${}^\bullet x$ of x

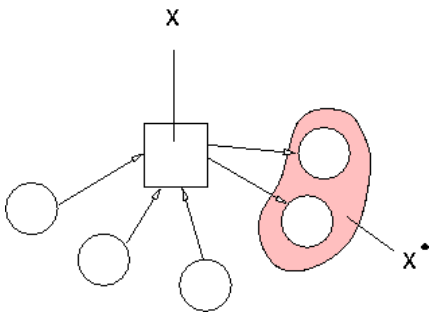


Figure 2.21 – Postset $x^\bullet$ of x

Definition 21 (Border) . A border x of a subnet N' (of a net N) is defined as: $\{x \in P' \cup T' \mid ({}^\bullet x \cup x^\bullet) \backslash (P' \cup T') \neq \emptyset\}$. That is, preset ${}^\bullet x$ and postset $x^\bullet$ can lay outside the subnet N' but inside N in case of border elements. □

That is, a border of a subnet N' are places or transitions of the subnet. Thereby, border elements have a preset or postset outside the subnet in the net N, or in case the border element is a start element or an end element of N, it has no preset or postset.

If the border only consists of places $\{p \in P' \mid ({}^\bullet p \cup p^\bullet) \backslash T' \neq \emptyset\}$, the subnet is called *place bordered* (see Figure 2.22). If the border only consists of transitions $\{t \in T' \mid ({}^\bullet t \cup t^\bullet) \backslash P' \neq \emptyset\}$ the subnet is called *transition bordered* (see Figure 2.23).

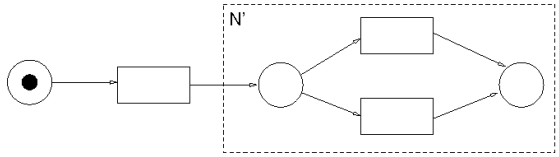


Figure 2.22 – Place bordered subnet N'

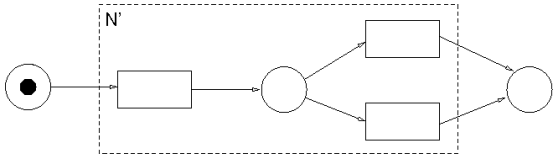


Figure 2.23 – Transition bordered subnet N'

Are Hierarchical Petri nets suitable for the definition and verification of static workflows?

Hierarchical Petri nets were created to ease the understanding and use of Petri nets by introducing levels of abstraction. By introducing abstraction levels a significant improvement of the net presentation respectively structuring and thus usability can be achieved. They do not add constructs for the improvement of the mapping from workflows to non-hierarchical Petri nets, but they introduce the mapping from hierarchical workflows to hierarchical Petri nets. Hierarchical Petri nets support different kinds of Petri nets — from low level to high level Petri nets — thereby, the supported net determines the supported routing constructs of Hierarchical Petri nets.

Are Hierarchical Petri nets suitable for the definition and verification of dynamic workflows?

As described before non hierarchical CEN have no means to support a dynamic assembly of workflows. Hierarchical CEN support the refinement of nodes. Places can be replaced by place bordered subnets and transitions by transition bordered subnets. In higher levels the subnets are hidden and a user has to look at a lower level of abstraction to see the whole workflow even though the whole workflow is always executed. If there was not only one sub workflow but a set of sub workflows that could be chosen dynamically at run time, Hierarchical Petri nets would support dynamic workflows. But this is not the case, therefore, Hierarchical Petri nets are not directly suitable for the definition of dynamic workflows.

2.5.4 Colored Petri nets

Colored Petri nets (CPNs) (see [JKW07, JK09]) were developed as a graphical modeling language for concurrent systems and the analysis of their properties. CPNs combine the capabilities of Petri nets with that of a high-level programming language. It adds to the above described properties of Petri nets the definition of data types (colors), and the description of data manipulation. CPNs use the programming language in net inscriptions such as initial markings, arc expressions and guards, in internal algorithms that calculate the enabling and occurrence of bindings, and in code segments belonging to transitions [Cor93]. In principle CPNs can use many different programming languages [Jen92]. The original literature uses the functional programming language CPN ML[12]. The combination of Petri nets and CPN ML makes CPN models compact and parameterizable but also more complex.

The most important contribution of CPNs is that tokens are attached with data values. The data values are called *token colors*. For each place the set of possible data types, the so called *color set*, has to be specified. In a place can only reside tokens of the specified color set.

In the following text let:

- $EXPR$ denote a set of expressions provided by the inscription language of the CPNs. (The inscription language is the programming language used in the CPNs.)
- $Type[e]$ denote the type of an expression $e \in EXPR$.
- $Var[e]$ denote the set of free variables in an expression e, where free variables are variables not bound in the local environment of the expression.
- $EXPR_V$ be the set of expressions $e \in EXPR$ such that $Var[e] \subset V$, where V is the set of variables.

Then Colored Petri nets are defined as follows:

Definition 22 (Colored Petri net) . CPNs are nine tuples $CPN = (P, T, F, \Sigma, V, C, G, E, I)$.

- P is a finite set of places.
- T is a finite set of transitions $(P \cap T = \emptyset)$.
- F is a set of directed arcs, where $F \subseteq (P \times T) \cup (T \times P)$.
- Σ is a finite set of non-empty color sets (data types).
- V is a finite set of typed variables (when $v \in V$ then $Type[v] \in \Sigma$).
- C: $P \to \Sigma$ is a color set function that assigns a color set to each place.

[12] CPN ML bases on Standard ML [JK09].

- G: $T \rightarrow EXPR_V$ is a guard function that assigns a guard to each transition ($Type[G(t)] = Bool$).
- E: $F \rightarrow EXPR_V$ is an arc expression function that assigns an arc expression to each arc $f(Type[I(p)] = C(p)_{MS}$ and p = place connected to f).
- I: $P \rightarrow EXPR_\emptyset$ is an initialization function that assigns an initialization expression to each place p such that $Type[I(p)] = C(p)_{MS}$.

□

Just like all kinds of Petri nets, Colored Petri nets have an easily understandable graphical notation. The most important graphical concepts are shown in Figure 2.24.

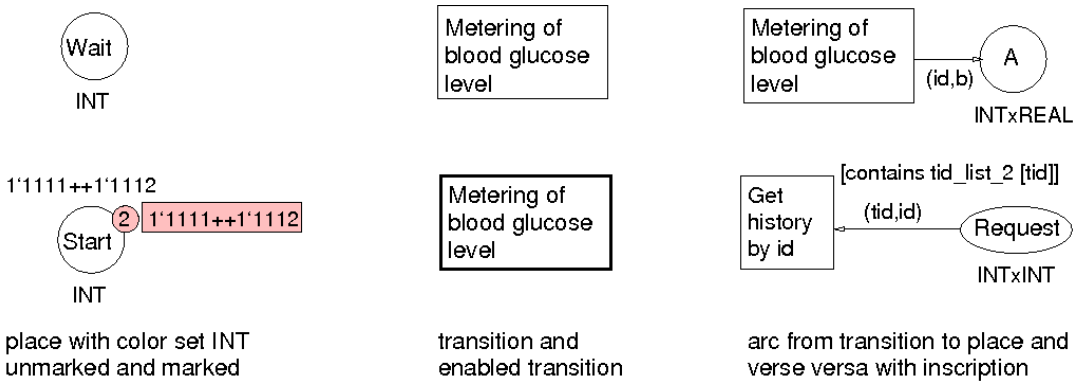


Figure 2.24 – Visual presentation of CPN concepts

Places represent the state of the modeled system [JK09] and are depicted by circles. For every place the color set (data type) of tokens that mark this place must be specified, e.g., INT for place *Wait*. If there is enough space, the color set is written below the place. Places can optionally have a name inscription and an initial mark inscription that specifies the initial tokens for this place. For a marked place, the number of tokens is shown in a small circle that is attached to the place. The detailed token colors (data values) are shown in a rectangle next to the small circle. This representation makes colored tokens visual distinguishable and allows to instantly grasp markings. In Figure 2.24 initial and actual marking of place *Start* is shown.

Transitions describe the actions in a net. They are represented by rectangles with optional inscriptions that determine bindings, such as guards, time, code segments. These inscriptions add extra constraints for the enabling of transitions. For instance, guards are Boolean expressions that are used to implement deterministic choices. Guards have to evaluate to true for enabling a transition. They are written in square brackets and positioned near to the transition. The transition *Get history by id* has the guard *contains tid_list_2 [tid]*. Transitions with a thick border represent enabled transitions.

Arcs connect places and transitions. Arcs have one inscription, whereby the color set of the arc expression has to match that of the attached place. In other words, arc inscriptions describe the colors of tokens that are fired. Arc inscriptions can be variables, constants, operators, and functions. The arc expressions decide about the enabling of a transition in a given marking.

After having introduced the concepts of CPNs we need to take a closer look at the example depicted in Figure 2.25[13]. The example reuses the diabetes scenario of Figure 2.3 and Figure 2.16 (dotted background) but adds the management of a metering history. In the following text we will focus on the scenario extension consisting of the components: data storing and data querying.

Storing data After the estimation of carbohydrates we will store the blood glucose level, the carbohydrates and the timestamp of storing, together with the id of the diabetic. We will store these values in a list, that stores records consisting of uid and a list of metered values. In Colored Petri nets, tokens may overtake each other. Thus, it may happen that a person wants to query the print history but this history is not up to date. Therefore, every token gets an id that is stored in a tid_list when the token values are stored in the history. This values in the tid_list can be easily compared to the requesting tid (see Figure 2.26). This tid is generated in the subnet as shown in 2.30. Thereby we used the ID Manager pattern described in [MvdA05].

Querying data For data inquiry we implemented the pattern *Synchronous Transfer* which is described in [MvdA05]. Thereby the sender is blocked until the requested data become available (see Figure 2.25 and 2.27).

Mapping of Workflow concepts to Colored Petri nets

Colored Petri nets are high level Petri nets with typed, and therefore distinguishable, tokens. In a CPN tokens represent a workflow instance. The token color represents the workflow attributes [vdA98] and the color values represent the attributes values of a case. By means of colored tokens deterministic choices can be exactly represented. Van der Aalst [vdA98] introduces two alternatives to model a deterministic choice basing on workflow attributes: (i) adding preconditions to each transition subsequent to the place representing the exclusive choice. A precondition is an additional enabling requirement that bases on one or more workflow attributes. This precondition allows that only one of the subsequent transitions are enabled. Figure 2.17(c) shows that case. The second case (ii) uses a transition as deterministic choice. The transition with more than one output place produces a token in one of the subsequent places (see Figure 2.28(a)). Van der Aalst points out that the difference between both cases is the moment of

[13] We modeled and simulated this example with CPN Tools, the quasi standard tool for modeling CPNs. For more information see http://cpntools.org/documentation/start.

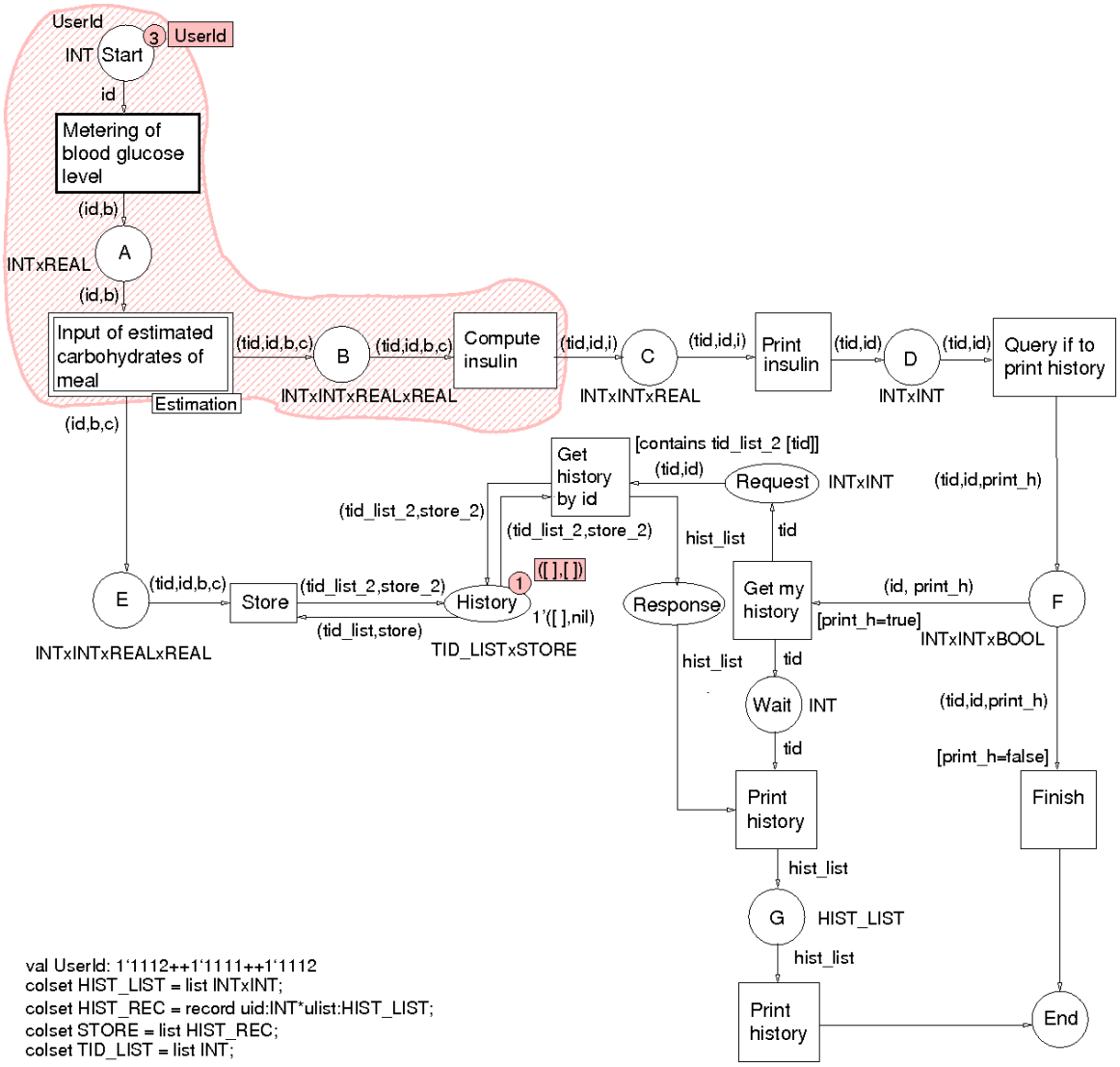


Figure 2.25 – Colored Petri net that illustrates a simplified detail of a form of therapy for diabetes (ICT). It describes the metering process (dotted background) plus the archiving of metered data and shows the initial marking of the net.

choice. In case (i) the choice is made as late as possible in the moment when the transitions with preconditions are executed. In case (ii) the choice is made earlier when the transition representing the choice is completed. For the modeling of iterations basing on workflow attributes the same both variants of mapping exist (see Figure 2.17(d) and 2.28(b)).

Are Colored Petri nets suitable for the definition and verification of static workflows?

In addition to the mapping of routing constructs CENs support, Colored Petri nets support also deterministic choices and iterations based on deterministic choices. Therefore, they are able

Figure 2.26 – Picture detail of 2.25: storing metered data in history.

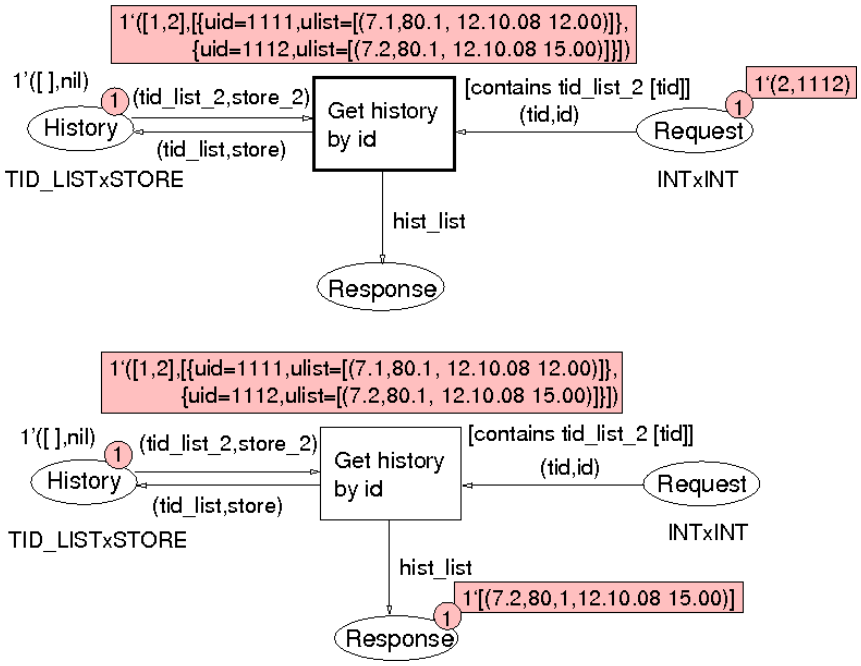
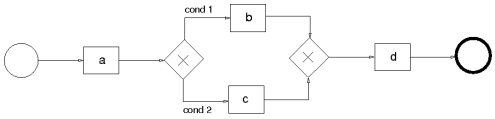
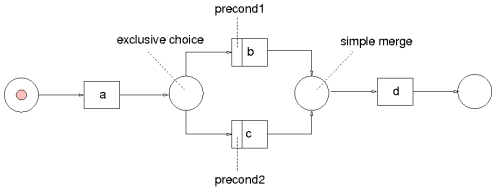


Figure 2.27 – Picture detail of 2.25: reading data from history.

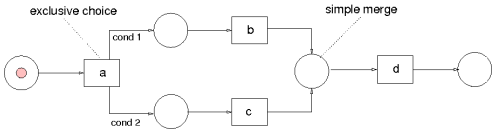
to map all routing constructs that can be used for building static workflows. On the other hand, nets, modeled by Colored Petri nets are much smaller than the same net modeled with a CEN because of the usage of token colors (data types). Additionally, when we analyzed CENs we found that we have to take care that different workflow instances are not getting mixed. To avoid this, for every workflow instance a unique instance of the Petri net is used. In case of CPNs there is another option to avoid the mixing of instances [vdA98]: In Colored Petri nets the usage of token colors makes it possible that tokens can contain information about the identity of the corresponding workflow instance. A way to check the identity of tokens is to use preconditions in transitions that inspect and compare tokens. Beside these significant advantages Colored Petri nets have the drawback of a higher complexity than Condition event nets. Additionally, non hierarchical CPNs are not able to express hierarchies of workflows.



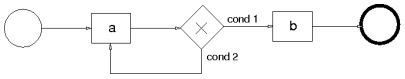
(a) Conditional routing: uses the patterns exclusive choice and simple merge



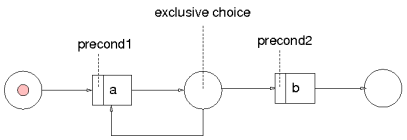
(a1) Conditional routing 1: deterministic choice with preconditions for the transitions b and c



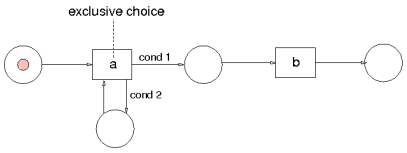
(a2) Conditional routing 2: deterministic choice, transition a is an exclusive choice



(b) Iterative routing: uses the pattern exclusive choice



(b1) Iterative routing 1: deterministic choice with preconditions



(b2) Iterative routing 2: deterministic choice, transition a is an exclusive choice

Figure 2.28 – Routing: on the left routing constructs of workflows, not supported by CENs, are shown. On the right the mapping of these constructs into CPNs according to [vdA98] is depicted.

Are Colored Petri nets suitable for the definition and verification of dynamic workflows?

For the ability to define dynamic workflows Colored Petri nets have to be extended. The dynamic cannot be expressed directly, even though Colored Petri nets offer the possibility to carry information in their tokens.

2.5.5 Hierarchical Colored Petri nets

When modeling large systems as non hierarchical CPNs, the models become very large. It is difficult to handle and to understand these models [JK09]. Jensen et al. propose the use of Hierarchical Colored Petri nets (HCPNs) to manage this problem. Just like Hierarchical Petri

nets (see Section 2.5.3), Hierarchical CPNs offer modelers the possibility to work on different levels of abstraction. Each level of a net can be refined by a subnet. A higher level of abstraction offers a more abstract view on the behavior of the net than a lower level. A high level view is less detailed. The modeler can abstract from details that are not needed for the current work and can have a more detailed view on the net if necessary. A further advantage HCPNs offer is the fact that subnets can be reused, which eases net maintenance.

A HCPN is a net, which can always be transformed into an equivalent non-hierarchical CPN. In turn, a non-hierarchical CPN can always be transformed into an equivalent Place/Transition net [JK09]. Therefore, HCPNs add no expressive power to Place/Transition nets although they have the above described advantages in practical use.

Language constructs

HCPNs divide a CPN in modules laying in different layers. The most relevant information can be found in the higher levels. The higher levels abstract from details. The lower levels hold the detailed information. In the following we explain the most important constructs of HCPNs referring to the publication [JK09]. Thereby, we use Figure 2.29 to explain the constructs.

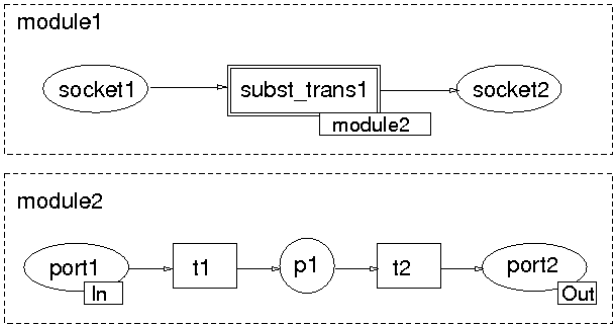


Figure 2.29 – Simple hierarchical colored Petri net with module module1 and submodule module2.

An important construct for implementing HCPNs is the *Substitution transition*. Substitution transitions generalize more detailed parts of the CPN. They act on higher levels of abstraction as placeholders for detailed transition bordered nets on lower levels. In a graphical model a rectangular with double-lined borders marks a substitution transition. The substitution tag attached to the double-lined rectangular contains the name of the substituting submodule. In Figure 2.29 there is only one substitution transition in module1: $T_{sub}(module1) = \{subst\_trans1\}$. The substituting submodule to this substitution transition is $module2$.

Definition 23 (Substitution transition) . Substitution transitions are a subset of all transitions in the CPN: $T_{sub} \subseteq T$. They can be substituted by a transition bordered subnet on a lower level of abstraction. Substitution transitions show a less detailed view on the behavior of the net than represented by corresponding subnets. □

Substitution transitions and substituting subnets (modules) are two different views on the same part of the CPN. Jensen et al. [JK09] propose to relate them by using *socket places* and *port places*. Socket places are the places that are connected by an arc with the substitution transition. Socket places are the interface of the substitution transition. Port places are the interface of the transition bordered subnet and they are usually equivalent to the interface of the substitution transition. In graphical models, port places are attached by a port-type tag. The tag describes whether a port belongs to a socket that is a In-, Out-, or I/O-node. In Figure 2.29 we have the following sets of port and socket places:
$P_{port}(module1) = \{\}$, $P_{sock}(module1) = \{socket1, socket2\}$,
$P_{port}(module2) = \{port1, port2\}$, $P_{port}(module2) = \{\}$.

Definition 24 (Socket place) . Socket places immediately surround the substitution transition. They are the interface of the substitution transition and can be of following type: input, output and input/output. Referring to the definition of subnet borders only consisting of transitions in Section 2.5.3, as: $\{t \in T' \mid ({}^\bullet t \cup t^\bullet) \backslash P' \neq \emptyset\}$, we define socket places as follows: $\{p \in P \mid \exists t \in T' : ({}^\bullet t \cup t^\bullet) \backslash P' \neq \emptyset \wedge p \in^\bullet t \vee p \in t^\bullet\}$. □

Definition 25 (Port place) . Port places constitute the interface of the subnet to the environment. They are added to transition bordered subnets. □

Port-socket relations bind a substitution transition and the belonging subnet by relating belonging port and socket places. In Figure 2.29 the port-socket relations are $(socket1, port1)$ and $(socket2, port2)$.

Definition 26 (Port-socket relation) . Port-socket relations relate port place (p') and belonging socket place (p) and constitute two different views of a single place.[14] The port-socket relations describes how the environment of the substitution transition interacts with the subnet. □

The port-socket relations of a single substitution transition are grouped in a port-socket relation function. For Figure 2.29 this function is $PS(module1) = \{(socket1, port1), (socket2, port2)\}$:

Definition 27 (Port-socket relation function) . A Port-socket relation function PS assigns port-socket relations to substitution transitions.

$PS(t) = \{(p, p')\}$, whereby: p is a socket place of the substitution transition t, and
p' is a port place of submodule $SM(t)$ assigned to t. □

Corresponding ports and sockets have to have the same type (input, output, input/output). Socket and port type functions assign types to this places. For Figure 2.29 we see the socket type function:

$$ST(module1) : \begin{cases} socket1 \mapsto IN \\ socket2 \mapsto OUT \\ \emptyset \mapsto I/O \end{cases}$$

[14] Hence, related port and socket places share the same marking.

and the port type function:

$$PT(module2) : \begin{cases} port1 \mapsto IN \\ port2 \mapsto OUT \\ \emptyset \mapsto I/O \end{cases}$$

Definition 28 (Type function) . Socket type functions assign a socket type to each socket place. Port type functions assign a port type to a port place.

Hierarchical Colored Petri nets are a composition of *colored petri net modules*. Thereby, places and transitions of different modules have to be disjoint. Figure 2.29 shows the modules module1 and module2.
Colored Petri Net Modules are defined as follows [JK09]:

Definition 29 (Colored Petri Net Module (CPN Module)) . A Colored Petri Net Module CPN_M is a four-tuple: $CPN_M = (CPN, T_{sub}, P_{Port}, PT)$[15], where:

1. $CPN = (P, T, F, \Sigma, V, C, G, E, I)$ is a Colored Petri Net.
2. $T_{sub} \subseteq T$ is a set of substitution transitions.
3. $P_{Port} \subseteq P$ is a (possibly empty) set of port places.
4. $PT : P_{Port} \rightarrow IN, OUT, I/O$ is a port type function that assigns a port type to each port place.

□

A substitution transition is mapped into a submodule via a submodule function. For our example in Figure 2.29 this means: $SM : subst\_trans \mapsto module2$.

Definition 30 (Submodule function) . A submodule function SMF assigns a submodule to each substitution transition:
$SMF : T_{sub} \rightarrow SM$. □

In summary, HCPNs are defined as follows (see [JK09])[16]. HCPNs are composed of CPN modules. Modules are hierarchically organized by relating substitution transition and belonging sub modules by submodule functions. Thereby, port-socket relations (pairs of socket places of a substitution transition and port places of belonging sub modules) are assigned to the substitution transition by port-socket relation functions.

[15] [JK09] did not explicitly define socket places as part of the CPN Module as they are implicitly given because arcs connect socket places with substitution transitions.

[16] We will skip the definition of fusion sets because they are irrelevant for our work.

Definition 31 (Hierarchical Colored Petri nets (HCPNs)) . Hierarchical Colored Petri Nets are a tuple $CPN_H = (SM, SMF, PS)$:

1. SM is a finite set of CPN Modules sm, whereby:
   $sm = ((P^{sm}, T^{sm}, F^{sm}, \Sigma^{sm}, V^{sm}, C^{sm}, G^{sm}, E^{sm}, I^{sm}), T^{sm}_{sub}, P^{sm}_{port}, PT^{sm})$.
   It is required that $(P^{sm_1} \cup T^{sm_1}) \cap (P^{sm_2} \cup T^{sm_2}) = \emptyset$ for all $sm_1, sm_2 \in SM$ such that $sm_1 \neq sm_2$.
2. SMF is a submodule function, that assigns submodule and substitution transition:
   $SMF : T_{sub} \rightarrow SM$.
3. PS is a port-socket relation function that assigns a port-socket relation $PS(t) \subseteq P_{sock}(t) \times P^{SM(t)}_{port}$ to substitution transition t. It is required that $ST(p) = PT(p'), C(p) = C(p')$, and $I(p)\langle\rangle = I(p')\langle\rangle$ for all $(p, p') \in PS(t)$ and all $t \in T_{sub}$.

□

Figure 2.30 shows a more complex example of a HCPN that is a part of Figure 2.25. The example focuses on the hierarchical components. We call the CPN module that is a detail of Figure 2.25 ICT_simplified. ICT_simplified contains the substitution transition *Input of estimated carbohydrates of meal* that can be substituted by the more detailed submodule *Estimation*. As explained before, a CPN module is connected with its submodules by equating places in module and submodule. These places are called socket places in the module and port places in its submodule. In our example net $A^{ICT\_simplified}$, $B^{ICT\_simplified}$, and $E^{ICT\_simplified}$ and $A^{Estimation}$, $B^{Estimation}$, and $E^{Estimation}$ are such places. In our example places have the same name in different modules. In this case the module name is superscript to the place name: $place^{module}$.

Mapping of Workflow concepts to Hierarchical Petri nets

Because HCPNs build on Colored Petri nets, the mapping to workflows works in the same way as described for CPNs (see Section 2.5.4). Additionally, HCPNs offer the possibility to map hierarchies in workflows. A sub process without start and end node corresponds to a subnet (a sub module without port transitions) of a HCPN.

Are Hierarchical Colored Petri nets suitable for the definition and verification of static workflows?

HCPNs build on CPNs. CPNs support also deterministic choices and iterations that base on deterministic choices. Therefore, they are able to map all routing constructs that can be used for building static workflows. Additionally, hierarchical constructs can be expressed. HCPNs offer

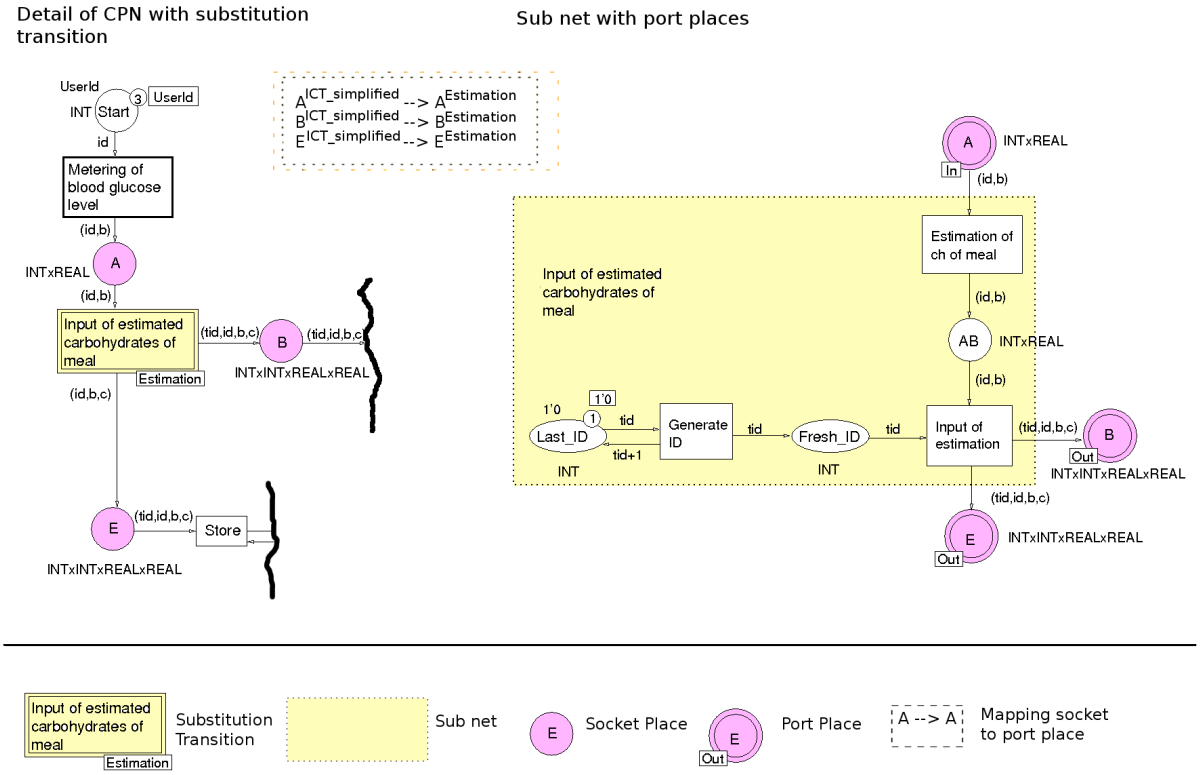


Figure 2.30 – Example of a HCPN (a part of Figure 2.25)

more modeling constructs and are therefore more compact and a bit more complex than CPNs. HCPNs can also avoid the mixing of instances [vdA98] by tokens that contain information about the identity of the corresponding workflow instance.

Are Hierarchical Colored Petri nets suitable for the definition and verification of dynamic workflows?

We described that the possibility to have a set of subnets that could be chosen at runtime would support dynamic workflows. HCPNs support subnets (submodules). Colored Petri nets offer the possibility to carry information in their tokens. The colors could carry information that allow for finding the convenient subnet. HCPNs can be used for the definition and verification of dynamic workflows but it is necessary to extend the concept with a component for dynamic binding of submodules to a substitution transition at run time. Colored Petri nets offer the possibility to carry information in their tokens. The colors could carry information that allow for finding the convenient subnet. HCPNs can be used for the definition and verification of dynamic workflows but it is necessary to extend the concept with a component for dynamic binding of submodules to a substitution transition at run time.

2.6 Rule-based systems

There are different ways to select a subnet in line with the current context, such as: manual selection, writing a new computer application to select a subnet, semantic (ontology based) selection, or a selection by Rule-based systems (RBSs). In our work we develop an automated approach, which is flexible, but avoids complex computation and a large footprint. To achieve this, we have chosen Rule-based systems to select subnets. RBSs allow for a separation of rules from the process description and thus rules can be changed at run time without changing the process description. In this section we describe RBS in detail.

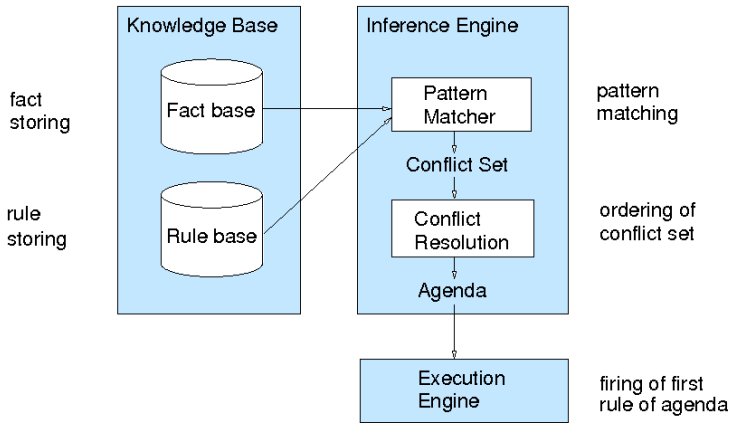


Figure 2.31 – rule engine: The knowledge base stores the knowledge of the rule engine, that is, the rules are stored in the rule base and the facts in the fact base. The inference engine is the core of a rule engine. Its tasks are the matching of facts against conditions of rules, processed by the pattern matcher and the ordering of the result of the pattern matching, processed by the Conflict resolution. (see [Neg11, GR05])

Rule-based systems were developed as a means to imitate the problem solving process of human experts electronically. That is, these automated experts are, just like a human expert, able to store, reason about and apply knowledge automatically. The foundation for these systems were developed by Newell and Simon in the 60th and 70th of the last century (see [NS63]). Such a system is shown in Figure 2.31 and consists of:

The *knowledge base*, that stores the expert knowledge. It consists of rule base and fact base:

- The *rule base* stores the domain knowledge. This knowledge is represented by rules. That is, by adding new rules to the rule base, knowledge is added to the RBS [Neg11]. It is possible to remove rules from the rule base. This allows to keep the knowledge in the rule base up to date. Rules are written in a rule language in a declarative way.
- The *fact base* stores the data (facts) about the current situation [Neg11]. These facts are matched against the rules.

The *inference engine*, that is the core of the rule-based system. It matches rules against facts and infers new facts. That is, it uses the available knowledge to generate new conclusions [AB92] respectively new knowledge. It has the following components:

- The *pattern matcher*, that matches facts against the condition-part of rules[17]. To fasten up pattern matching of many rules against many facts, algorithms such as Rete [For79] and Leaps [Bat94] were created. The result of the pattern matching is the *conflict set*, an unordered list of rules with satisfied conditions.
- The *conflict resolution*, that prioritizes the conflict set by applying strategies such as: priority first, specific rules first [San87]. The output of conflict resolution is an ordered list of rules with satisfied conditions, called *agenda*.

The *(rule) execution engine* executes the rule with the highest priority in the agenda. After execution, the execution engine adds the resulting new facts to the fact base.

In analogy with the human brain, the rule base is also called *long term memory*. Rules are usually valid for a long time span. The fact base is also called *short term memory* [Neg11]. Facts are currently valid data, they are valid only for a short time span. The inference engine which matches facts against rules and infers new facts is also called *reasoning engine*.
This consequent implementation of the analogy results in a clear separation of knowledge and its processing. Thus, building and maintaining of the system is facilitated [Neg11] and the quality of problem solving is improved. For instance, the rule base is easy to maintain since changes to the rule base (adding and removing of facts) are easy to accomplish [Neg11]. Additionally, just like the human brain, a Rule-based system can deal with incomplete, uncertain and fuzzy data and allows for inexact reasoning [Neg11].

2.6.1 Rules

Rules, that give rule-based systems their name, are a commonly known and used way of human experts to represent knowledge. Rules can represent relations, heuristics, recommendations, directives and strategies. They are characterized by a comparatively simple structure, consisting of an if-part called *condition* (premise or antecedent) and a then-part is called *conclusion* (consequent):

```
IF <condition>
THEN <conclusion>
```

The condition (pattern) is a *fact*, consisting of an *object* and a *value* combined by an *operator* (= ,< ,> ,...).

[17] Pattern matching in Rule-based systems differs from string pattern matching algorithms, such as the Boyer-Moore algorithm. String pattern matching algorithms search for a string, the pattern, in a text, that is, in another string [Gal79]. Rule pattern matching algorithms, search for facts that satisfy the conditions of a rule.

```
IF <object operator value>
THEN ...
```

An example is:

```
IF blood sugar level(x) < 3,9 mmol/L
THEN x has hypoglycemia
```

A rule can have more than one condition. Multiple conditions are connected by conjunction and/or disjunctions. The conclusion can consist of multiple parts that share the same condition(s).

```
IF <condition1> AND/OR <condition2>
THEN <conclusion1> AND <conclusion2>
```

That is, a rule is a piece of knowledge that declares, which conclusion(s) can be inferred when a certain condition/certain conditions are known [RODS78].

When the then-part describes an action, then the rule is of type of *production rule*. Production rules are often used in the controlling of production systems, e.g., in automotive industry. When the condition of the rule is satisfied (true), the rule can be fired, that is, the conclusion can be executed. The firing of a (production) rule may change the fact base by adding a new fact [Neg11].

Rules are determined as well as selected by an inference engine and they are executed by an execution engine to solve problems. Thereby, *determining* means that patterns are matched against facts. From the resulting list of rules usually one rule is *selected* for execution (acting, firing). This problem solving process is an iterative process and is also known as *recognize-act cycle* [MBLG90].

The inference engine can use different *inference rules* to produce new knowledge, that is, to add new facts to the rule base, such as *modus ponens* and *modus tollence* [AS10]. This production of new facts from rules by means of inference rules is called *inference*.

The pattern matching needed to reach a conclusion, produces *inference chains*. This chains show how rules are applied to reach a conclusion [Neg11] or in other words: an inference chain is a group of multiple inferences that connects a problem with its solution [GR05]. Basically, there are two problem solving strategies: forward chaining and backward chaining. In the next sections, we will have a closer look at these both inferencing methods used for problem solving.

2.6.2 Forward chaining

Forward chaining is *data directed* inference. It is the reasoning from facts to the conclusion. The chain of inferences is traversed from the problem to solution [GR05]. In forward chaining

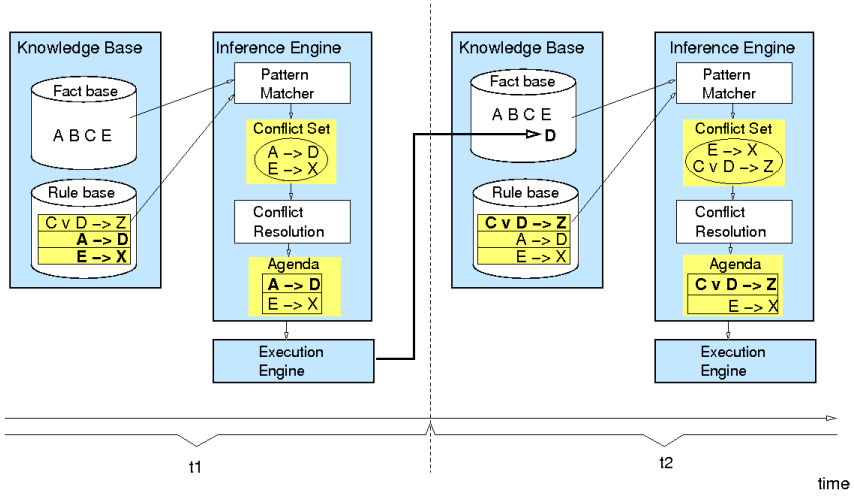


Figure 2.32 – Forward chaining: At time t1, the pattern matcher matches rules and facts to compute a conflict set of applicable rules. The conflict resolution creates from the conflict set the agenda, an ordered list of rules. The first rule of the list (A $\rightarrow$ D) is fired by the execution engine and a new fact (D) is added to the fact base. At time t2 the fact base contains D and the next rule to be fired is searched. (The figure refers to [AB92, Neg11].)

first the rules which can be applied are determined by the pattern matcher. The result of this process is the so called *conflict set*. The conflict set is ordered in the conflict resolution process. There exist different strategies to order the list, e.g., prefer more specific rules, prefer newer rules, prefer rules with a higher priority, use meta rules [San87]. The *agenda* is the result of conflict resolution. The first item of the agenda is executed by the execution engine. This leads to a new entry in the fact base. The steps, pattern matching, conflict resolution and firing are as long executed as no rule is in the agenda or if exit is specified by the fired rule. The principle of forward chaining is depicted in Figure 2.32.
We will sketch an extremely simplified example of the diabetes scenario, with an empty fact base and the following rule base[18]:

```
#1 RULE hypoglycemia
   IF blood_sugar_level(x) < 2,78 mmol/L AND
      (being_hungry(x) OR being_pale(x) OR sweat(x))
   THEN x has hypoglycemia

#2 RULE hyperglycemia
   IF blood sugar level(x) > 6,9 mmol/L AND
      (frequent hunger OR frequent thirst OR frequent urination)
```

[18]The values for hypoglycemia and hyperglycemia differ in the literature. The values we use are preprandial plasma glucose levels for adults with diabetes and we refer to The American Diabetes Association http://www.diabetes.org/living-with-diabetes/treatment-and-care/blood-glucose-control/checking-your-blood-glucose.html .

```
   THEN x has hyperglycemia

#3 RULE handle_hypoglycemia
   IF hypoglycemia(x)
   THEN take carbohydrate

#4 RULE handle_hyperglycemia
   IF hyperglycemia(x)
   THEN take insulin
```

Now suppose, that Jane suffers from diabetes and uses a RBS with the rule base introduced before. She measures her blood sugar value. Additionally, she inputs in the system that she is pale. The value of measurement is 8,1 mmol/L. This value is added to the fact base. Now the system searches in the rule base for rules whose conditions are getting true now. The condition of rule 2 matches the fact (8,1 mmol/L $>$ 6,9 mmol/L and being_pale). The conclusion of rule 2 (Jane has hyperglycemia) is added to the fact base. Now rule 4 is getting true and the result and advice for Jane is: *take insulin*.

2.6.3 Backward chaining

Backward chaining is *hypothesis driven* or *goal directed* inference; it is the reasoning from the hypothesis back to the facts. To accomplish a goal, the process of backward chaining sets up sub goals and tries to satisfy them [GR05]. The goal or hypothesis is formulated by the user of the Rule-based system. First, backward chaining searches in the rule base for rules whose THEN-part matches the goal. Is such a rule found, and the IF-part of this rule matches facts in the fact base, the rule is fired. Usually the matching facts are not found in a first step. When the needed facts are not in the fact base, the inference engine has to stack the goal, that is, it has to put the goal aside. It sets up a new goal respectively a sub-goal to prove the IF-part of the rule. The inference engine searches for rules that can prove the sub-goal. This process is iteratively repeated until the goal is fired or no rules are found to prove the current sub-goal. This description and the illustration of backward chaining in Figure 2.33 is based on [Neg11].
We also want to give an example from the diabetes scenario. Thereby, we use the rule base introduced in Section 2.6.2 and add to the fact base: blood_sugar_level(Paul) = 2,1 mmol/L. Now we ask: ”Who has to take carbohydrate?” First the system searches for a fact that answers the query directly. But there does not exist one. Then it searches the rule base for rules that would answer the question when their conclusion would be true. The system finds rule 3. Now the system looks for a fact that would fulfill the conclusion of rule 3 (hypoglycemia(x)). It does not find a fact, thus, it searches again for a rule that would give the answer to the question who has hypoglycemia, if its conclusion would be fulfilled. The system finds rule 1. When it searches for a fact, it finds the fact blood_sugar_level(Paul) = 2,1 mmol/L. Now rule 1 can be fired and the fact

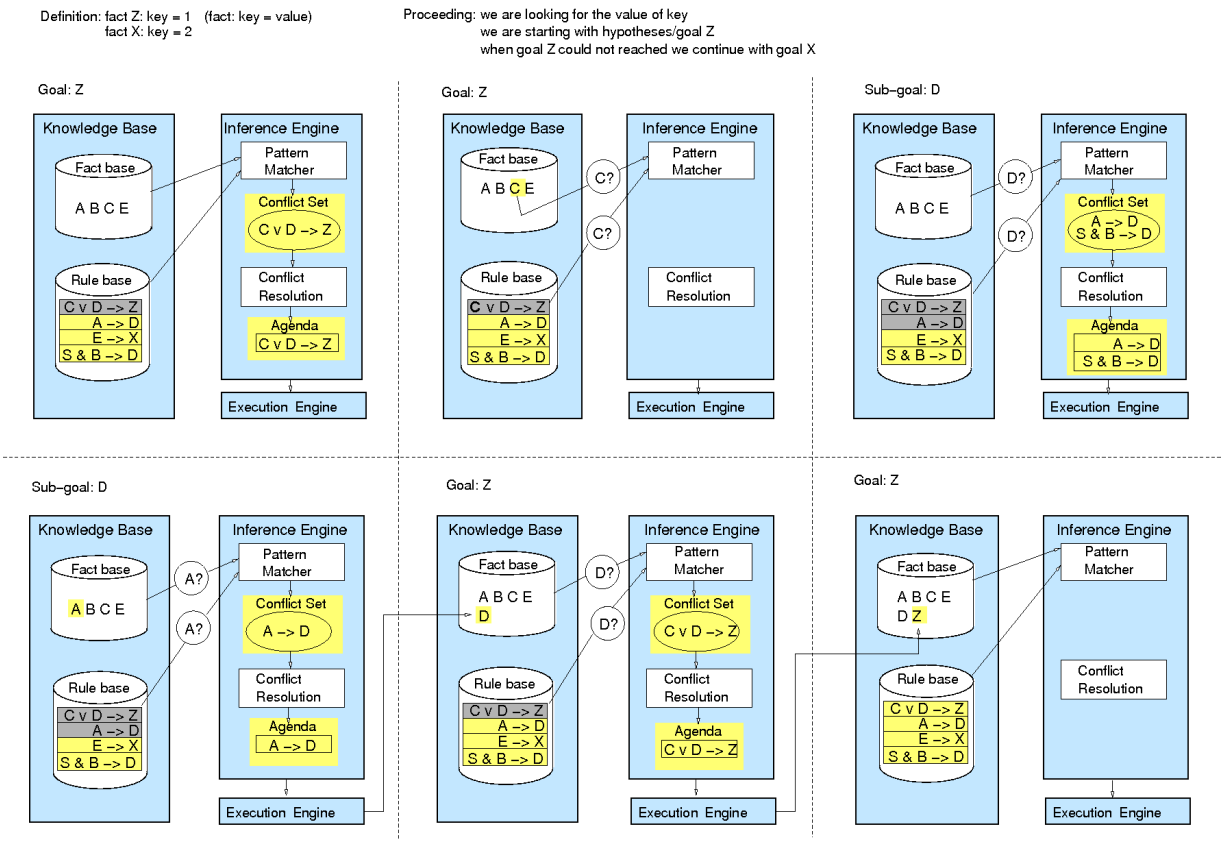


Figure 2.33 – Backward chaining: The inference engine aims to infer fact Z. It searches in the rule base for rules with Z in its THEN part. It finds C v D $\rightarrow$ Z. Fact C and D have to be established. C is in the fact base. Sub goal D is set up. D is not in the fact base but there are two rules with D in the THEN-part (A $\rightarrow$ D and S & B $\rightarrow$ D). The conflict resolution determines A $\rightarrow$ D to be used first. A is in the fact base. A $\rightarrow$ D is fired. D is inferred and now available in the fact base. The inference engine returns to goal Z. C v D $\rightarrow$ Z can be fired and Z can be established. (The figure refers to [AB92, Neg11].)

hypoglycemia(Paul) is added to the fact base. Thus, the initial question can be answered.

2.6.4 Discussion forward chaining vs. backward chaining

Forward chaining matches facts against rules and thereby generates new facts. Backward chaining starts with a goal. It looks backwards for facts that matches rules and therefore satisfy the goal. If needed, it looks for rules that generate the needed facts (satisfying of sub goals). Forward chaining is executed when a new fact is added to the fact base.
Luger [Lug05] states, that the question whether to use forward or backward chaining is determined by the problem itself. Forward and backward chaining have exponential complexity. But dependent on the complexity of the rules, the nature and availability of the problem data, and

the shape of the state space[19] it can be more efficient and therefore faster to use one or the other reasoning strategy.
Luger [Lug05] proposes to use forward chaining in three situations: (i) when it is difficult to formulate a goal, (ii) for solving interpretation problems, when all or most data are given in the initial problem statement and no data have to be acquired, (iii) when the number of potential goals is large but there are only a few ways to use facts of the problem instance. Otherwise forward chaining can cause the generation of a lot of facts that are never needed. Furthermore, according to [Lug05] backward chaining should be applied (i) when a goal is given or can easily be formulated, (ii) when the problem data are not given but have to be acquired to solve the problem, because backward chaining can efficiently guide the acquisition, (iii) when there is a large number of rules that match the facts of the problem. In this case, an early selection of goals can decrease data that have to be searched.
That is, backward chaining uses knowledge of the desired goal to guide the search and reduce the search space. Backward chaining is executed when a goal is formulated. Therefore, there are no superfluous inferences drawn and facts generated. The algorithms used by backward chaining are more complex and causes higher execution costs than that of forward chaining (see [AB92]).

2.6.5 The Rete algorithm

The pattern matcher of the rule engine matches facts against the condition-part of rules and creates the conflict set. Usually, the conflict set has to be recomputed several times. For forward chaining the recomputation has to be done every time a fact is added to or deleted from the fact base. For backward chaining every formulation of a new goal causes a recomputation of the conflict set. Because the process of pattern matching is the heart of the Rule-based system, its efficient execution is extremely important.
The simplest way to recompute the conflict set, is to iterate over rules and facts and to compare the conditions of rules to facts one by one. This approach performs extremely bad for large collections of rules and facts. To efficiently execute pattern matching, pattern matching algorithms, such as Rete [For79, For82], TREAT [Mir87], LEAPS [MBLG90] and Matchbox [Per89] were created. Today, Rete (respectively its successors) is a de facto standard for pattern matching algorithms, used in many established rule-based systems. This is the case although e.g. LEAPS is significantly faster and less memory consuming than Rete [MBLG90]. The problem with LEAPS is that it is difficult to comprehend (and therefore to implement) its data structures and algorithms [Bat94]. Here we will only describe Rete because it is the most commonly used algorithm.

[19] The *state space* is a graph that models a deeper structure of the problem. The nodes represent stages of a problem solution, the arcs represent steps in problem solution e.g. inferences. Solving a problem is the search in the state space for a path to the solution. [Lug05]

Rete was developed in the 1970th for use in production systems. It compares large collections of patterns with large collections of objects. The Rete algorithm is very fast because it is based on two empirical observations [GR05, For79]:

1. Temporal Redundancy: Only a few facts are changed by firing of a rule and only a few rules are affected by these changes of facts.
2. Structural Similarity: Often, the same pattern appears in the left-hand side of multiple rules.

Rete limits the effort for recomputing the conflict set after a rule is fired. Rete avoids a one to one comparing of facts and rules by maintaining the internally state information equivalent to that in the fact base. That is, for each pattern, matching facts are stored in a list. When a fact enters the fact base, matches and partial matches are computed and added to the list. This information is stored until the facts leave the fact base. Thus, the pattern matcher using Rete never has to touch the rule base, but it computes changes to the conflict set out of the small set of recently changed facts.[20] The iteration over the rule base is avoided by using a tree-structured sorting network for rules. (For details see [For82].)

2.7 Summary

In this section we have explained the fundamentals of this work. Today, business processes have to be dynamic as today's business processes are subject to frequent changes. Thus, business processes have to adapt fast, easily but consistently to new situations. Today's workflow management systems do not offer the required flexibility. This work aims to develop dynamic and context-aware workflows that fit the requirements of today's businesses.

A workflow is flexible, if it can adapt its structure to its context. Thus, we started this chapter by explaining the terms context and context-awareness. Afterwards we described and defined workflows in general and explained how workflows are specified and executed by means of workflow management systems. We discussed dynamic workflow management and classified dynamic WfMSs.
WfMSs need computer understandable descriptions of workflows as input to execute the workflows. These descriptions have to be written in a form that can be easily and efficiently analyzed. Petri nets are a modeling language for the description of distributed systems that is specially suited to describe workflows in the needed form. Petri nets have an intuitive graphical notation. They are a formal language and a mathematical theory. Thus, Petri net models can be analyzed by simulation and formal analysis methods. Furthermore, workflows can be easy mapped to Petri nets and are well researched. Therefore, we conclude the chapter by a detailed description of Petri nets, especially Condition Event Nets, Hierarchical Petri nets, Colored Petri nets,

[20] The facts have be changed in the last recognize-act cycle.

and Hierarchical Colored Petri nets.
We analyzed, whether the introduced kinds of Petri nets are suitable for the definition and verification of static and dynamic workflows. Colored Petri nets have the same computational power as low level Petri nets but they add data types to Petri nets. Thus, nets modeled by Colored Petri nets are much smaller and large workflows can be better modelled with them than with low level nets. Hierarchical Colored Petri nets additionally add the concept of hierarchies to CPNs. This concept allows for the mapping of hierarchical workflows. This concept can also be used as base concept for the mapping of dynamic workflows because by adding a dynamic binding of sub nets at run time, it supports the dynamics of workflows we want to provide. Therefore, we decided to build our concept on Hierarchical Colored Petri nets.

Additionally, we described Rule-based systems which will be used to select the sub workflows that adapt workflows at run time in case of context changes.

Chapter 3

Flexwoman — a flexible context-aware Workflow management system

Traditional WfMSs offer a high degree of control over the flow of work, but their ability to react to events that lead to run time changes of the workflow are very limited. The events and the belonging reactions have to be predefined at build time by means of the construct conditional routing (see Sections 2.3.4 and 2.3.5). Unforeseen events cannot be handled by traditional WfMSs. But, usually almost all non trivial workflows are affected by events that require unforeseen run time changes [AEtH03]. Thus, users bypass the WfMS to handle these events. If this is done too frequently, the workflows no longer reflect reality and are of little avail to the users.

In this work, we want to find an approach that can on the one hand handle non predictable events and events for which the time of their appearance is non predictable. These events lead to changes of one or few workflow instances at run time. On the other hand, we want to preserve a large part of the control that traditional WfMSs offer. Furthermore, we focus in this work on processes that are affected by more foreseen than unforeseen changes.

Even though our approach is a generic one, which is applicable for different application scenarios, we choose as the motivating example the healthcare domain. Healthcare is an appropriate application domain for our research, because treatments have to react flexibly to events such as: emergencies, allergic reactions of patients, abnormal laboratory results, and frequent new scientific findings. These special properties of healthcare influence our work.

Another goal of this work is to support workflow based applications that act on mobile devices. In mobile environments the context changes frequently. Thus, adaptations to the current context are especially of great benefit in mobile environments. For instance, for a traveler with diabetes it might be a great gain if the application adapts automatically to the current location of the traveler and gives on request immediately the telephone numbers of the nearest hospitals.

With our work we will design, develop and implement a flexible context-aware workflow management system that automatically replaces workflow parts based on context information. There-

fore, we call the system *Flexwoman* (**Flex**ible context-aware **wo**rkflow **man**agement system). In this chapter we will give a detailed overview of our work. We will first introduce the requirements to our work. Subsequently, we will discuss possible solutions. The ideas and concepts we developed to achieve our objective, and the challenges we were faced with while developing our solution will be described in detail in this chapter.

3.1 Motivation and requirements

Usually, every non trivial workflow is affected by events that require non or not exactly predictable run time changes. Additionally, there is the trend to support domains by WfMSs, which traditionally have non predictable or not exactly predictable business processes, to benefit from the advantages of workflows such as documentation and process automation (for details see Section 2.3.3). Not exactly predictable workflows are workflows with control flow information that are not available until run time, for instance, workflows for which it is not clear, whether or at which point in time a change occurs, and workflows whose structure depends on run time results of the workflow or on external context information. Traditional WfMSs cannot handle such workflows. This limitation results from the architecture of traditional WfMSs. This architecture follows the workflow reference model, which recommends a strict separation of process definition at build time and process execution at run time (see Section 2.3.2). When all possible changes have to be pre-modeled at build time, only changes foreseen at build time, but no changes that occur unexpected run time, can be supported.

When unpredictable changes cannot be handled by a system, but these changes occur, the only way to handle them is to bypass the system. But a frequent bypassing of a WfMS makes workflow management to a burden and in worst case no longer profitable because: (i) the workflows do not reflect reality, (ii) the flow of work is not correctly documented and (iii) the costs for bypassing a system are enormous.

But also the handling of foreseen changes of the workflow as suggested by traditional WfMSs causes problems. Pre-modeling of foreseen changes implies a complete modeling of the alternative routing path for every possible change. Doing this with the constructs available in traditional workflow systems, that is, by control flow patterns such as exclusive choice and simple merge, makes modeling not only time consuming but also leads to a loss of clarity. The lack in clarity is especially a problem, when many alternative routing paths have to be designed. Additionally, applying these constructs means to interweave workflow and routing rules, which complicates workflow maintenance.

The following general requirements result from these general problems:

G1: *Reaction to unforeseen events*: We have to bridge the gap between build time modeling and run time execution, to allow for a flexible reaction to unforeseen events.

G2: *Easily understandable representation*: We have to find an approach that makes the representation of the workflow easy understandable and maintainable.

Furthermore, we do not aim to create an approach tailored for only one application scenario or application domain. Our goal is to develop a generic approach applicable in different mobile application scenarios of different application domains. Nevertheless, we have chosen the healthcare domain as our application domain. This domain is characterized by processes that are potentially life-saving. Thus, mistakes might cause irreparable damages of patients' health state and must be avoided. In critical situations it is necessary that the physician in charge has location independent (authorized) access to the patient record to allow for flexible reactions to unforeseen and foreseen events. This flexible reaction to events has to be added to the *organizational memory*[1], e.g., by means of logs, to allow for i) its reuse in similar cases and ii) its traceability to recognize problems in the process definition. Furthermore, time is a critical factor in healthcare. Therefore, the physician has to be unobtrusively and efficiently supported by automated adaptations of the application to the current context such as location, time, or health state of the patient.

The general requirements (G1-G2) introduced above, are relevant in the context of healthcare. In the following, we set them in the context of healthcare.

G1: *Reaction to unforeseen events* is important, because in healthcare, deviations of the workflow occur often. Without flexibility the user has to bypass the system to perform his/her tasks. The bypassing implies a gap in the documentation of workflows, which is not acceptable because healthcare processes have to be completely documented to understand treatments and activities of health personnel in case of problems.

G2: *Easily understandable representation* is relevant in healthcare, because there is a lack of time. In case of manual adaptations at run time, it is important that the user understands the workflow easily and fast. Otherwise there is a serious risk to introduce errors by the manual adaptation. Furthermore, it should be easy to identify similar cases that can be derived from existing cases. The deriving can avoid errors.

Healthcare has special requirements we have to consider. Thus, the general requirements have to be extended by special, healthcare specific requirements:

S1: *Automatic context-aware adaptation* (including automatic context acquisition and change detection): The system has to support physicians in a time-critical environment, where mistakes have to be avoided. Efficiently executed and automated context adaptations are a good way to support the physician.

[1] See [AH98] for more information on the term organizational memory.

S2: *Correctness*: Healthcare applications have to work correctly, also after automatic or manual adaptations. Any mistakes have to be avoided because they can cause problems to the health status of a patient.

S3: *Efficient execution*: Usually, time is a critical element in healthcare. Therefore, we have to find an approach that is able to perform context-aware workflow adaptations efficiently and quickly.

3.2 Evaluation of flexible WfMS concerning our requirements

In Section 2.4 we described the following approaches of flexible workflow management systems:

- Schema evolution: evolutionary changes of the workflow definition,
- Ad-hoc changes: short term changes of the workflow instance,
- Open point approach: open points of partly defined workflow definitions are completed at run time with workflow fragments,
- Multi instance activity: makes the number of task iterations changeable,
- Late composition: workflow fragments defined at build time are composed at run time to a workflow instance, and
- Generic interface: generic interface for dynamic changes at every workflow task.

In this section, we classify a selection of the introduced approaches and compare them regarding the requirements defined in Section 3.1. As explained before, our aim is to support changes resulting from context changes occurring at run time and affecting only one or few instances. Therefore, *Schema evolution*, which handles build time changes of the workflow definition and affects all workflow instances, is not relevant for this work. We will also exclude *Multi instance activity* from our discussions as this approach solves only a limited problem of dynamic workflow adaptation and usually, it cannot be used alone to apply workflow adaptations. The remaining approaches can be grouped in the following classes, which will be the subject of our comparison:

C1. *Complete workflow defined*: the complete workflow is defined at build time and changed or refined at run time (Ad hoc changes and Generic interface).

C2. *Workflow partly defined*: the workflow is partly defined at build time whereby placeholder tasks mark points to be defined by fragments at run time (Open point approach).

C3. *Only fragments defined*: workflow fragments are defined at build time and composed at run time (Late composition).

In the following, we compare these classes (C1 - C3) regarding the requirements (G1-G2, S1-S3) defined in Section 3.1.

C1: Complete workflow defined at build time, changed at run time:

(G1) This approach can handle foreseen and unforeseen changes. There is no limitation regarding the application of changes to (specially marked) parts of the workflow (so called change regions). Every task can be changed. Manual workarounds are not needed. Thus, a seamless documentation of the workflow can be created.

(G2) There exists a complete, readable definition of the (default) workflow at build time. Known deviations are not pre-defined and cannot be visualized beforehand or used as template for adaptations.

(S1) The adaptations of the workflow are usually managed manually by the user. An automation can be reached by using a rule engine.

(S2) There exist approaches to guarantee correctness of the changed workflow instance. When ad hoc changes are manually applied, checks have to be executed at run time and not beforehand at build time.

(S3) To guarantee correctness, run time checks of run time changes are necessary. These run time checks are complex and time consuming. On the other hand, the approach has the advantage that in case of no changes the workflow is handled like a static workflow and no additional checks have to be executed.

The definition of workflows belonging to that class have to be adapted and checked for correctness only in case of context changes at run time. The approach is suitable and fast for workflows where changes of the workflow structure are the exception rather than the rule.

C2: Workflows with placeholder tasks defined at build time, completed at runtime:

(G1) One approach of this class (Late modeling) allows for a modeling of fragments at run time. This approach can handle foreseen and unforeseen changes. A seamless documentation can be created for the execution of managed workflow instances. Another approach of this class (Late selection) only allows for handling foreseen changes. Thus, the approach Late selection should not be used when a seamless documentation is needed.
In general, the pre-definition of change regions at build time using placeholders reduces

the flexibility of this approach. A way to increase flexibility is the enlargement of change regions. But this enlargement decreases understandability and increases the effort for correctness checks at run time significantly.

(G2) Workflows are partly defined at build time. Abstract tasks are modeled as placeholders for possible changes. At build time only a partly specified workflow definition and not an easily readable complete workflow definition exists.

(S1) The run time replacement of placeholder tasks can be done manually or automated by rule engines.

(S2) There exist approaches to guarantee correctness of the changed workflow instance. If the fragments that replace placeholder tasks at run time are already checked at build time, these checks have not to be executed at run time. This approach saves time at run time.

(S3) Partly defined workflows and the fragments defined at build time are checked for correctness at build time. In case of Late selection, the fitting fragment is chosen from a small selection of pre-checked fragments. Therefore, the replacement algorithms are comparatively simple. The search for suitable fragments starts immediately if an open point is entered, a check for exceptional cases (that causes a run time definition of fragments) can be omitted. In case of Late modeling, the late modeled fragments and affected workflow instances have to be checked for correctness at run time. This requires more complex algorithms and is therefore more time consuming.

In approaches of this class, workflow parts not known at build time are modeled as placeholder tasks, for replacement by concrete fragments at run time. Dynamic changes are only applicable to placeholder tasks. Fewer placeholder tasks require less adaptation work at run time. Therefore, this approach is efficient for workflow definitions with few placeholder tasks.

C3: Definition of fragments at build time, which are composed at run time:

(G1) This approach builds the workflow out of fragments every time it has to be executed. Thus, an extra check for exceptional cases that change the workflow can be omitted. This approach is designed for processes which often change the order of execution of fixed components. Approach C3 can allow for fragment modeling a run time. When fragment modeling at run time is allowed, foreseen and unforeseen changes can be managed, otherwise only foreseen changes can be handled. But only if C3 supports also the handling of unforeseen changes, workflows can be documented without gaps.

(G2) The approach deals with fragments that are newly combined at run time for each workflow execution. Thus, it lacks an easily understandable build time representation of the workflow.

(S1) The composition of the workflow can be done manually but usually an automated composition by AI planners is executed.

(S2) To guarantee correctness, correctness checks have to be executed. The correctness checks of fragments and their combinations should be done at build time if possible. This reduces or completely avoids time consuming correctness checks at run time. Thus, only for unforeseen changes run time checks are necessary.

(S3) For the composition of the workflow complex and therefore time consuming algorithms are needed. If not only the composition is executed at run time, but also the correctness checks, this results in a large overhead at run time.

Class C3 is extremely flexible. It deals with fragments that are for each workflow execution newly combined at run time. It is not limited by a pre-defined change region. It is suitable for workflows where changes of the workflow structure are not the exception but the rule.

Discussion Table 3.1 summarizes the identified classes of approaches for flexible workflow management (C1-C3) regarding their compatibility with our requirements. The classes cover the range of approaches from complete workflows defined at build time, where changes to the workflow structure are the exception, to approaches, where only fragments are defined at build time, because the workflows change that often that a default version cannot be defined.

Column *Flexibility* shows that the flexibility of workflows with placeholder tasks (C2) is lower than that of other approaches, because changes can only be applied to placeholder tasks. Thus, it is the only approach, where changes are limited to pre-defined change regions.

In column *Representation* we analyze whether a user[2] can understand the workflow at build time. C1 (complete workflow) defines a default workflow at build time. Such a workflow is well understandable. The problem of C1 is that also foreseen deviations of the default workflow are not identified beforehand and have to be designed each time at run time (when changes are manually applied). Class C2, which works with placeholders, does not explain how the default workflow looks like, but the foreseen changes can be visualized. The most flexible approach, C3, composes fragments. At build time, it offers only fragments of the workflow. This is difficult to understand for a user.

The column *Automation* describes how the approaches are usually automated. All approaches offer the possibility to manually configure the system. For all approaches the automation of the adaptation is researched. While for the classes C1 and C2 rule engines are the commonly used way to automate it, for C3 AI planners are used to compose the pre-defined fragments. Rule engines are today well researched and optimized. They are usually at run time faster than the AI planners, because rule engines perform the largest part of work at build time and AI planners doing nearly all the work at run time.

[2] the workflow modeler or the manager of a business

The column *Correctness* shows that for all considered workflow classes algorithms exist, which guarantee the correctness of the workflow.

In column *Efficiency* the run time speed is depicted. This column shows that approaches with placeholders (C2) and workflows completely defined at build time (C1) are faster in run time execution than approaches where only fragments are defined at build time (C3). The reason is that the build time definition of workflows saves time consuming composition work and correctness verifications at run time.

Class / property	Flexibility (G1)	Representation (G2)	Automation (S1)	Correctness (S2)	Efficiency (S3)
Complete workflow defined. (C1)	++	+	++	+-	++
Workflow partly defined. (C2)	+	+	++	+	++
Only fragments defined. (C3)	++	+-	++	+-	+

Table 3.1 – Comparison of classes of dynamic workflow management regarding the requirements of our application environment defined in Section 3.1 (++ best marking, - - lowest marking)

Referring to this comparison we decided that class C3 is not convenient for our application scenario because of its representation issues (G2) and the run time composition of fragments (S3). Class C1 lacks predefined deviations of the workflow (G2). This shortcoming causes the devaluation of this class. Thus, class C2 is the approach we use as base for our approach. Its flexibility is not as high as that of C1 and C3 but we focus in this work on processes that are affected by more foreseen than unforeseen changes. Such processes can be handled by C2.

3.3 Concept of flexibility in Flexwoman

The basis for our work is the class of dynamic workflow management that makes workflows flexible by using placeholder tasks (C2). In the following section we derive our approach (Flexwoman) from C2. Thereby, we keep the positive properties of C2, while additionally searching for solutions to the weaknesses of C2. We visualize the differences of C2 and Flexwoman in Figure 3.2.

The positive properties of C2 we will keep (see above), are the following:

- This class offers the possibility to automatically react to context changes by predefined sub workflows that can be chosen dependent on context.
- Additionally, some approaches of this class allow for the reaction to unforeseen events by

the possibility to define of sub workflows at run time. In case of run time definition, sub workflows can be derived from pre-defined sub workflows, or they can be defined from scratch. In case of deriving of sub workflows, methods of class C1 are used for the minor adjustments of the sub workflow (e.g. moving or swapping tasks). The deriving is less error prone than a new sub workflow definition.

- This class reduces errors as it allows for the repeated use of sub workflows to refine different substitution transitions. Thus, sub workflows do not have to be redefined or copied (what causes problems in case of changes at the sub workflow). Additionally, errors are avoided by deriving sub workflows from existing ones in case of unforeseen events (see above).

- This class hides details not needed on a particular level of the workflow, and it hides details that need authorized access from workflow levels. The information can be hidden by allowing for workflow refinement over more than one level (by using sub workflows).

- An efficient execution of predefined sub workflows is offered. They can be chosen at run time and do not have to defined at run time (C1) or composed at run time (C3).

In the comparison in Section 3.2 we found that one main drawback of using placeholder tasks (C2), is, that the placeholders fix the change regions at build time. This limits the flexibility of this approach, because outside the fixed change regions, changes cannot be applied. To improve this, **every task** of a workflow can be replaced at run time, just like an abstract task. That is, for every task there is a list of fragments defined which can replace this task at run time. The fragments will be reusable by other tasks. Another problem of approaches of class C2 is the loss of readability and understandability of the workflow when the number of placeholder tasks grows. Therefore, we propose to inline the default sub workflows in the workflow definition at the highest level. Thus, a **default workflow** or the most likely workflow is available at build time. This eases understandability and readability at build time. The inlining of sub workflows has the drawback that possibly details not necessary for the understandability of the workflow definition at highest level are shown. For the handling of this case the inlining of the default sub workflow has to be avoided, and the default sub workflow has only to be marked in the list of possible sub workflows.

Before showing the visual comparison of Flexwoman and the class of partly defined workflows (class C2) in Figure 3.2, we present the example used in Figure 3.2 as a conventional workflow in Figure 3.1. The representation of the conventional workflow aims to support the understanding of the following comparison. In Figure 3.1 we define a small workflow with a conditional routing path. At run time, one branch (a or d or e,f) is executed dependent on a condition. Afterwards b is executed independent of the branch executed before. Approaches that use placeholder tasks avoid conditional branches. Instead of these branches fragments are defined that can replace an abstract placeholder task (see Figure 3.2(a)). These fragments can not

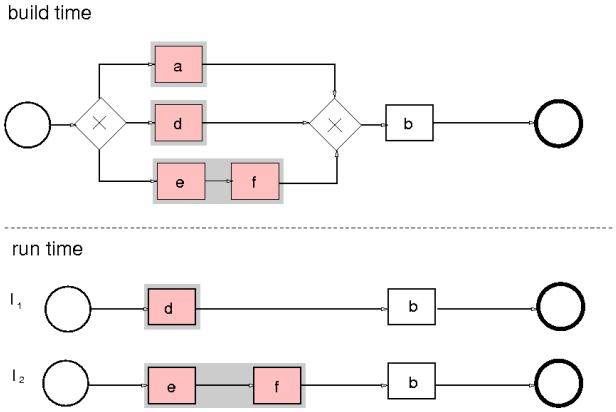


Figure 3.1 – The upper part of the figure shows a conventional workflow definition. The lower part shows examples of workflow instances of this workflow definition (I_1, I_2). Conventional workflows split and merge branches to define conditional routing paths. Usually, there is no way to change the workflow definition when running workflow instances exist and the workflow instance cannot be adapted to unforeseen changes at run time.

only be defined at build time but also at run time. Flexwoman is shown in Figure 3.2(b). In Flexwoman every task can be replaced at run time. Thus, the fixing of the change regions is avoided. Flexwoman explicitly supports the reuse of fragments. A fragment defined for one task can be reused for another task as shown in Figure 3.2(b) for task b. In Flexwoman the default workflow is defined at build time. This workflow is complete, and therefore always readable and understandable.

The definition of workflows with Flexwoman is of similar complexity than that of other approaches that work with placeholder tasks. At build time, the following objects have to be defined: a standard workflow, the fragments that replace tasks at run time, their interdependencies (replacement of which task by which fragment), and the rules for replacing. At run time, the workflow is executed and adapted (to the current context), if necessary. Before a task is executed at run time, the modeler can change existing rules and fragments or add new fragments and new rules that refer to this task. An automated execution of this process requires an explicit definition of rules in a language which the tool that applies the changes understands.

3.3.1 Realization of the flexibility concept

For the realization of Flexwoman we build on a concept traditional workflow systems offer – hierarchical workflows. Hierarchical workflows offer a multi level view on a workflow definition. Sub workflows can detail a workflow on more than one level. Each level offers a more detailed view on the workflow than the level in the hierarchy above it. A sub workflow is part of a larger workflow, the so called initiating workflow (see Definition 7). In the initiating workflow the sub workflow is represented by a single workflow element. This workflow element points to exactly ***one*** sub workflow. At run time, the complete initiating process including the sub workflow is

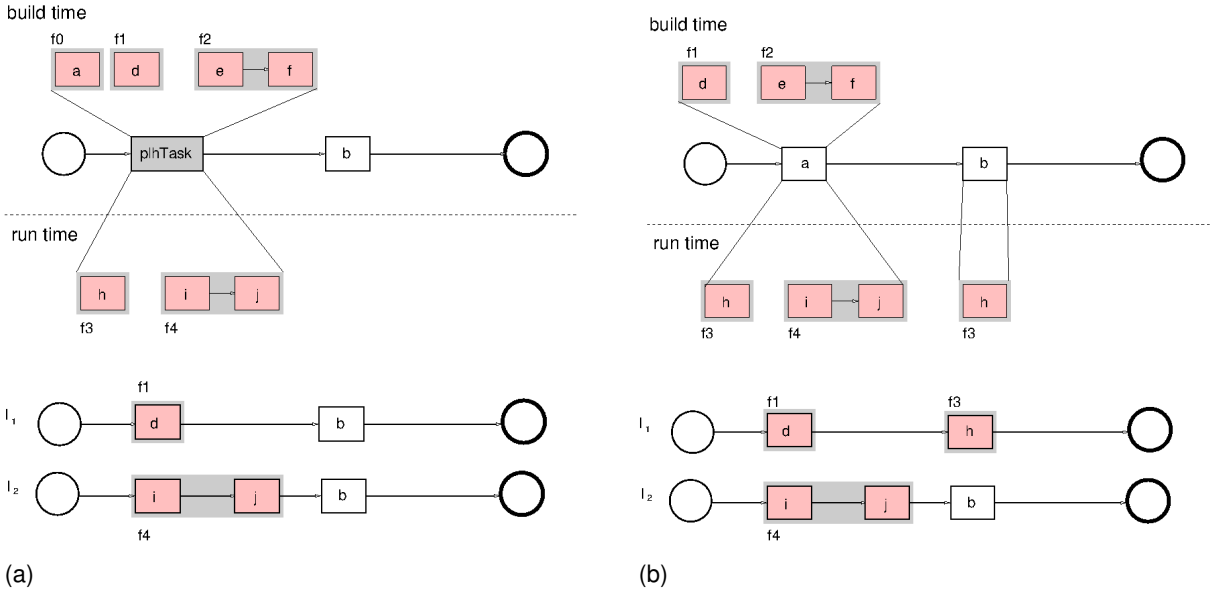


Figure 3.2 – Subfigure (a) shows the workflow from Figure 3.1 as defined in class C2. The fragments f0-f2 are defined at build time, f3 and f4 at run time. f0-f4 are alternative replacements of the placeholder task plhTask at run time. Task b is not marked as a placeholder. It is a concrete task. Thus, b cannot be replaced at run time. Subfigure (b) presents the example workflow described with Flexwoman. Flexwoman does not have placeholder tasks. The most likely fragment (a) is used to define the workflow at build time. At run time it can be replaced by f1-f4. Additionally, for task b fragments can be defined to replace b at run time. Thereby, fragments can be reused (f3).

executed.

We extend this idea and link the single workflow element that represents the sub workflow in the initiating workflow to ***many*** alternative sub workflows (see Figure 3.3). That is, we have several specifications of one sub process element of the initiating process, e.g., in Figure 3.3 task t1 of workflow wf1 has the sub workflows sub1.1 and sub1.2. This proceeding is natural, as usually, a sub process can have different specifications dependent on the current context of the initiating workflow. At runtime one specification is chosen dependent on the context. A further advantage of this procedure is, that for a check, whether the attached sub workflow is valid, the same approaches known from hierarchical workflows can be used. Our approach also realizes two concepts introduced for hierarchical workflows (i) the usage of sub workflows for the refinement of more than one tasks and (ii) the refinement of workflows over more than one level. Both concepts are visualized in Figure 3.3. The usage of sub workflows for the refinement of more than one tasks, is shown by sub workflow sub12.2 that refines task t13 and task t16. The example for the refinement over more than one level is t1 of workflow wf1. It is refined by sub workflow sub1.2. Sub1.2 contains task t13, which is refined by sub workflow sub12.1.

Because the flexibility concept is the key concept of this work, we visualize the main idea of this

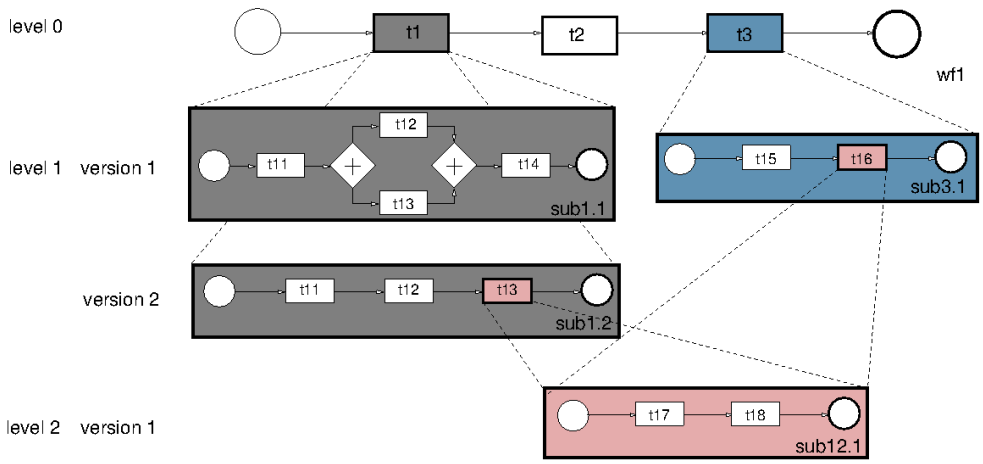


Figure 3.3 – We are allowing for attaching of *several* sub workflows to *one* workflow element on the level of the indicating workflow, e.g. t1 of wf1 has the sub workflows sub1.1 and sub 1.2; t3 has the sub workflow sub3.1. A sub workflow can be used to refine several elements, e.g. sub12.1 refines t13 and t16. Sub workflows can act as indicating workflows, e.g. t13 of sub1.2 is refined by sub12.1. Thus, we are allowing for the refinement of workflows over more than one level.

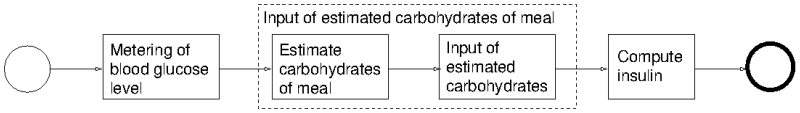
concept in Figure 3.4 with a concrete but simplified example that shows a form of therapy for diabetes (ICT). This example extends the example in Figure 2.3. Its initiating process is *Compute insulin for injection before each meal* (a). This initiating process has two sub processes to input the estimated carbohydrates of a meal – (b): the estimation and the input is done manually by the user and (c): the estimation is executed automatically by a software program that analyses a picture. Dependent on the context, the sub workflow (b) or (c) is chosen. As usually people are not equipped with such a software, the default process is (b). We integrate the default sub process in our initiating process for a better understanding of the process.

3.3.2 Formal description of the main concepts in Flexwoman

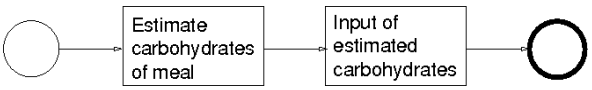
In this section we formally describe the concepts introduced before. The main ideas of Flexwoman were introduced in Section 3.3 and in Subsection 3.3.1:

1. Replacement: A task, or fragment, can be replaced by several different sub workflows.
2. Reuse: A sub workflow can be used by several tasks (fragments).
3. Multiple levels: A workflow can be refined over more than one level.
4. No predefined change regions: Every task in a workflow can be replaced by a sub workflow.

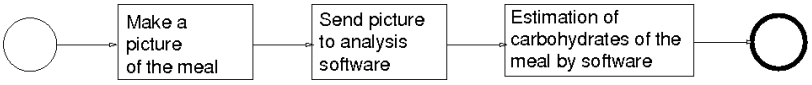
The formal foundation for the implementation of Flexwoman are Hierarchical Colored Petri Nets (see Section 2.5.5). We have to extend the definition of HCPNs to describe Flexwoman. We



(a) Process: Compute insulin for injection before each meal. This process is the initiating process for (b) and (c). The Sub process (b) is the default case and therefore integrated in the initiating process.



(b) Sub process of process (a): Input of estimated carbohydrates of a meal. This is the default sub process.



(c) Sub process of process (a): Estimation of carbohydrates of meal by an image analysis tool.

Figure 3.4 – Process - Injection of insulin before each meal for a simplified diabetes example (according to Intensified Conventional Therapy (ICT)). According to our concept, process (a) has several sub processes, (b) – the default sub process, and (c).

especially refer to the concept of substitution transitions in HCPNs described in Definition 23[3].

As defined in Definition 23, substitution transitions are transitions, that show a less detailed view on the behavior of the net than represented by corresponding subnets. According to idea 1 (idea replacement), we will support not only a single subnet to a substitution transition, but more than one. The assignment from substitution transition and subnet is defined with port-socket relations (see Definition 26). Ports are the interface of the subnet to the environment and sockets are the interface of the substitution transition. A Port-socket relation function assigns port-socket relations to substitution transitions (see Definition 27):

$PS(t) = \{(p, p')\}$, whereby: p is a socket place of the substitution transition t, and
p' is a port place of submodule $SM(t)$ assigned to t. □

We only need to slightly extend the concept of port-socket relations to deal with different submodules. We have to make sure that port-socket relations of different assignments of a substitution transition should not be confused. Thus, we will define port-socket relation functions for a substitution transition t and a sub model sm.

[3]The first paper describing HCPNs [HJS91] additionally introduced the concept of *substitution places* — places that can be replaced by place bordered subnets. The authors expected that both substitution types can be handled in the same way. Lakos [Lak93] disproved that and adapted the channel communication extension of CPN (see Christensen et al. [CH94]) to handle substitution places. The graphical notation of this approach is not intuitive and the handling seems complex. Because we do not aim to work at hierarchies of states, we will skip the discussion about substitution places because it is not relevant for our work. We will focus on substitution transitions.

Definition 32 (Port-socket relation function (t,sm)) . A port-socket relation function $PS(t, sm)$ assigns port-socket relations to substitution transitions (t) and a submodule (sm).

$PS(t, sm) = \{(p, p')\}$, whereby: p is a socket place of the substitution transition t, and
p' is a port place of a submodule sm = $SM(t)$ assigned to t. □

Definition 33 (Port-socket relation function) . A port-socket relation function $PS(t)$ embraces **all** groups of port-socket relations that belong to **one** submodule or in other words, it embraces all port-socket relation function (t,s) ($PS(t, s)$).

$PS(t) = \{PS(t, sm)\}$, whereby: $PS(t, sm)$ are port-socket relation functions
that belong to the same substitution transition (t)
but to different submodules (sm). □

Additionally, there is the submodule function defined for HCPNs (see Definition 30). This function assigns a submodule to each substitution transition ($SMF : T_{sub} \rightarrow SM$), and it allows for the assignment of one submodule to several substitution transitions. For Flexwoman we need a function that allows for the assignment of several submodules to one substitution transition, and that one submodule can be assigned to several substitution transitions. We define the submodule function for Flexwoman as follows:

Definition 34 (Submodule function Flexwoman) . A submodule function SMF assigns several submodules to one substitution transition:

$SMF(t) = \{(t, sm)\}$, whereby: t is the substitution transition and sm is the submodule. □

Idea 3 (idea multiple levels) aims to support the refinement of workflows over more than one level to hide detailed information. This information hiding is useful for a better readability and understanding of processes. HCPNs come with this property by definition (see Definition 31). Our extensions do not touch this property.

Idea 2 (idea reuse) aims to reuse previously defined sub modules. Such a reuse is only possible for sub processes which are not inlined.

Idea 4 (no predefined change regions) aims to avoid the limitation of changes to special predefined regions. In HCPNs every transition can be transferred to a substitution transition.

Definition 35 (Flexwoman) . Flexwoman is a tuple:
$Flexwoman = (SM, SMF_{Flexwoman}, PS_{Flexwoman})$.

1. SM is a finite set of CPN Modules sm, whereby:
   $sm = ((P^{sm}, T^{sm}, F^{sm}, \Sigma^{sm}, V^{sm}, C^{sm}, G^{sm}, E^{sm}, I^{sm}), T^{sm}_{sub}, P^{sm}_{port}, PT^{sm})$.
   It is required that $(P^{sm_1} \cup T^{sm_1}) \cap (P^{sm_2} \cup T^{sm_2}) = \emptyset$ for all $sm_1, sm_2 \in SM$ such that $sm_1 \neq sm_2$.

2. $SMF_{Flexwoman}$ is a submodule function that assigns several submodules and one substitution transition:
   $SMF(t) = \{(t, sm)\}$.

3. $PS_{Flexwoman}$ is a port-socket relation function that assigns a port-socket relation $PS(t, sm) \subseteq P_{sock}(t) \times P_{port}^{SM(t)}$ to substitution transition t. It is required that $ST(p) = PT(p'), C(p) = C(p')$, and $I(p)\langle\rangle = I(p')\langle\rangle$ for all $(p, p') \in PS(t, sm)$ and all $t \in T_{sub}$.

□

3.3.3 Plan of Flexwoman's system model

With our work we design a flexible WfMS, called *Flexwoman*, that adapts the structure of workflows flexibly and in a context-aware manner. This section gives the big picture of the impacts of our concept on the system architecture of Flexwoman. The impacts result from our requirements as described in the following text and lead to additional functionality a workflow system has to offer to be flexible and context-aware[4].

Requirement S1 strives for automatic context-aware adaptation, if possible. Therefore, context has to be sensed, analyzed and published if necessary. Thus, we will need (i) a component for **context management** that senses and stores context and (ii) a component for **context filtering** that analyzes whether context changes influence the structure of the workflow and publishes the changes if they influence the structure of the workflow.

Requirement G1: Reaction to unforeseen events, implies the need to change the workflow at run time, and inform the workflow enactment about this change. Thus, we need an additional component for **workflow adaptation** that (automatically) executes structural adaptations and informs the workflow enactment about the change.

To achieve requirement S2: Correctness, we need a component that executes **correctness checks**. This component checks the resulting workflow instance after applying structural changes and before it is published, for consistency.

In parallel, the execution engine executes the workflow and has to be informed about workflow adaptations during execution. We also offer the possibility to change/add fragments and rules manually at run time. These changes have to be checked before usage, too. Flexwoman can adapt the workflow until all tasks are executed and the workflow is finished.

[4]The requirements G2: Easily understandable representation and S3: efficient execution do not influence the architectural concept at this level of abstraction.

We can formulate a meta workflow for the context-aware adaptation done with Flexwoman, which provides a process oriented view on Flexwoman in addition to the component oriented description shown above. There is one fact in the meta workflow that possibly needs a short explanation: The iterative character of the process is designed by the task *loop*, as Flexwoman aims for avoiding conditional branches in the workflow. The meta workflow is depicted in Figure 3.5.

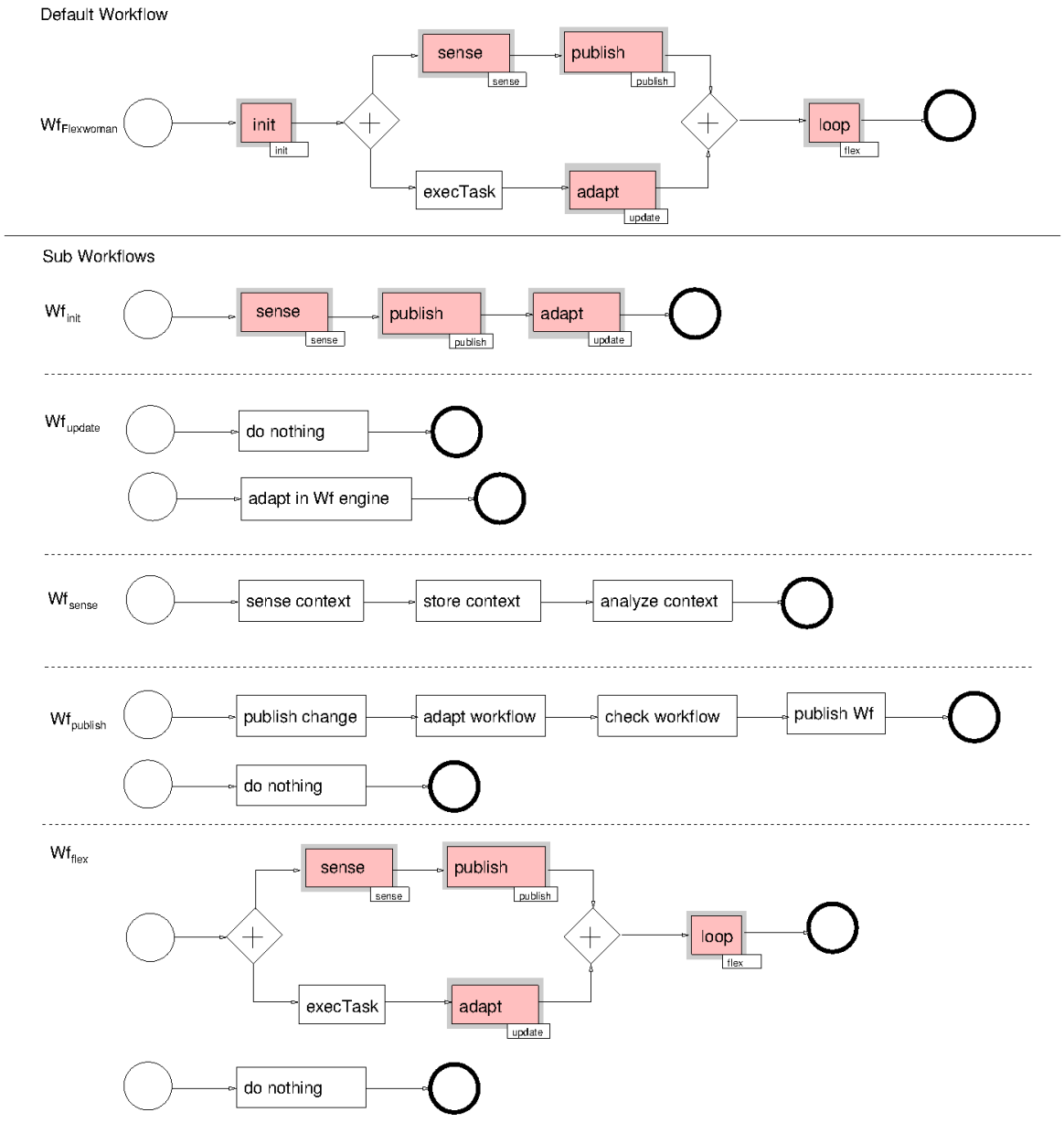


Figure 3.5 – Meta workflow of workflow adaptation with Flexwoman

3.3.4 Structural change operations required of Flexwoman

Flexwoman has to support several operations of structural workflow changes to adapt the workflow to the current context. Thereby, we have to implement three of the main ideas of our concept (see Section 3.3.1):

- Idea insert: A task, or fragment, can be replaced by several different sub workflows.
- Idea reuse: A sub workflow can be used by several tasks (fragments).
- Idea multiple levels: A workflow can be refined over more than one level.

In [WRR07] change patterns are identified. Change patterns describe solutions for commonly occurring changes. We used the patterns as a guide to find the operations of structural workflow adaptation we have to support in Flexwoman.[5]

With our concept, a default workflow is defined at build time and adapted to the current context by exchanging sub workflows at run time. One of the most important operations we have to support is the *exchange of a workflow fragment with a sub workflow*. Thereby, we allow for sub workflows that do nothing, or in other words, sub workflows that process a null operation. Additionally, we want to support the operation *extract sub process*. This operation transforms a fragment of a workflow into a sub workflow, and hence structures the workflow hierarchically to hide information, to increase understandability, and to offer the reuse of workflow parts. In summary, the main operations we have to support are:

1. exchange of workflow fragment and sub process (exchanging of a workflow fragment with a sub process), and
2. extract sub process (extraction of a fragment and the replacing of this fragment by a sub process)

On the other hand, we aim to react to events that cause unforeseen changes. Unforeseen changes are not covered by the operations described above, but require the modeling of additional sub workflows or the adaptation of existing sub workflows at run time. For the adaptation of existing sub workflows, we have to offer ad hoc changes, or rather their generalization to adaptation patterns described by [WRR07]. The adaptation patterns allow for structural changes of a process definition at type or instance level with high level change operations, operations dealing with process fragments and not only with single tasks. In the following text we will describe which patterns we aim to support:
In [WRR07] the following basic adaptation patterns are identified:

[5] The patterns are developed for the usage in flexible process-aware systems (PAISs). PAISs are a superset of workflow systems.

1. insert process fragment
2. delete process fragment
3. move process fragment
4. replace process fragment (by another process fragment)
5. swap process fragment

All these basic patterns have to be supported in our approach.

Another kind of adaptation patterns affects the levels of a workflow: beside the extraction of a sub process introduced above (2), there is the operation:

6. inline sub process (sub process is dissolved and directly embedded in parent schema)

We will support the pattern *inline sub process* too. It is usually used for improving the workflow structure, e.g., when too many hierarchical levels exist. It is not needed to manage unforeseen events. We will support the operation for completeness, especially for handling modeling errors.

Furthermore, there are patterns concerned with the adapting of control dependencies:

7. embed process fragment in a loop (add a loop construct around existing process parts)
8. parallelize process fragments
9. embed a process fragment in a conditional branch (the process fragment should be only executed when special conditions are met)
10. add control dependency
11. remove control dependency

With our concept we try to avoid conditional branches and therefore loops as used in traditional WfMSs. We avoid traditional conditional branches and loops by selecting sub workflows depending on context[6]. The context-aware selection has to be done by manually defined or learned rules. Patterns for adapting control dependencies (without parallelizing) can be applied by rules. Parallelization of process fragments will be supported without using rules. We will implement all of the control dependency patterns for sub workflows.

Furthermore, a pattern for changing transition conditions exists:

12. update condition (update transition condition)

[6] For the implementation of loops we use self referencing sub workflows in combination with non self referencing sub workflows.

This pattern can and will be implemented by using rules.

When applying change operations to a workflow, there is the problem that errors can be introduced to the workflow. The errors introduced can be syntactic errors, e.g., a task is not connected with the rest of the workflow, or semantic errors such as while one part of the workflow is changed, another part of the workflow is changed in a way that results in an inconsistent workflow, or duplication of work, skipping of tasks, deadlocks, livelocks [vdA01]. The latter errors are much more difficult to identify. They are known as dynamic change bug ([EKR95], [vdA01]) or dynamic change problems [RRD04]. A detailed discussion of these challenges is out of focus of this thesis but influence our decisions regarding the implementation of Flexwoman.

3.4 Summary

In many application domains a default workflow definition exists and necessary adaptations change the workflow instances only slightly. This is also true for processes we are faced with in our healthcare application scenario. That is, the high flexibility, WfMS of class C3 (only fragment defined at build time) offer, is not needed. The generation of every workflow instance on the fly at run time is an overhead. A high speed at runtime and a good representation of the workflow at build time is preferred. Therefore, fragment composition at run time is not suitable for our work. Workflows of class C1 either do not store workflow adaptations or use log files to trace the adaptations. When adaptations are not traced, expertise is needed to do adaptation work and time as well as knowledge is lost. When adaptations are traced, analysis tools are needed to extract and reuse adaptations done before. Thus, we decided to use class C2 (Workflow partly defined) as base approach for Flexwoman and improve it to gain better readability, flexibility and speed.
Flexwoman allows for a replacement of every task in the workflow at runtime, just like class C1 (Complete workflow defined). The replacement of tasks depends on context changes. The changes trigger rules that cause context adaptations. The user models the standard workflow, that is, the workflow that is usually executed. To handle foreseen events, alternatives for a task are modeled at design time. To handle unforeseen events, the alternatives for a task are modeled at runtime, before the execution of the task. During workflow execution the context of the workflow determines whether a task is replaced and, if necessary, by which alternative. The workflow context determines the workflow structure. We aim to acquire context information and apply corresponding workflow changes to the greatest possible extent automatically.
For the realization of our concept we use Hierarchical Colored Petri nets. We offer the (virtual) possibility to attach more than one subnet to a substitution transition. The selection of the relevant subnet is described with rules and executed by a Rule-based system. The process execution engine works with regular Hierarchical Colored Petri nets and is not affected by the virtual attachment of more than one subnet for a substitution transition.

Chapter 4

Architecture and Implementation

In this chapter we will describe the architecture of our context-aware dynamic WfMS Flexwoman in detail. We will give insight into the implementation of the components of Flexwoman, their interfaces and their interaction.
We start with the description of the architecture, followed by a description of the interaction between the components of our system. After this big picture, we describe the main components in detail. Therefore, we explain which tools we use for the realization of the components, how we choose the tools, and we explain the implementation of the components. We finish the work with a description of the overall process, the integration and the interfaces of the components.

4.1 Architecture

This section gives an architectural overview of Flexwoman. In Section 3.3.3 we identified that the following components have to be added to a conventional WfMS to fulfill our requirements: (i) context management, (ii) context filtering, (iii) workflow adaptation, and (iv) correctness check. We described the components and their interplay in a high level view. This section concretizes the high level view of the system components and Figure 4.1 illustrates our architecture[1].

The component *context management* takes over the tasks: i) context sensing, ii) context storing, and iii) context change publishing. In our system we call the component **Context Collection Tool**. It consists of **Sensors** that sense the context, **Adapters** that unify the different formats of sensor data and a **Context DB** that stores the unified context. After context storing, publishing of context changes is triggered by database triggers.

The tool *context filtering* subscribes for such context changes and analyses them by means of a rule based system. We call this tool **Context-Change Analysis Tool**. In a first step,

[1] Figure 4.1 refers to the workflow reference model introduced in Section 2.3.2. The workflow reference model describes the components and interfaces that should be supported by a conventional workflow management system.

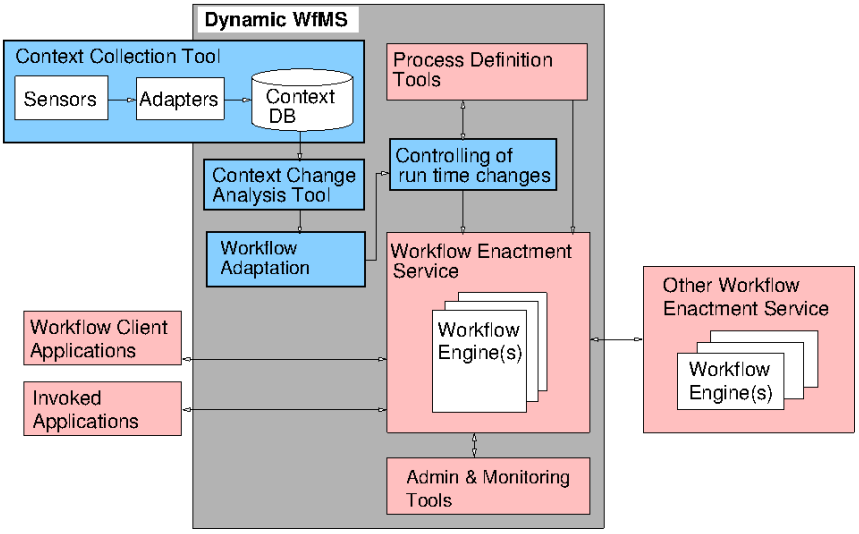


Figure 4.1 – Concept of extensions needed to make a WfMS context-aware and flexible. (The blue color marks the components added to the workflow reference model.)

the tool filters out relevant changes that require a structural change of the workflow and categorizes them. In a second step, the tool triggers the actions for workflow adaptation. The actions differ dependent on their categorization.

The ***Workflow Adaptation*** component adapts the workflow by executing the actions triggered by the Context-Change Analysis Tool. The adaptation to context changes comprises i) the context-aware refinement of a task to a sub process, and ii) the context-aware exchange of a sub process, that is the context-aware replacement of a sub process by another sub process.

The component *Correctness Check* is needed to execute correctness checks of structural changes to the workflow before they are published to the workflow enactment. This component checks workflows after applying foreseen and unforeseen changes. It is especially important that interfaces of refined or exchanged sub processes fit to the respective interfaces of the initiation process. In our architecture, we call the component **Controlling of runtime changes**. After finishing its work, it notifies the workflow enactment about the changes.

Beside the extension of the architecture by additional components, some of the original components of the workflow reference model have to be extended. The Admin and Monitoring tools have to be extended to monitor how context changes influence the workflow structure. The Process Definition Tools have to support the run time modeling and changing of sub processes and rules to handle unforeseen events.

Figure 4.2 shows the messages sent between the components but only for the part of the architecture that is relevant in this thesis.

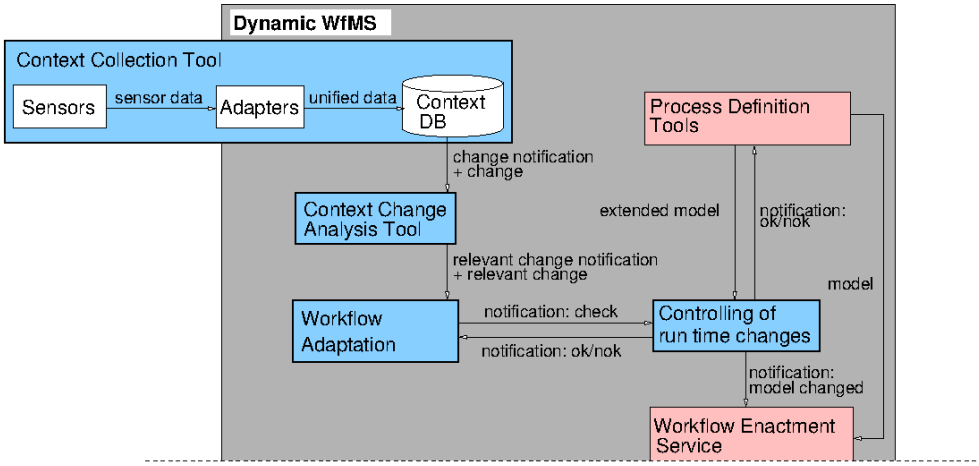


Figure 4.2 – Detail of Flexwoman's architecture with the messages sent to adapt a workflow.

4.2 Implementation

In this section we describe the implementation of the system components that implement context management, context filtering, workflow adaptation, and correctness checks. Figure 4.3 is a system overview that complements Figure 4.1 by showing which tools we used to implement the components.

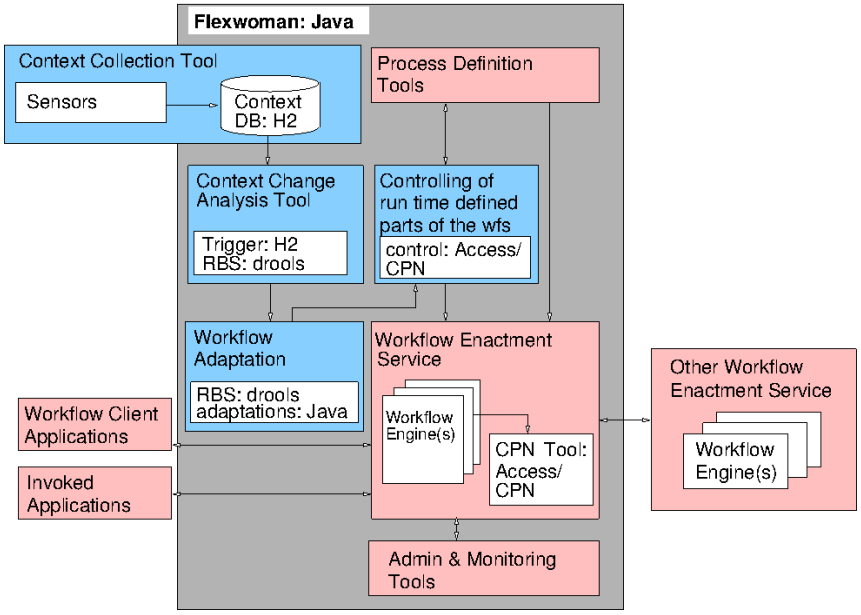


Figure 4.3 – Architecture with the tools/programming languages used for the implementation.

4.2.1 Context management

Context data is data that characterizes a situation. When this data is used for a software system, firstly, it has to be sensed by sensors. We will support different classes of sensors, *hardware sensors* just like devices for blood sugar metering or cameras, *software sensors* like Web services that compute the carbohydrates of a meal by a photography, and *humans* that input data to the system, e.g., their individual bread exchange coefficients. The sensor outputs have a wide range of different file formats. For an efficient analysis of the context data coming from different sensors, the data has to be stored in a uniform way, independent of the sensor they are sensed with, the aim they are sensed for, and independent of their complexity. After unifying, data can be handled in the same way while publishing and querying.

For a uniform storage of the context data, we need a database schema that handles all context data in a uniform way, and we need adapters that transform the heterogeneous sensor data into this schema. In the LoCa-project we created a database schema (see [FMM+09]) that follows the context definition of Dey [DA00]: *Context is any information that can be used to characterize the situation of a subject. A subject is a person, place, or object that is considered relevant to the interaction between a user and an application....* The central entity in this schema is **context object**. Context objects are the sensed data. The data can be combined of different values, e.g., GPS data are a combination of longitude, latitude, altitude, speed, and bearing. The relationship **logical combination** offers the creation of these composed context values. The entity **data generator** describes where the data is coming from, and the entity **mode** allows for determining the frequency of the data sensing. In this thesis, we will use a similar, slightly optimized, schema (see Figure 4.4).

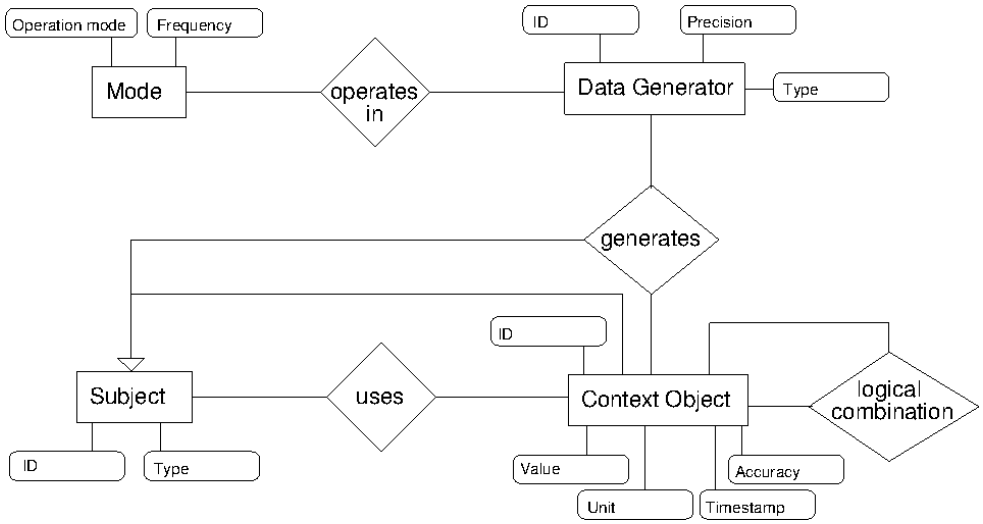


Figure 4.4 – The generic database schema of Flexwoman that implements the context definition of Dey [DA00].

The task of transforming the different sensor outputs into unified data stored in the introduced database schema is performed by the adapters of the Context Collection Tool. For every sensor with a proprietary file format and for every standardized format of used sensors one adapter is needed.

Our generic database schema can be implemented for data stores of different paradigms. Flexwoman has the following requirements that influence the choice of the database:

- Flexwoman aims for supporting applications running on mobile devices that have limited resources.
- Flexwoman acts in a Java environment.

The LoCa project had similar requirements, thus we will rely on a benchmark we made for this project (see [FRMS10]). The benchmark compares an object oriented data base, relational data bases and an RDF triple store based on the database schema we developed in [FMM+09]. We researched two different settings: i) embedded context databases directly running on a mobile device, and ii) context databases that are accessed from a mobile device. Not all of the evaluated databases support both modes. The background of the benchmark is an eHealth use case. The test load was generated from statistical data of a medium sized hospital for the period of a week, a quarter and a year. The queries we used in the benchmark reflect how mobile users and context-aware applications interact with the context database in the eHealth use case. The benchmark showed that the performance of the researched object oriented database was bad compared to the other evaluated data stores (in embedded and in client/server mode). Additionally, the performance of the triple store (only embedded mode) is only sufficient for small data sets. From the relational databases (H2, SQLite, mySQL) we compared, the open source Java database H2[2] wins over with a good performance, default support of referential integrity, working in embedded and client/server settings as well as least configuration problems. Furthermore, the footprint is small (the jar file size is around 1,5 MB).
We decided not to use the features of semantic data. Hence, we chose the relational databases H2 for Flexwoman. H2 has the additional advantage of supporting triggers written in Java. The support of triggers allows for an uncomplicated recognition of (relevant) changes in the database and a smooth integration in the Java based environment of Flexwoman. As H2 is a SQL database, the context data adapters transform the different sensor data to SQL statements.

4.2.2 Context filtering

In a mobile environment, many kinds of context data change frequently. Not all of these changes are relevant enough to cause workflow adaptations, e.g., context changes within a context range

[2] http://www.h2database.com/html/main.html

or changes in too short time periods may be irrelevant. The first step for the filtering of relevant context data is to get the information about changes of the context. We use database triggers to inform the Context Change Analysis Tool about changes in the database. The Context Change Analysis Tool categorizes the changes, based on rules, to allow for a sophisticated adaptation of the workflow based on context changes. Thereby, the database trigger acts as Observerable (observed object) and the class that fires the rules is implemented as Observer (observing object) (see Observer Pattern in [VHJG95]).
In Flexwoman we used JBoss drools as rule-based system that categorizes context changes and triggers context adaptations. We chose drools because this Java based system is an open source tool that is stable, easy and intuitive to use, and has the advantage of a big and active community. Therefore, it fits best in the Java based environment of Flexwoman. For a detailed evaluation of Java based rule engines see [Mül11]. Drools allows us to formulate rules in a simple format:

```
rule "my rule"
    attribute = value
when
    conditions
then
    actions/consequence
end
```

The structure of the rule is nearly self-explanatory. The rule starts with the keyword `rule` followed by the name of the rule. The name is used for a unique identification. Then optional attributes follow. After the keyword `when` the conditional part of the rule is given and the last part of the rule contains the code that will be executed. The keyword `end` completes the rule.
There are a lot of rule attributes that can be used in drools. We will only explain the attributes we will use in this document (For further details see [Tea13].):

- salience: sets the priority of a rule. The priority is used in the conflict resolution of drools.
- activation-group: when multiple rules belong to one activation group, the firing of one of these rules cancels the activation of the other rules belonging to this group.
- no-loop: no-loop=true avoids looping of rules caused by rule changes in the action/consequence part of a rule.

Drools implements ReteOO, an extension of the efficient pattern matching algorithm Rete (see Section 2.6.5), and starting with drools6 the Phreak algorithm. Drools using Rete performs well for large data sets, but it performs even better for large data sets with Phreak.[3]

[3] Phreak takes all of the existing drools code that implements ReteOO plus its enhancements and extends this code to address the issues of Rete. While Rete is eager and data oriented, Phreak is lazy, goal oriented, and aggressively delays partial matching. Phreak is actually the default rule algorithm. [Tea13]

Flexwoman is a generic framework for the dynamic and context-aware adaptation of workflows. It does not come with rules that are tailored to special context data or a special application scenario, or application. The rules for context categorization and context adaptation triggering have to be manually created.
In the following we give an example how rules can be defined for our application scenario. We start with a rule for the categorization of context. In our application scenario, we differentiate between three categories of blood sugar values. Blood sugar values can be too low ($<$2.78 mmol/l), too high ($>$6.9 mmol/l and $<$16.65 mmol/l) or much too high ($>$16.65 mmol/l). We show a rule that checks whether the new sensed context value is much too high. Is this the case the rule sets the status of the message to MUCH_TOO_HIGH.

```
%CATEGORIZATION RULE
rule "much_too_high"
no-loop
    when
        m : Message( myMessage : message , Float.parseFloat(myMessage) > 16.65)
    then
        m.setStatus( Message.MUCH_TOO_HIGH);
        update( m );
end
```

After categorization, the context filtering component starts the workflow adaptation process by means of rules. The adaptation rules differ dependent on the categorization. In a simplified example, the process could print the message that the diabetic has to take dextrose in case of too low blood sugar, in case of too high blood sugar it could print the advice to inject insulin and compute the needed insulin. In case of a much too high blood sugar level the process could trigger an emergency call. The rule *much_too_high_adapt*, which checks whether the status of the message is MUCH_TOO_HIGH, is shown below. The rule triggers the workflow adaptation in its action part. In our concrete example rule, the action part triggers the creation of a subnet for the transition *takeAction*. The .cpn file *callAmbulance.cpn* contains a pre-defined sub module that calls an ambulance. The sub module is attached to the transition *takeAction*.

```
%ADAPTATION RULE
rule "much_too_high_adapt"
no-loop
    when
        m1: Message( status == Message.MUCH_TOO_HIGH, myMessage : message )
    then
        CreateBloodsugarChange cd = new CreateBloodsugarChange();
        cd.createSubNet("takeAction", "callAmbulance.cpn");
end
```

In summary, the context filtering process is divided into two parts: i) filtering and categorization of relevant data with categorization rules, ii) triggering of workflow adaptations dependent on the categorization with adaptation rules. In principle both parts, and consequently the belonging rules, could be merged into one. In practice, the dividing of the process eases its understandability and maintenance.

4.2.3 Workflow adaptation

The workflow adaptation component provides the methods for the adaptation of a workflow according to a context change. The configuration of the adaptation (which substitution transition is substituted by which sub module) is determined in the adaptation rules of the Context Change Analysis Tool. The concepts of the methods or operations we aim to support with our workflow adaptation component are described in Section 3.3.4. The most important operations we aim to support are those operations which mirror the ideas of our approach. The main idea is to use the hierarchical constructs of workflows and to attach a set of sub workflows to a fragment of the initiating workflow. Whether the fragment or one of the attached sub workflows is executed depends on the current context of a workflow instance. Therefore, the main operation we have to support is the following: *exchanging of a fragment in the initiating workflow and a sub workflow*. Furthermore, the "helper" operations: *attaching a sub workflow to a fragment* and *deletion of one or all attachments* are needed to configure the sub workflows that are chosen dependent on the context.

Tools

The implementation of the exchange operation depends on the formalism but also the tool used for process execution. In Flexwoman the adaptations of a workflow are based on the adaptations of Colored Petri nets. We use the CPN tool Access/CPN[4] as base for our work. We choose it based on an evaluation of Petri net tools, we conducted in this thesis (see Appendix A for the complete evaluation). Access/CPN was created in the context of the CPNTools project[5] and depends on the simulator of CPNTools. Thus, before we describe the implementation of the exchange operation in detail, we shortly describe CPNTools and Access/CPN.

CPNTools: The open source tool CPNTools was created in the group of the founder of CPNs Kurt Jensen in the CPN Group at Aarhus University (Denmark). It is the standard CPN tool, now maintained by the AIS group, Eindhoven University of Technology (Netherlands)[5]. CPNTools offers the possibility to visually model, simulate and analyze hierarchical CPNs. It is stable and has a large and active community. The CPNTools simulator is written in SML, a functional programming language basing on Standard ML/NJ. The GUI is written in Beta.

[4] http://cpntools.org/accesscpn
[5] http://cpntools.org/start

Access/CPN: The open source tool Access/CPN[4] was created as a Java interface to CPNTools. Access/CPN is targeted at developers, and not at end users. It offers the possibility to model, simulate and analyze CPNs programmatically in the programming language Java. It is based on the Eclipse Modeling Framework (EMF). The models generated with Access/CPN can be imported in and visualized by CPNTools. It can also be used as stand-alone tool, without GUI [WK09, Wes11]. In this case it also uses the simulator developed for CPNTools (see Figure 4.5). This fact makes Access/CPN a stable and elaborated tool. On the other hand, error messages coming from the SML simulator can be difficult to interpret.

Implementation

Access/CPN offers the functionality on which we can base the development of our dynamic and context-aware WfMS Flexwoman. Especially important is the fact that Access/CPN allows us to execute Colored Petri Nets and change their behavior at runtime. We can model and manipulate all elements of a Colored Petri Net at run time, for instance, places, transitions, sub modules. After applying changes, the syntactic correctness of the net can be checked to avoid problems with CPN execution at run time.
Thereby, Access/CPN offers the possibility to *manually* apply changes to Petri nets at run time. This is the foundation of the *automatic* adaptation of Petri nets with Flexwoman at run time. Additionally, Access/CPN as a native Java tool makes it easy to react to arbitrary events that trigger structural adaptation to a Petri net. Thus, Access/CPN offers an API for the implementation of the exchange operations we aim to support.
We encapsulate the available methods offered by the API and offer an easier to use interface for implementation of workflow adaptations. The interface is generic and therefore available for every kind of fragment and every kind of sub workflow. The operations offered by the new API can be used just like building blocks. If the interfaces for the adaptations are correct, these building blocks can be arbitrarily combined. For instance, it is possible to replace a fragment

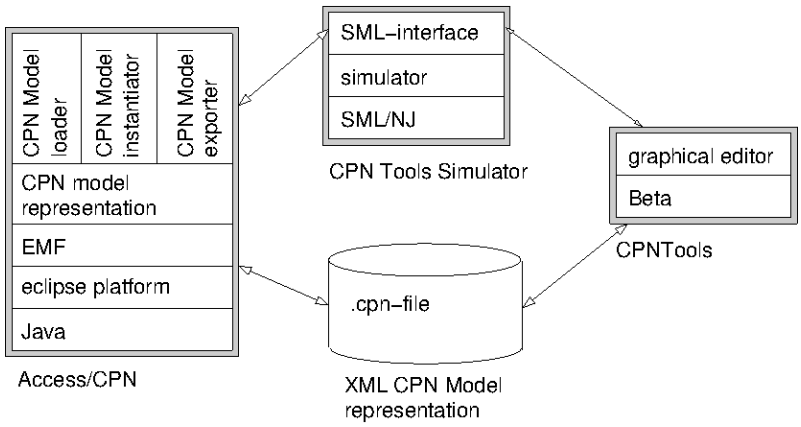


Figure 4.5 – The interplay of Access/CPN and CPNTools. (The figure refers to [WK09, Wes11].)

with another fragment and subsequently replace the second fragment with a sub workflow.

4.2.4 Correctness checks

While manipulating workflows, many mistakes can happen (interfaces of substitution transition and sub module do not fit, the data flow is destroyed, the past is changed, ...). Such mistakes are especially problematic when a running workflow is changed because a broken workflow can stop work in a company, costs time and can lead to an immense loss of money. Access/CPN offers e.g. checks for declarations, pages and markings, that can be applied to a running Colored Petri net. We apply the needed checks when implementing the adaptation functionality of Flexwoman. For instance, for an adaptation operation that replaces a fragment with a sub process, the interfaces of the fragment to be replaced and the sub process that replaces the fragment must be compatible. This means the interface of the fragment (socket place) and the sub workflow (port place) have to be compatible regarding data types and port type (input, output, input/output), otherwise the checks fail and the operation is not executed. After successfully applying these checks, the model is syntactically correct and execution will not fail. Thus, the execution engine is informed about the changes by Access/CPN.

4.3 Summary

In principle there are two processes we have to deal with in this work. The first process is the regular execution of the workflow instance. The second process is the adaptation process that adapts the workflow structure to the current context.

Our work is focused on the adaptation process. The functionality used to adapt workflows is implemented as building blocks in the programming language Java. The building blocks can be arbitrarily combined, provided that the interfaces fit to each other. The building blocks are implemented on the top of the Access/CPN tool and extend it by a simplified API that allows the user to easily and intuitively apply changes at workflow run time.

Flexwoman does not only cover the design and implementation of the needed adaptation functionality, but offers a whole infrastructure framework for run time adaptation of workflows. It consists of four main components: context management, context filtering, workflow adaptation and correctness check. For the implementation of the components we used existing tools: the database H2, the rule engine drools and the CPN tool Access/CPN. The used tools have in common that they are open source tools that are widely used and stable. If there is implementation work to do to interconnect the tools, this is done with Java. A big challenge for the implementation was to pay attention that the adaptation does not disturb the regular execution (e.g., slow it up, or breaks execution).

Chapter 5

Evaluation

In this chapter we evaluate Flexwoman. We start with a qualitative evaluation that evaluates whether our concepts are meaningful and whether Flexwoman implements our concepts correctly. For instance, the qualitative evaluation aims for showing that the flexible reaction on foreseen and unforeseen changes is meaningful in a WfMS and it aims for proofing that Flexwoman offers this functionality. Subsequently, we evaluate the system and its implementation quantitatively. For this evaluation we generate rules to find out how our system manages a growing number of rules. We finish this chapter with further evaluations that check what happens in potentially critical situations[1]. Quantitative and further evaluations check the system implementation.

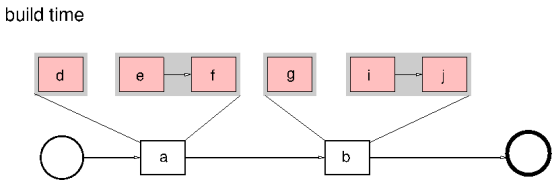
5.1 Qualitative evaluation

In Section 3.3 we explained the ideas and concepts of Flexwoman. In summary, the main goals are:

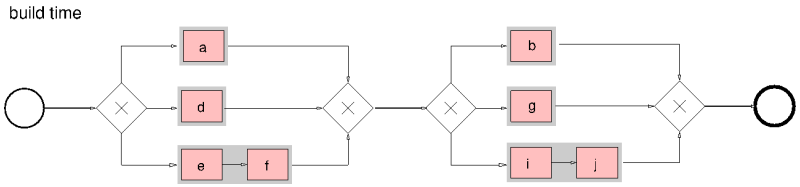
- In case of foreseen changes we want to hide complexity (see Figure 5.1).
- In case of unforeseen changes Flexwoman makes it possible to model sub workflows at run time (see Figure 5.2).
- In case of unforeseen changes we can apply run time changes to every task of the process (see Figure 5.3)[2].

[1] In this context, the term critical situations means adaptations that can potentially stop the execution of the running system.

[2] In this figure, we do not compare Flexwoman to traditional WfMS just like we did it in the examples above as they do not provide changes at run time. However, we can and do compare Flexwoman to open-point approaches that fix the change region in advance at build time.



(a) In Flexwoman choices are hidden as sub workflows.



(b) In traditional WfMS choices are modeled in the main workflow and cannot be hidden.

Figure 5.1 – Figure (a) shows Flexwoman and its way to hide choices. Figure (b) shows the traditional way to model workflow choices and its implications to complexity.

In this section we will show that Flexwoman achieves these pre-set goals.

5.1.1 Evaluation setup

Quick summary of main terms needed in the evaluation

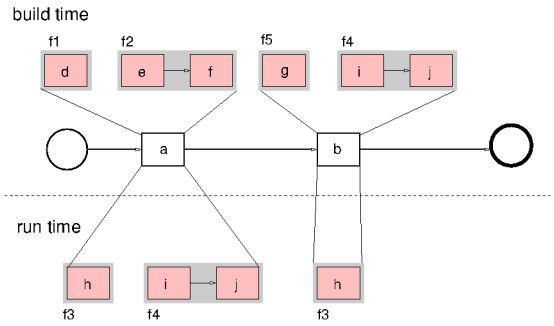
To make the understanding of the evaluation easier, we remind the reader of the following concepts: **task**, **fragment** and **sub process**. A **task** is a logical step in process execution (see Definition 6). A **fragment** is a sub process that is inlined in an initiating process. A **sub process** is a part of a larger (the initiating) business process (see Definition 7). A sub process has a place holder in the initiating process, the so called substitution transition that links to the sub process.

Adaptations

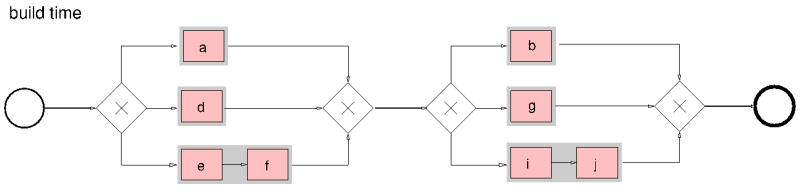
There are three elements that can be changed to adapt a workflow:

- the **process (with simple tasks and fragments)** that should be adapted,
- the **sub processes/fragments** used for the adaptation, and
- the **rules** used for adaptation.

For a flexible adaptation to *foreseen changes*, it is enough to define a process partly, and the



(a) Flexwoman allows for a modeling of sub workflows at run time.



(b) In traditional WfMS choices are modeled at build time. A modeling of sub workflows at run time is not provided.

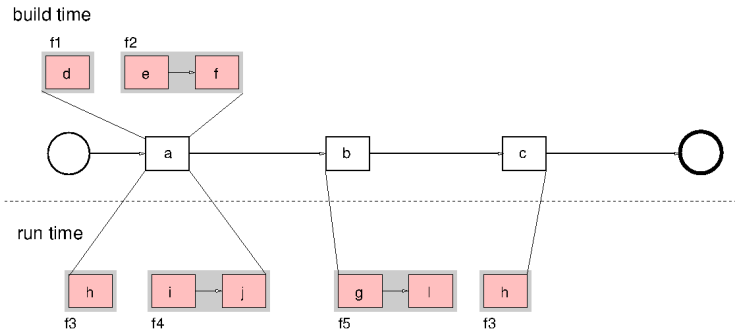
Figure 5.2 – Figure (b) shows Flexwoman and that Flexwoman allows for the modeling of sub workflows at run time and (a) shows that traditional WfMS does not offer this possibility.

sub processes/fragments and rules at build time. At run time there is no definition work to do, the process is executed and adapted if needed.

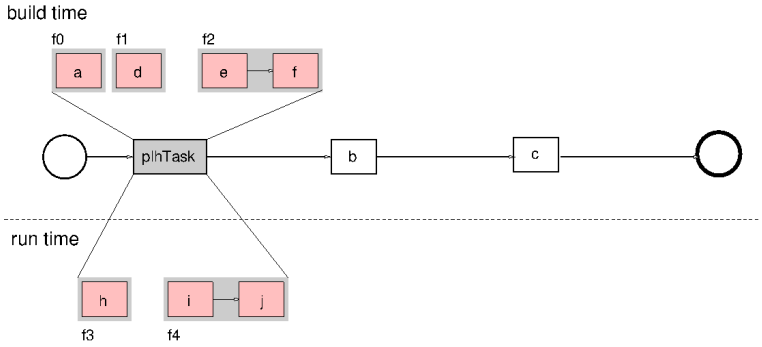
For a flexible adaptation that reacts on *unforeseen changes*, the process has to be defined partly at build time. At run time sub processes/fragments and rules have to be defined or/and adapted. An adaptation is only possible, when the tasks/fragments of the process that undergo a change are not already executed or are in execution.

We support the following operations for workflow adaptation and evaluate them in the remainder of this chapter:

- substitution transition and sub processes
  - replace a sub process of a net by another sub process
  - delete a sub process, that is, replace a substitution transition and the corresponding sub process by a simple task
- fragments (including simple tasks)
  - replace a fragment by another fragment
  - replace a fragment by a substitution transition and sub workflow
  - delete a fragment



(a) Flexwoman allows to replace tasks or larger parts (b,c) of the workflow by sub workflows defined or attached at run time (f5, f3).



(b) In open-point approaches the change regions are fixed in advance at build time. A modeling of new change regions at run time is not provided.

Figure 5.3 – Figure (a) shows that in Flexwoman every task or fragment can be replaced by a sub workflow defined at run time just like in systems that support ad hoc changes. Figure (b) shows that in open-point approaches change regions of a workflow are fixed at build time and cannot be changed at run time.

- combinations of adaptations
- rules
  - add a rule
  - change a rule
  - delete a rule.

Running example

Figure 5.4 shows our running example. The example is derived from the example used in the chapters before. We had to adapt the original example slightly to be able to evaluate all the adaptation operations we listed above.

In the running example the blood sugar is metered and printed, and afterwards another action will be executed. Which action will be executed depends on the context:

In the simplest case the action *react on bloodsugar* is executed. This action does not more than printing an advice.

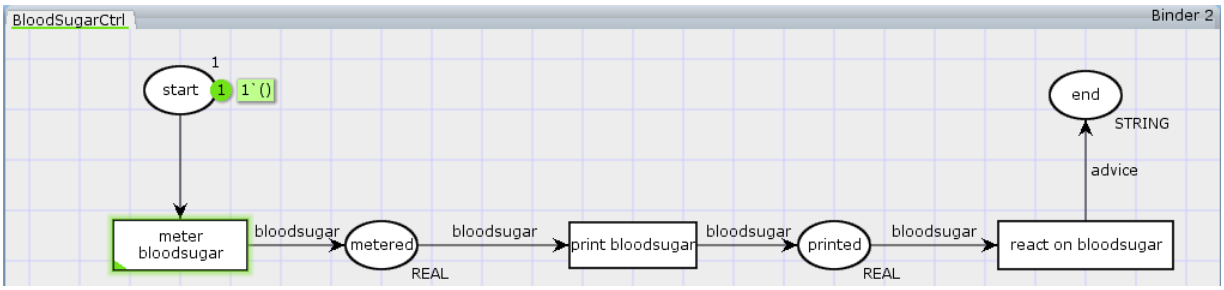


Figure 5.4 – The running example - metering blood sugar and react on the blood sugar value.

In case the blood sugar is below 2.78 mmol/L the sub process *TakeDextrose* is executed. A blood sugar level below 2.78 mmol/L means that the blood sugar is too low. The user that meters his/her blood sugar has hypoglycemia. That is why, the user should take dextrose. Figure 5.5 shows the sub process, that is attached to the transition *react on bloodsugar* in case of hypoglycemia.

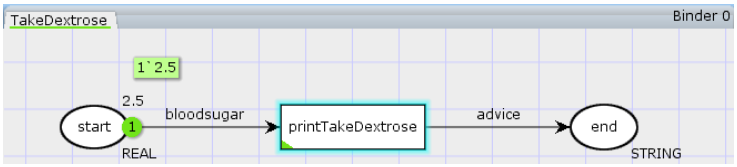


Figure 5.5 – The replacement in case of a too low blood sugar level.

In case the blood sugar is over 6.9 mmol/L the user has a hyperglycemia and sub process *InjectInsulin* is executed. Is the blood sugar value between 6.9 mmol/L and 16.65 mmol/L the situation is not extremely critical and the user has to inject insulin. The sub process that replaces *react on bloodsugar* in case of such a hyperglycemia is shown in Figure 5.6.

Figure 5.6 – The replacement in case of a too high blood sugar level.

In case the blood sugar is higher than or equal to 16.65 mmol/L the user does not only have hyperglycemia but an extremely critically hyperglycemia. The sub process to be executed in such a case is *CallAmbulance*. This sub process triggers the execution of several activities. For

instance, in a simplified case, the process has to call an ambulance and in parallel to compute the insulin to inject (1/5 of the daily dose rate). Additionally, the process will print some advices for the user such as *Drink 1 L of mineral water*. This sub process is shown in Figure 5.6.

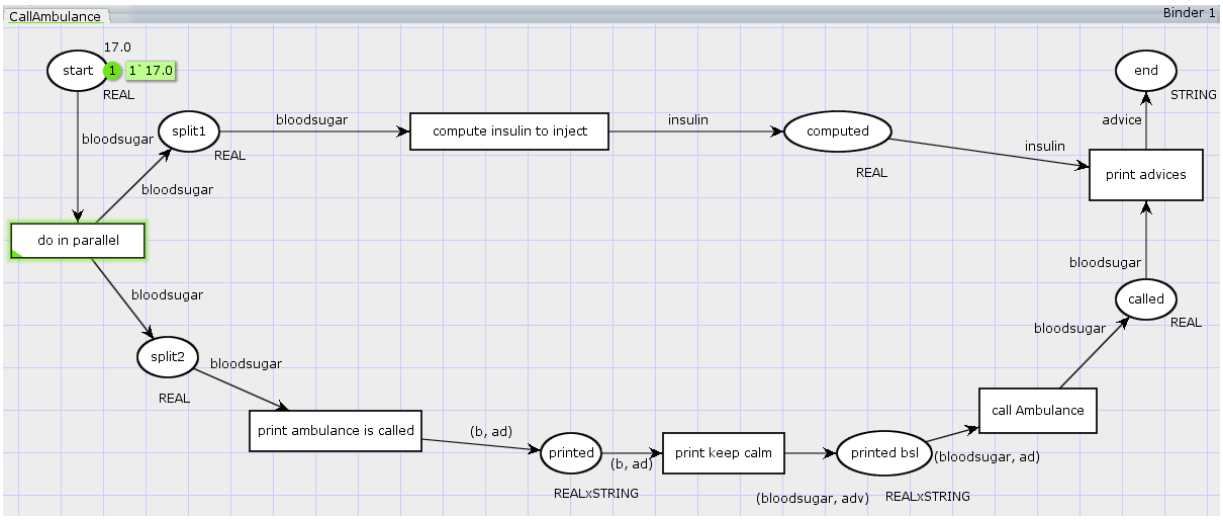


Figure 5.7 – The replacement in case of a much too high blood sugar level.

For a working example we need to define rules that control the reaction on context changes. We need rules for a too low blood sugar level, a too high blood sugar level and a much too high blood sugar level. In the following we only show the rule ”too_high”. The other rules are created analogously.

```
rule "too_high"
no-loop
    when
        m : Message(myMessage : message, Float.parseFloat(myMessage) > 6.9
        && Float.parseFloat(myMessage) < 16.65)
    then
        m.setStatus(Message.TOO_HIGH);
        update(m);
end
```

The rule reacts on context values between 6.9 mmol/L and 16.65 mmol/L. In such a case the status of a message is set to *TOO_HIGH* and the update command informs the fact base about this change.

Furthermore, we need to define rules that control the adaptation of the workflow. In our case we need to define the adaptations in case the status of our context changed to *TOO_LOW*, *TOO_HIGH* or *MUCH_TOO_HIGH*. For the adaptations we defined generic methods that executes the adaptation described in Section 5.1.1. We can choose application dependently be-

tween them. In the following, we describe the rule for a too high blood sugar level. The other rules have to be defined analogously.

```
rule "too_high_adapt"
no-loop
    when
        m1: Message(status == Message.TOO_HIGH, myMessage : message)
    then
        CreateBloodsugarChange cd = new CreateBloodsugarChange();
        cd.createSubNet(
          "C://org.cpntools.accesscpn.model.tests//BloodSugarCtrl.cpn",
          "react on bloodsugar",
          "C://org.cpntools.accesscpn.model.tests//InjectInsulin.cpn");
        PetriNet petriNet = cd.getPetriNet();
        DOMGenerator.export(petriNet, "too_high_change.cpn");
end
```

Our example rule ”too_high_adapt” reacts on a status change to *TOO_HIGH*. In this case a subnet is created that attaches a pre-defined process *InjectInsulin* to the transition *react on bloodsugar* in the process BloodSugarCtrl.

Evaluation pattern

The evaluation should be easy and fast to comprehend. Therefore, we developed and used the following evaluation pattern, that guides through the qualitative evaluation of Flexwoman’s adaptations to context changes:

Number: Every rule gets a unique identifier. (The identifier consists of the letter E[3] and a consecutive number n.)

Title/name: The title describes the main content of the pattern. It describes for what the pattern is used.

Given: This part describes which elements are needed in the evaluation.

Goal: This part describes the goal of the evaluation.

Method: In this part it is described what happens during adaptation.

Evaluation result: In this part it is described whether and how the adaptation worked.

Figure: If possible, there is a figure shown that visualizes the result of the adaptation.

[3] E stands for evaluation.

5.1.2 Evaluation results

In the following text we show nine quantitative evaluations belonging to four evaluation classes (substitution transition and sub processes, fragments (including simple tasks), combinations of adaptations and rules). The evaluations are described according to the pattern introduced before. In the evaluation we assume that interfaces of adapted processes and the processes used for the adaptation of this process (sub processes and fragments) fit and the applied operation does not change the past of workflow execution. The examples show adapted process and processes used for adaptation which are predefined at build time. A manual change of processes used for adaptation or a new definition of them at run time of the adapted process is possible without any qualification.

Substitution transition and sub processes

In this section we evaluate processes refined by sub processes.

The running example has no sub processes attached. We create a sub process for the evaluation. This sub process is replaced during the evaluation. First we simulate a blood sugar value below 2.78 mmol/L. In such a case the rules defined in Section 5.1.1 and the method *createSubNet* are applied. The method has the parameters: the initiating net, the transition that is changed to a substitution transition and the newly attached sub process.

```
cd.createSubNet(
  "C://org.cpntools.accesscpn.model.tests//BloodSugarCtrl.cpn",
  "react on bloodsugar",
  "C://org.cpntools.accesscpn.model.tests//TakeDextrose.cpn");
```

The transition *react on bloodsugar* gets a substitution transition and the sub process *TakeDextrose* is attached to it (see Figure 5.8).

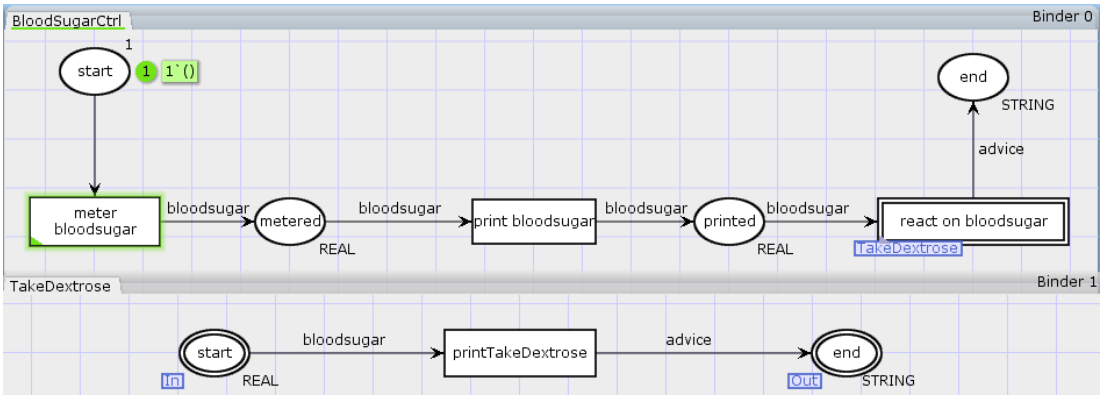


Figure 5.8 – Create a sub process for the task *react on bloodsugar*.

E1: Replace a sub process of a net by another sub process

Given: In this evaluation we need an initiating process, including a substitution transition that is attached by a sub process (*BloodSugarCtrl*), the sub process that replaces the original sub process (*TakeDextrose*), the substitution transition, which sub process is exchanged (*react on bloodsugar*), and the rules that react on context changes and trigger the adaptation according to the current context (see Section 5.1.1).

Goal: An already attached sub process is replaced by another sub process when interfaces of substitution transition and replacing sub process match. The adaptation is triggered by a context change.

Method: The method that we apply to execute this adaptation is called *exchangeSubNet*. It has as parameters the initiating process, the substitution transition and the sub process that replaces the original process:

```
exchangeSubNet(
  "C://org.cpntools.accesscpn.model.tests//too_low_change.cpn",
  "react on bloodsugar",
  "C://org.cpntools.accesscpn.model.tests//InjectInsulin.cpn");
```

Evaluation result: When the context changes to a too high blood sugar value, the sub process *TakeDextrose* of the Petri net *too_low_change.cpn* is exchanged by the sub process *InjectInsulin*. The substitution transition *react on bloodsugar* now links to the sub process *InjectInsulin* (see Figure 5.9).

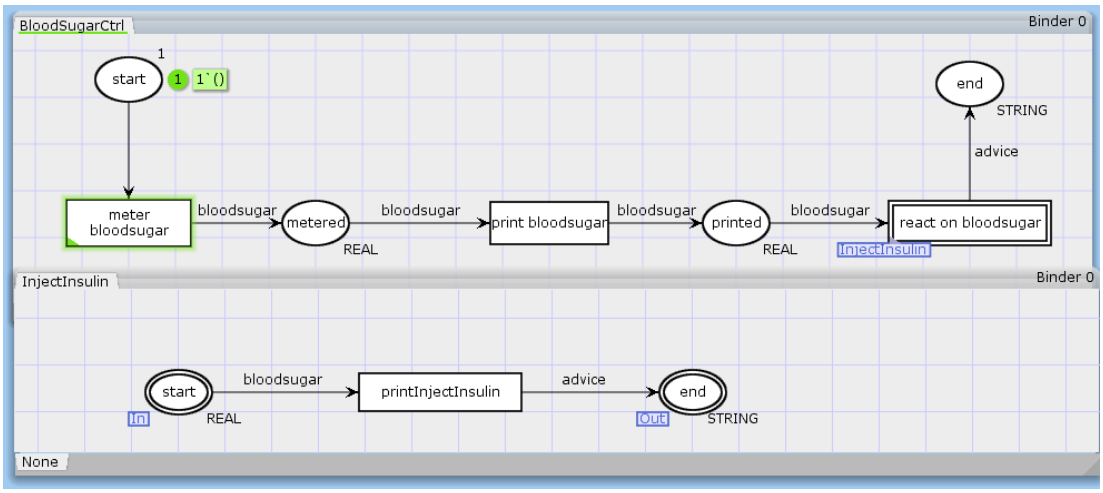


Figure 5.9 – Exchange a sub process (*TakeDextrose*) by another one (*InjectInsulin*).

E2: Delete a sub process or in other words: replace a substitution transition and the belonging sub process by a simple task

Given: In this evaluation we need the initiating process (*BloodSugarCtrl*), including a substitution transition that is attached by a sub process and will be transfered to a regular transition (*react on bloodsugar*), the sub process that should be deleted *InjectInsulin*, and the rules, that react on context changes and trigger the adaptation according to the current context (see Section 5.1.1).

Goal: An already attached sub process is deleted. The adaptation is triggered by a context change.

Method: The method that we apply to execute this adaptation is called *deleteSubNet*. It has the parameters: initiating process, and substitution transition that will be replaced by a ”usual” transition:

```
deleteSubNet(
  "C://org.cpntools.accesscpn.model.tests//too_low_change.cpn",
  "react on bloodsugar");
```

Evaluation result: A context change triggers the deletion of the sub process *InjectInsulin*. The substitution transition *react on bloodsugar* is transfered to a ”usual” transition (see Figure 5.10).

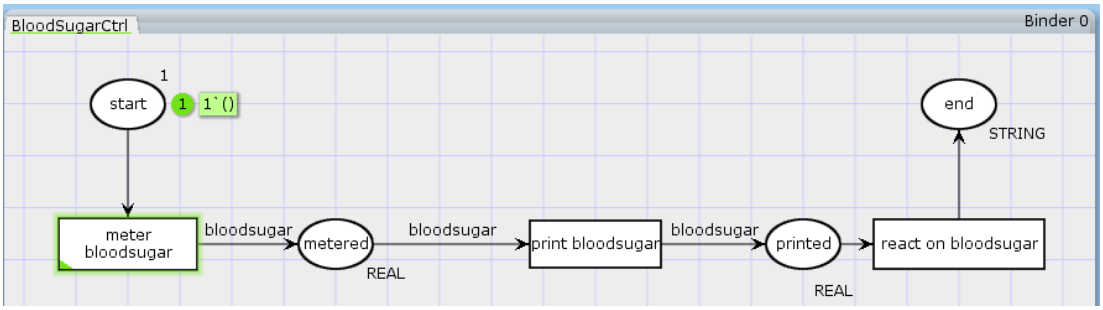


Figure 5.10 – Delete a sub process (*InjectInsulin*) and replace the substitution transition by a ”usual” transition *react on bloodsugar*.

Fragments (including simple tasks)

In this section we evaluate processes refined by inlining sub processes/fragments.

E3: Replace a fragment by another fragment

Given: In this evaluation we need the initiating process, the start and the end of the fragment, that is replaced, the replacing sub process (fragment), and the rules that react on context changes and trigger the adaptation according to the current context.

Goal: Replace an arbitrary fragment by another fragment provided that the interfaces of the fragments match. The replacing of a fragment is triggered by a context change.

Method: The method that we apply to execute this adaptation is called *replaceFragmentByFragment*. It has as parameters the initiating process, start and end transition of the fragment to be replaced, and the name of the file that contains the process with the replacing fragment. In case we aim to replace a fragment with a simple fragment, that is, a transition only, the method *replaceFragmentByTransition* can be used. It has the parameters, initiating process, start and end transition of the fragment to be replaced, name of the replacing transition and code. The code is needed when the inscriptions of the incoming and outgoing arcs of the new transition do not match:

```
replaceFragmentByFragment(
  "C://org.cpntools.accesscpn.model.tests//BloodSugarCtrl.cpn",
  "print bloodsugar", "react on bloodsugar",
  "C://org.cpntools.accesscpn.model.tests//CallAmbulance_BSC.cpn"
);

replaceFragmentByTransition(
  "C://org.cpntools.accesscpn.model.tests//BloodSugarCtrl.cpn",
  "print bloodsugar", "react on bloodsugar",
  "react",
  "input(bloodsugar); output(advice);
   action (let
           val advice = \"your blood sugar is\"^Real.toString(bloodsugar)
           in advice
           end);"
);
```

Evaluation result: The fragment is automatically replaced by another simple or complex fragment (see Figure 5.12 and Figure 5.11).

In case the fragment starting from the transition *print bloodsugar* and ending with the transition *react on bloodsugar* of the initiating process *BloodSugarCtrl* is replaced by a fragment that is contained in the file *CallAmbulance_BSC*.

In the simpler case the fragment starting from the transition *print bloodsugar* and ending with the transition *react on bloodsugar* of the initiating process *BloodSugarCtrl* is replaced by the transi-

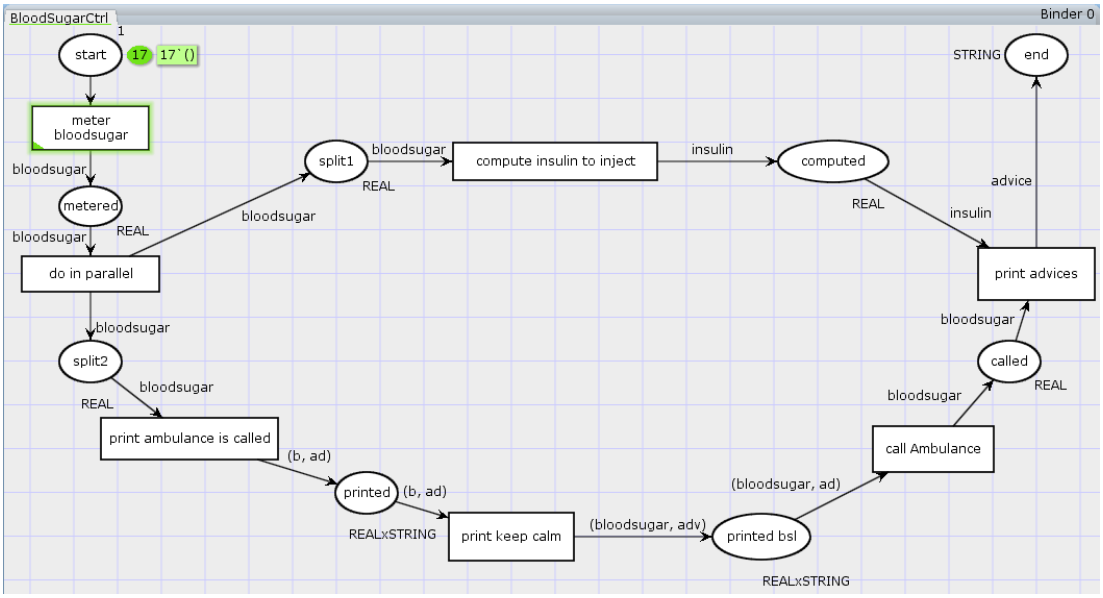


Figure 5.11 – Replacing a fragment (from the transition *print bloodsugar* to *react on bloodsugar* in the process *BloodSugarCtrl*) by the fragment contained in the file *CallAmbulance_BSC*.

tion *react*. This transition is attached by source code to map the content of variable *bloodsugar* of color (type) $Real$ to the variable *advice* of color (type) $String$.

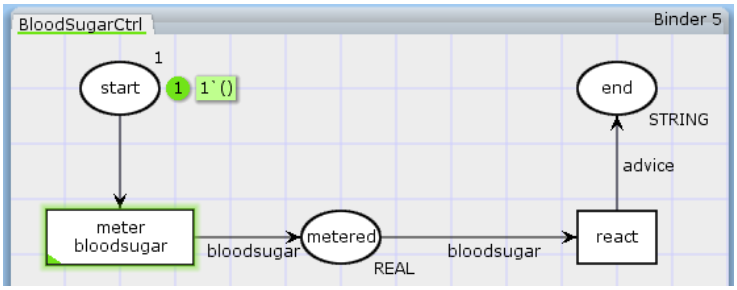


Figure 5.12 – Replacing a fragment (starting from the transition *print bloodsugar* and ending with the transition *react on bloodsugar* of the process *BloodSugarCtrl*) by the transition *react*.

E4: Replace a fragment by a substitution transition and sub process

Given: In this evaluation we need the initiating process, including the start and the end of the fragment that has to be replaced (in case the fragment consists only of one transition it is enough to give the transition, that is refined to a substitution transition), the name of the substitution transition, the sub process that replaces the fragment, and the rules that trigger the adaptation according to the current context.

Goal: An arbitrary fragment of a process is replaced by another sub process. The interfaces of

the fragment and replacing sub process have to match. The adaptation is triggered by a context change.

Method: The method that we apply to execute this adaptation is called *replaceFragmentBySubnet*. It has as parameters the initiating process, the start and the end of the fragment, that is replaced, the name of the substitution transition, and the name of the file that contains the sub process. In case we only want to replace a transition, the method *createSubNet* is used. This method is described in Section 5.1.2:

```
replaceFragmentBySubnet(
  "C://org.cpntools.accesscpn.model.tests//BloodSugarCtrl.cpn",
  "print bloodsugar", "react on bloodsugar",
  "call ambulance",
  "C://org.cpntools.accesscpn.model.tests//CallAmbulance_BSC.cpn"
);
```

Evaluation result: Simple transitions such as *react on bloodsugar* of the process *BloodSugarCrtl* (see Figure 5.8) and more complex fragments such as the fragment starting from the transition *print bloodsugar* ending with the transition *react on bloodsugar* of the process *BloodSugarCrtl* (see Figure 5.13) can be replaced by an arbitrary sub process with fitting interfaces. In the simple case the sub process *TakeDextrose* replaces the simple transition and in the more complex case the sub process *CallAmbulance_BSC* replaces the fragment.

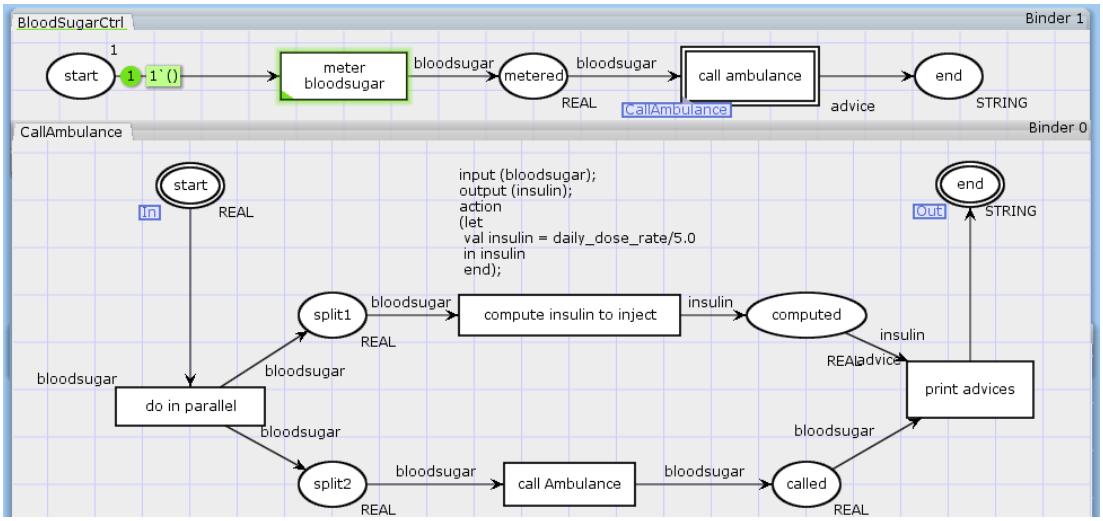


Figure 5.13 – Replacing a fragment with more than one transition (*print bloodsugar, react on bloodsugar*) of the process *BloodSugarCtrl* by a sub process (*call ambulance*).

E5: Delete a fragment

Given: In this evaluation we need the initiating process, the start and the end transition of the fragment to delete, and the rules that trigger the adaptation according to the current context.

Goal: A fragment of a process is deleted. The adaptation is triggered by a context change.

Method: The method that we apply to execute this adaptation is called *deleteFragment*. It has as parameters the initiating process and the start and the end of the fragment to delete, and if needed source code to adapt the interface of the transition that follows the replaced fragment:

```
deleteFragment(
  "C://org.cpntools.accesscpn.model.tests//BloodSugarCrtl.cpn",
  "print bloodsugar", "print bloodsugar");


deleteFragment(
  "C://org.cpntools.accesscpn.model.tests//callAmbulance_BSC.cpn",
  "print ambulance is called", "callAmbulance",
  "input (insulin); output(advice);
   action (let
           val advice=\"high blood sugar level\"
           in advice
           end);");
```

Evaluation result: The fragment is deleted in simple cases (see Figure 5.15) and in complex cases (see Figure 5.14). In the first case the transition *print bloodsugar* of the initiating process *BloodSugarCrtl* is deleted. In the second case the whole branch starting with the transition *print ambulance is called* to the transition *callAmbulance* of the process *callAmbulance_BSC* is deleted. Because in the second case the input to the following transition is no longer correct, the source code attached to this transition is changed.

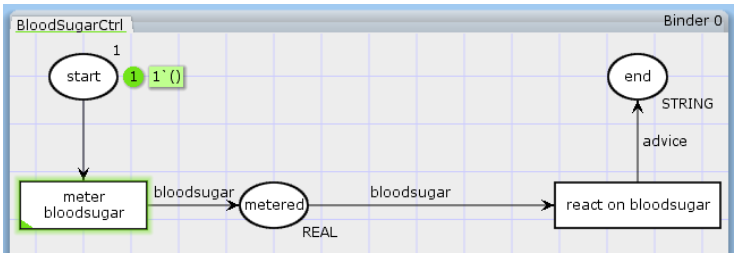


Figure 5.14 – Deleting a simple fragment consisting of only one transition *print bloodsugar* of the process *BloodSugarCrtl*.

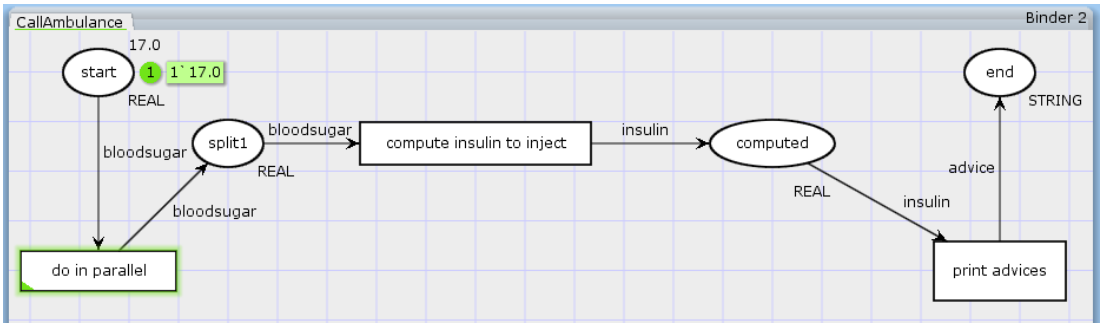


Figure 5.15 – Deleting a more complex fragment (a whole branch) of the process *callAmbulance_BSC*.

Combinations of adaptations

In this section we evaluate whether more than one operations, of the operations introduced before, can be combined. All the operations we described can be thought of as building blocks. We aim for an arbitrary combination of different building blocks with Flexwoman. That is, after applying one of the change operations introduced above, Flexwoman can apply another of these change operations on the result of the first operation. We tested several combinations that all worked. That is why, we assume that all possible combinations work. We picked one clear combination for the description in this evaluation.

E6: Combinations of adaptations

Given: In this evaluation we need at least two operations, all elements that are needed to execute each of these operations, and rules, that trigger the adaptation.

Goal: The goal is to combine different operations just like building blocks and both operations are successfully executed.

Method: We need two adaptation rules, whereby both are triggered at the same time. One rule takes the output of the other rule as input. Therefore, we have to fix the execution order of the rules by the salience attribute of the rule syntax. We have to pay attention that the interfaces of the adaptation operations that are incrementally executed fit to each other. In our concrete case, the first rule replaced a fragment of the CPN *BloodSugarCrtl* starting from transition *print bloodsugar* to transition *react on bloodsugar* by the transition *react*. In the second rule, the transition *react* was replaced by the subnet *InjectInsulin*. The following text shows the applied rules. By way of derogation from the evaluations before we show the complete rules and not only the adaptation operations to show how the attributes can be applied.

```
rule "change1"
   salience 4
   no-loop
```

```
   when
      m1: Message(status == Message.Rule1, myMessage : message)
   then
      CreateBloodsugarChange cd = new CreateBloodsugarChange();
      cd.replaceFragmentByTransition(
       "C://org.cpntools.accesscpn.model.tests//BloodSugarCtrl.cpn",
       "print bloodsugar", "react on bloodsugar", "react",
       "input(bloodsugar); output(advice); action (
        let val advice =\"your blood sugar is\"^Real.toString(bloodsugar)
        in advice end);");
      PetriNet petriNet = cd.getPetriNet();
      DOMGenerator.export(petriNet, "change1.cpn");
end

rule "change2"
   salience -50
   no-loop
   when
      m1: Message(status == Message.Rule1, myMessage : message)
   then
      CreateBloodsugarChange cd = new CreateBloodsugarChange();
      cd.createSubNet("C://org.cpntools.accesscpn.model.tests//change1.cpn",
      "react",
      "C://org.cpntools.accesscpn.model.tests//InjectInsulin.cpn");
   PetriNet petriNet = cd.getPetriNet();
   DOMGenerator.export(petriNet, "change2.cpn");
end
```

Evaluation result: The evaluation was running without problems. In Figure 5.16 the two-stage replacement process is visualized. This result mirrors our idea of using building blocks of operations that can be combined when interfaces fit.

Rules

In this section we evaluate whether rules can be added/changed/deleted in Flexwoman.

E7: Add a rule

Given: In this evaluation we need the initiating process, when needed sub processes or fragments for the adaptation of the initiating process, and usually rules.

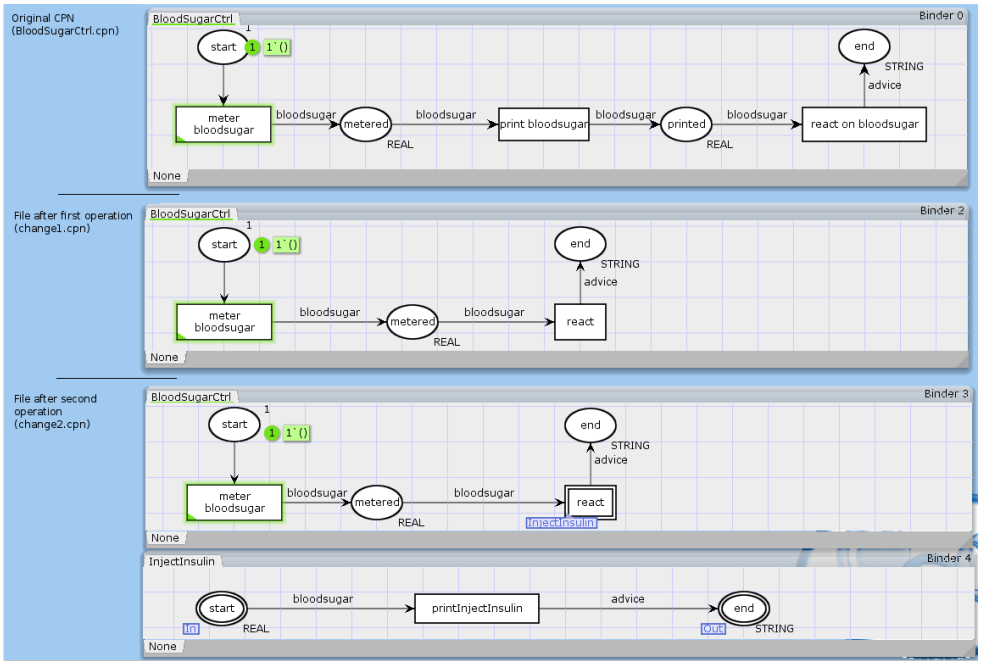


Figure 5.16 – The execution of two operations after the other. Firstly, a fragment of the CPN *BloodSugarCrtl* starting from transition *print bloodsugar* to transition *react on bloodsugar* is replaced by the transition *react*. Secondly, the transition *react* was replaced by the subnet *InjectInsulin*.

Goal: The goal is to add a rule, which is successfully applied when a context change triggers it.

Method: The addition of this rule is manually applied by adding this rule in the rule file that is used for building of the rule base.

Evaluation result: The rule is successfully applied. The rule should not have conditions overlapping with other rules, this can lead to endless loops. Are conditions equal but actions differ, both actions (that of the old rule and that of the newly added rule) are executed.

E8: Change a rule

Given: In this evaluation we need the initiating process, when needed sub processes or fragments for the adaptation of the initiating process, and the rule to be changed.

Goal: The goal is to change a rule, which is successfully applied in the changed form when a context change triggers it.

Method: The change of this rule is manually applied by changing this rule in the rule file that is used for building of the rule base.

Evaluation result: The change of this rule is successfully applied. The rule should not have conditions overlapping with other rules, this can lead to endless loops. Are conditions equal but actions differ, both actions (that of the old rule and that of the newly added rule) are executed.

E9: Delete a rule

Given: In this evaluation we need the initiating process, when needed sub processes or fragments for the adaptation of the initiating process, the rule to be deleted.

Goal: The goal is to delete a rule successfully, that is, this rule is no longer applied when its condition would be true.

Method: The deletion of this rule is manually applied by deletion of this rule in the rule file that is used for building of the rule base.

Evaluation result: The rule is no longer applied.

5.2 Quantitative evaluation

In the qualitative evaluation we usually worked with one change rule and with one, maximal two, adaptation rules. Such an evaluation can show that the functionality is correctly implemented but it does not show how the system works when a larger number of rules is stored in the rule base. Therefore, we decided to have additionally a quantitative synthetic evaluation that tests our system with a growing number of rules.

5.2.1 Evaluation setup

For the tests, we run Flexwoman with an increasing number of rules, which were automatically generated. The categorization rules assign a generated state to each generated range of values. The adaptation rules assign randomly chosen combinations of CPNs and adaptation operations to states. That is, we use relatively simple rules: The categorization rules have two conditions and a simple action part. The adaptation rules have only one condition but the action part is a little bit more complex than that of the categorization rules as it triggers the adaptation of workflows (see Section 4.2.2). The complexity of rules influences the time needed for insert, update, and delete of them as well as the time for pattern matching.

We start our evaluation with 10 rules (5 categorization rules and 5 adaptation rules) and increased the number up to 10000 rules (5000 categorization rules and 5000 adaptation rules). The number of rules is always equally composed of categorization rules and adaptation rules.[4]

For every number of rules we execute five runs and compute the average run time. In each of the five runs six randomly generated context changes are executed. These context changes are inserted into a database and they trigger the adaptation of a CPN dependent on the rules

[4] In Section 4.2.2, we described that context filtering consists of categorization and triggering of adaptation, but in principle, both steps can be merged into one. Thus, usually the cardinality of this relationship is one to one and the number of generated categorization and adaptation rules has to be the same.

in the rule base. For each number of rules and each run the rule base and the fact base is build up from scratch. The time needed for the first context change contains also the time for building rule base and fact base. This time is not used for the computation of the average run time. Summing up it contains of the steps: 1) insert into database, 2) inform (the Context Change Analysis Tool) about a change, 3) build the rule base and insert rules a) build the rule base and add categorization rules, b) add adaptation rules, 4) build plus enlarge the fact base, and 5) adapt the workflow. In the following text we call this first run *initialization mode* or *shortly initialization*. This initialization should be hidden from the user. The average run time of a run is computed of the time needed from context change two to six. We call this mode *normal mode*. It consists of the steps: 1) insert into database, 2) inform (the Context Change Analysis Tool) about a change, 3) enlarge the fact base, and 4) adapt the workflow. This is the mode the user has to deal with when using Flexwoman.

For the evaluations we used a Notebook with Intel Core i7-4700MQ and 2.4GHz. It comes with 32GB DDR Ram, 256GB SSD plus 1TB HDD. The operating system is Windows 8.1 64 Bit.

5.2.2 Evaluation results

In this section we present the evaluation results. We start with a general view that is detailed in the second part of the section.

General view

Table 5.1 sums up the results of the quantitative evaluation of Flexwoman. The table shows the average run time for the adaptation of a CPN when a context value changes. The average run time is computed from five context changes. The metering excludes the building of the database and includes the time from the insert of the context value to the finishing of the adaptation of the CPN file. The values without brackets show the average run time in normal mode and in brackets the run time needed in the initialization mode is shown.

We see that the time needed in the initialization mode linearly grows with the number of rules that the rule base contains. The time for running the rule and adapting the workflow stays nearly constant when the number of rules grows. That is, the time needed for rule base creation grows linearly with the number of rules. Figure 5.17 visualizes that results graphically.

Usually the rule base of an application is rarely rebuilt, therefore its growing with the number of rules does not influence the run time characteristics of an application. That is, only the average run times are relevant.

The results of the metering are, that the values of initialization mode increase with the number of rules because the adding of an increasing number of rules costs more time. The time needed for inserting the categorization rules is higher than the time needed for the insert of the adaptation

number of rules	10	100	1000	2000	4000	6000	8000	10000
run1	5 (16)	7 (15)	7 (29)	7 (39)	8 (74)	7 (94)	9 (148)	9 (180)
run2	7 (6)	7 (8)	6 (13)	7 (22)	7 (78)	8 (85)	7 (136)	6 (184)
run3	5 (7)	5 (5)	6 (11)	7 (22)	8 (71)	7 (93)	8 (136)	6 (188)
run4	6 (8)	6 (7)	5 (13)	6 (21)	7 (64)	7 (97)	12 (128)	12 (160)
run5	6 (7)	6 (7)	6 (12)	6 (23)	9 (44)	8 (74)	8 (118)	8 (175)
average time	6 (9)	6 (8)	6 (16)	7 (25)	8 (66)	7 (89)	9 (133)	8 (177)

Table 5.1 – Quantitative evaluation: We show the average time in seconds needed for every run (normal mode) and in brackets the time in seconds of the first run which also includes the building of the rule base (initialization mode). Figure 5.17 visualizes the results.

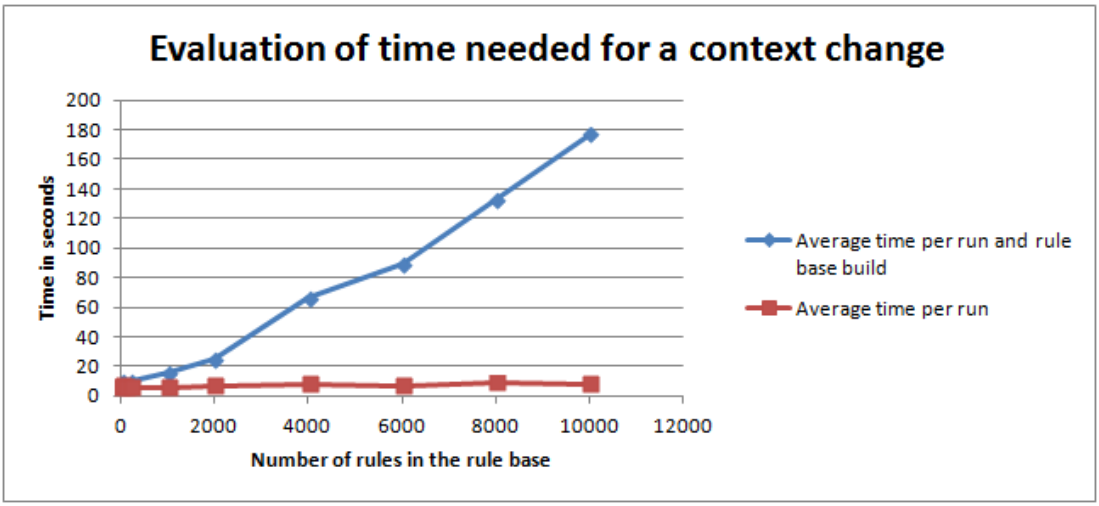


Figure 5.17 – Chart of the run time and the run time plus time for building a rule base.

rules as it contains the building of the rule base. The time needed for the building of the fact base is only contained in the initialization mode. The adaptation of the CPNs is a nearly constant value as the time needed for the adaptation of a workflow stays the same independent of the number of rules in the rule base. The adaptation time is not exactly the same as different adaptation operations have different execution times.

Detailed view

Each context change consists of several steps. The steps to be executed differ dependent on the mode (initialization and normal mode see Section 5.2.1). For some representative cases we will meter the run time of each step to understand the results in detail. We will meter the steps for an insert of 10, 100, 1000, 2000, 4000, 6000, 8000, and 10000 rules. The Tables 5.2 to 5.4 will contain the average values computed out of five measurements. Table 5.2 contains the values of the first initialization, 5.3 contains the average values of the initialization and Table

5.4 contains the average values out of five rounds in normal mode.

number of rules	10	100	1000	2000	4000	6000	8000	10000
insert into database	0	16	0	16	16	15	0	93
inform about a change	31	43	62	52	91	31	55	99
add categorization rules	4603	5443	9544	15219	25764	37438	59276	82441
add adaptation rules	235	953	4032	6531	14344	23643	35362	47157
enlarge fact base	930	886	2080	3172	10978	14161	24181	45638
adapt CPN	7560	13416	9261	10521	13221	7883	9801	12366
run time	13359	20757	24979	35511	64414	83171	128675	187794

Table 5.2 – Quantitative evaluation: We show the time in ms needed for the steps a change operation consists of. This table contains the values for the first round of the first run (initialization mode). These values are significant higher than the values of the average shown in table 5.3.

number of rules	10	100	1000	2000	4000	6000	8000	10000
insert into database	1	4	0	4	1	5	4	4
inform about a change	4	0	4	0	1	1	0	0
add categorization rules	228	741	2514	6770	15481	27595	53016	67742
add adaptation rules	277	687	1852	5610	14057	23484	29219	37513
enlarge fact base	64	122	579	1709	4956	9680	23753	25841
adapt CPN	7317	8308	6254	8112	6826	7043	10624	7439
run time	7891	9861	11203	22204	41322	67804	116615	138538

Table 5.3 – Quantitative evaluation: We show the average time in ms needed for the steps a change operation consists of. This table contains the values for the first round out of a run (initialization mode). It contains times for the creation of rule base and fact base. Figure 5.18(a) visualizes the results.

The tables and the figures show that an insert into the database and the information about a change constitutes a very small percentage of total run time costs. The values are nearly constant. Adding rules to the rule base needs more time the more values have to be inserted.The adaptation of the CPNs needs a nearly constant time over all measurements. The differences in the time needed for adaptation can be explained by the different CPNs adapted with different CPNs with different change operations.

number of rules	10	100	1000	2000	4000	6000	8000	10000
insert into database	3	3	0	3	8	3	3	6
inform about a change	6	0	6	0	4	2	9	3
enlarge fact base	9	22	22	63	29	107	1687	1484
adapt CPN	7934	7263	8376	8622	8387	7344	7566	8832
run time	7953	7288	8404	8688	8428	7456	9266	10326

Table 5.4 – Quantitative evaluation: We show the average time in ms needed for the steps a change operation consists of. This table contains the normal mode. Figure 5.18(b) visualizes the results.

5.3 Further evaluations

Beside the evaluations whether Flexwoman offers the target functionality and whether it performs well when the rule base is growing, we evaluated the system regarding some aspects that influence the usability of the system. We introduced these additional aspects in the following section.

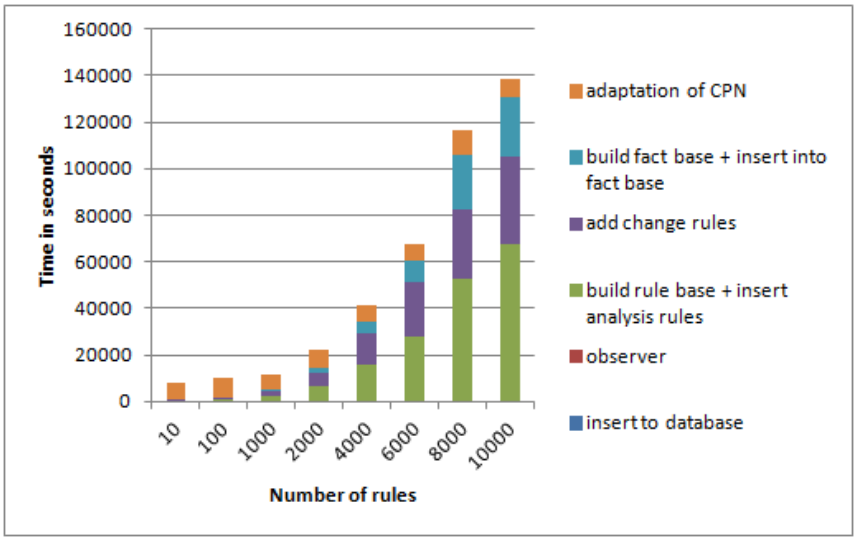
5.3.1 Growth of the knowledge base

What happens if a rule is added on top of the rule base when the system runs? Is the rule base completely rebuilt?

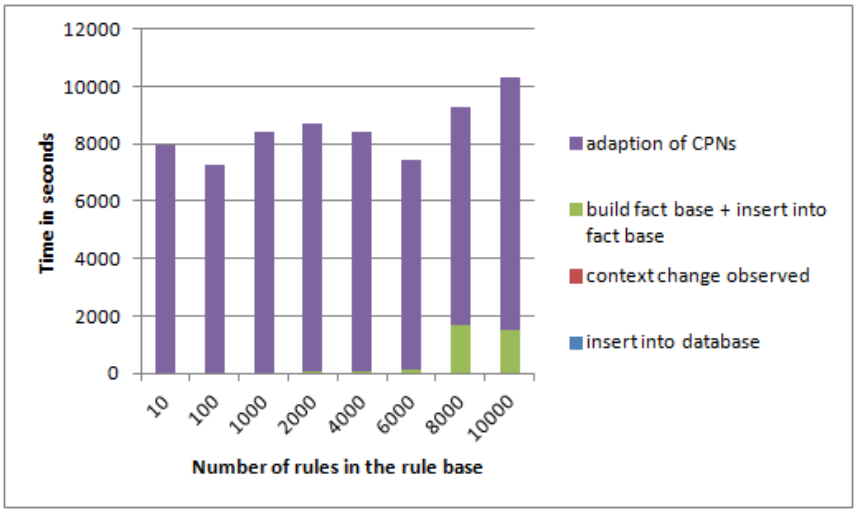
Test setting: Insert the rules from the same rule file twice (the number and the rules in the file are identical).
Test result: The time metered after the first insert is higher than that after the second insert. That is the first insert causes the built of the rule base and thus the metered time is a sum of time for building the rule base and inserting the rules. The time after the second insert of the rules is shorter as it does not contain the time for building the rule base. Thus, the rule base is not rebuilt when new rules are inserted but it is incrementally updated.

How does the system behave when the fact base grows?

Test setting: We increase the number of facts (context changes) that have to be inserted into the rule base and let the number of rules untouched.
Test result: There is usually a time needed to insert a fact into the database that is smaller than one Millisecond. The first context change needs much more time (around 200 Milliseconds) that is the time needed for building the fact base. The fact base is only built in the first run and not rebuild when facts are inserted.



(a) Results of the detailed quantitative evaluation of the initialization: The run time is dominated by the time needed for creating rule and fact base. Both increases with a growing number of rules. The time for adaptations is nearly constant.



(b) Results of the detailed quantitative evaluation of the normal mode: The run time is dominated by the time needed for the adaptation. This time is nearly constant. With an increasing number of rules the time needed for inserting rules into the fact base increases.

Figure 5.18 – The results of the detailed quantitative evaluation.

5.3.2 Rule interdependency

What happens when more than one rule can be fired (no salience and no activation group)?

Test setting: A small rule base is generated with one categorization rule only that is always triggered, and one adaptation rule that is triggered by the only categorization rule. The rule base is manually extended by another adaptation rule. This rule will also be always triggered by the only categorization rule.
Test result: Both rules are executed one after the other. Further tests with salience: The order of firing the rules can be influenced by the salience attribute of the rule. Further tests with the activation-group attribute: Only one rule within an activation-group will fire, i.e., the first one to fire cancels any existing activations of other rules within the same group.

What happens when more than one rule can be fired and the rules are contradictory?

Test setting: We use a net that contains transition a (only once). A small rule base is generated with one categorization rule only that is always triggered and one adaptation rule that is triggered by the only categorization rule. This adaptation rule replaces transition a by sub module b. The rule base is manually extended by another adaptation rule. This rule will also be always triggered by the only categorization rule. This adaptation rule replaces transition a by sub module c.
Test result: The system tries to execute both rules one after the other. After execution the first adaptation rule transition a is replaced by sub module b or c (dependent on the execution order). Afterwards the second rule is triggered. As a is no longer a transition of the net but is replaced with the first execution of a rule by b or c, the second rule that is triggered cannot replace a. Our system Flexwoman instead returns an error message that states that the transition to be replaced does not exist.

What happens when Rule1 replaces a by b and Rule2 replaces b by a? (alternation)

Test setting: We use a net that contains transition a. A small rule base is generated with one categorization rule only that is always triggered and one adaptation rule (Rule1) that is triggered by the only categorization rule that replaces transition a by b. The rule base is manually extended by another adaptation rule (Rule2). This rule will also be always triggered by the only categorization rule and replaces b by a.
Test result: One rule is executed after the other (only once) if and only if the net contains the transition that has to be replaced by the first rule that fires. The rule will not cause an infinitive loop instead after execution the net will look like before the execution of the applied rules. In case the net does not contain the transition to be replaced by the first rule that fires,

Flexwoman returns an error message because the transition to be replaced does not exist. Then the second rule is successfully executed.

5.3.3 Context change interdependency

What happens when a context change happens while the CPN is adapted?

Test setting: We insert several generated values into the database.
Test result: The several inserts trigger adaptations that are executed one after the other.

5.4 Summary

In the following text we summarize the evaluation results and additionally give some advices how to use Flexwoman.

5.4.1 Summary of evaluation results

In this section we evaluated Flexwoman. We started with a qualitative evaluation to test whether the system fulfills the functional requirements. The functionality Flexwoman aims to offer is the run time adaptation of rules and process parts as well as the combination of run time adaptations of process parts. The qualitative evaluation worked well. Flexwoman fulfills the functional requirements as it passed all tests.

The quantitative evaluation tested the system with a larger rule base to get an impression how the system scales and whether it can be applied in realistic scenarios. The results show that a running system needs a nearly constant time for handling a context change (normal mode). For the initialization of the system, that is a non or rarely recurring scenario, the system that has to build rule and fact base. The time needed for building rule and fact base mainly depends on the number of rules that have to be inserted into the rule base. With a growing number of rules the needed time for that initialization grows linearly. The times we metered show the system behavior but the concrete times are not optimized because the topic of low run times is not in the focus of this work.

Afterwards, we tested Flexwoman, especially the used rule base, for its run time behavior in a sense that we checked what happens in potentially critical situations.

5.4.2 Usage of Flexwoman

The evaluations showed that the system offers a proper infrastructure for the automated context-aware adaptation of processes. That is, the system supports the developer with an infrastructure but the developer has to do manual work to customize Flexwoman for the specific application environment in which it is intended to be used. The system developer has to write the rules that react on the context-changes and that adapt the processes in an application specific way. The rules have to be implemented in a semantically consistent way. Additionally, the database we provide is a generic database. If it is necessary to react on different kinds of context, the system developer has to adapt the database and the corresponding trigger. In summary, the developer has to care about the application specific functionality and the consistency of the rules.

Chapter 6

Related work

There are two different research communities working on the challenge of making workflows flexible. On the one hand there is the community that researches workflow management and on the other hand there is the Artificial Intelligence (AI) community, that applies AI techniques to make workflows more flexible. Both aim to bridge the gap between build time workflow definition and run time execution of workflow instances. Because of the different backgrounds of the researchers the developed approaches are very different in nature. The approaches from the workflow community focus on scenarios that do not require a very high degree of flexibility at run time. Many approaches do not support automatic adaptations but only manual adaptation of the workflow instances. Important goals of the workflow community are to keep the overall process correct with respect to the control flow and to apply automated adaptations of the workflow in a non blocking manner. The approaches from the AI community, focus on very high run time flexibility and automation of workflow adaptation. Some approaches accept non optimal solutions and a stop of the workflow execution to apply workflow adaptation. The different characteristics of the approaches of both communities have different strengths and can be applied for different application scenarios. Indeed, many of the new approaches of both communities successfully integrate solutions of both communities with the goal to combine their advantages and use the synergies of their combination.
Additionally, we have a look at approaches for dynamic workflow management used in eHealth. We discuss which of the approaches designed by the workflow community and the AI community are appropriate to handle which workflows in eHealth and we introduce some approaches tailored for special scenarios in eHealth.

6.1 Approaches of the workflow community aiming for flexible WfM

In the following section we will discuss representative approaches developed by the workflow community to achieve a flexible management of workflows. First, we discuss dynamic WfMSs

and subsequently a subclass of dynamic WfMSs: context-aware workflow management systems. The difference between the two classes is that dynamic WfMSs rely on the assumption that exceptions are the triggers for workflow adaptations and in context-aware WfMSs context changes are the triggers for workflow adaptations.

6.1.1 Dynamic WfMS

The first concepts to solve the problem of too strict workflow definitions and too rigid WfMSs originate naturally from the workflow community. In Section 2.4 we described three approaches for adapting workflows, that have their origin in the workflow community: i) Schema evolution, ii) Ad-hoc changes, and iii) Partly defined workflows.

Our work focuses at run time adaptation of workflow instances, where adaptations affect only individual instances and not the definition of a workflow. In contrast, schema evolution handles changes that affects the definition of a workflow. It does not affect an individual workflow instance but all instances of a workflow definition. Schema evolution is only slightly related to Flexwoman. Thus, we do not consider this approach in detail but refer to [CCPP96] for the description of the concepts as well as to the systems WIDE [CCPP96] and WASA [Wes98], that offer implementations for the theoretic concepts of schema evolution.

Ad-hoc changes are short term changes. Typical change operations are the deletion and insertion of activities, or the change of activity sequences. They are applied at run time and allow for a fast and convenient reaction on known and unknown exceptions or events. One of the most important systems supporting this concept is ADEPT [RD98][1]. ADEPT focuses on the development of a system that applies dynamic and structural changes of running workflow instances in a correct and consistent way. Pre- and post-conditions are used to assure the guarantees given at build time. The complexity of adaptation is largely hidden from the users. ADEPT does rely on manual intervention for choosing the activities for workflow adaptations. Flexwoman aims for automatic adaptation of foreseen workflow adaptations.
CCBRTools [WWB04] is the conversational case-based reasoning tool of CBRFlow, a tool that extends workflow execution with conversational case-based reasoning. For a more advanced support of manual intervention, ADEPT and CCBRTools are combined to ProCycle [WWRD07]. The idea of ProCycle is to annotate changes of the instance with the relevant context that causes the change and to store the resulting cases in the case base. That is, changes of the instance are stored together with the reason of the change (context). This allows for a reuse of the changes for other instances of the same pre-defined process affected by the same (relevant) context changes. Additionally, frequently occurring changes are automatically analyzed and serve as starting point for process improvements. Flexwoman also supports change operations at run time. But we try to avoid manual intervention to apply foreseen workflow adaptations.

[1] ADEPT also supports schema evolution of in-progress workflows, where the change of the workflow definition is not in conflict with the current instance state or previous ad-hoc change [RRD03].

Additionally, the adaptive workflow system AgentWork bases on ADEPT [MGR04]. This system aims for automatic adaptation of workflow instances just like Flexwoman. It also uses ECA (event condition action) rules that detect events which cause adaptations[2] of control and data flow. Additionally, it also uses rules to trigger required adaptations. Thereby, the system offers a predictive adaptation of not executed parts of running workflow instances, that is, the process adaptation is done in advance. In cases predictive adaptation is not possible reactive adaptation is applied, that is, the adaptation is executed immediately before the affected workflow part is executed. In contrast to Flexwoman the adaptation is done by cooperating agents. Relevant events are detected and adaptations are triggered by the event monitoring agent. The adaptation is performed by the adaptation agent and has to be manually confirmed. The workflow monitoring agent checks whether the assumptions of the adaptation agent are correct and triggers, if necessary, workflow readaptations. Additionally, an inter-workflow agent determines if there are implications to cooperating workflows. Where required, this agent informs the cooperating workflows. The system differs from Flexwoman conceptually, because the adaptations in AgentWork are based on failures handling while our system is based on handling of context changes. AgentWork uses the concept of ad-hoc changes. In Flexwoman fragments and sub workflows are explicitly formulated. Usually, WfMSs have execution logs and change logs to reconstruct execution and adaptations of workflow instances. But to make them visible, to analyze them[3] and to reuse the solutions, extra tools have to be developed. The approach of Flexwoman automatically offers and visualizes alternative solutions for reuse. Thus, in Flexwoman the reuse of adaptations is easier than in AgentWork. At implementation level AgentWork uses an agent based architecture while Flexwoman uses a component architecture.

A further concept for dynamic workflow management are partly defined workflows. With this approach short time changes and less structured workflows can be handled. There are three patterns that can be used to implement this concept: i) late selection, ii) late modeling and iii) late composition. The first pattern allows at build time only for applying pre-defined adaptations and not unforeseen adaptations. We explicitly want to react to unforeseen changes, thus we cannot apply this pattern. Late modeling offers the possibility to react to unforeseen changes. A representative for a WfMS applying this concept is Pockets of Flexibility [SSO01]. The idea of this concept is that there are parts of the workflow that are not affected by changes. These parts can be fixed at build time. The tasks in the fixed workflow parts are defined as conventional workflow tasks. Parts of the workflow unknown at build time are defined as open points (pockets of flexibility). Pockets of flexibility are associated with i) a set of workflow fragments and ii) a special workflow activity, the so called build activity, that provides rules to specify the pocket. The build activity takes care of a valid composition of fragments based on rules. The assembled fragments are of different complexity and range from single activities to larger sub processes. Additionally, the approach allows for fragment definition at run time and therefore it

[2] They call the events failures.

[3] Event logs can be analyzed and knowledge can be extracted from them with process mining techniques as described in [vdA11].

can react to unexpected events. The composition of the fragments is done successively at runtime while the workflow is executed. After finishing process building, the instance specification is called instance template. This instance template acts like the process model for a particular instance.
Another tool working according to the concept of late modeling is Weak dependencies [KLB+08]. The idea of this approach is to replace strong dependencies between process activities by weak dependencies. A strong dependency is a dependency, where a node (activity) A is predecessor of node (activity) B and no additional nodes can be added between them. Activity A is called source node and B destination node. For strong dependencies the following requirements hold: (i) The source node is precondition of the destination node and the destination node is not enabled until the source node is terminated. (ii) The destination node is enabled immediately after the source node is terminated. Weak dependencies relax the requirements and support only requirement (i). In other words, weak dependencies allow for adding activities or groups of activities between source and destination node. This approach acts on the assumption of manual workflow execution. The authors aimed for a non complex approach, convenient to support non experts in the adaptation of workflows. They kept the approach simple and avoided the support of many different adaptation operations.
Late modeling fixes workflow parts at build time that cannot be changed at run time. On the one hand, this can be seen as an advantage because that parts cannot be destroyed by accident or by lack of knowledge. On the other hand, in the real world unforeseen events happen that can lead to changes that affect fixed workflow parts. Flexwoman avoids to fix parts of workflows at build time.

The most flexible sub category of partly defined workflows is late composition. In this approach process fragments are created at build time. The fragments are at run time dynamically selected and completed by control dependencies to compose process instances [WRR07]. The flexibility comes at the cost of run time performance. These concepts shift the work of workflow composition from build time to run time to avoid as many changes to the workflow instances as possible. Although this is a concept describe by the workflow community, this pattern is predominantly implemented by the AI community. The AI approaches which use planners and which are explained in Section 6.2 are implementations of this pattern.
In Flexwoman, we do not need to support the flexibility this pattern offers and we aim for a fast behavior of the workflow at run time. Thus, we do not follow this concept.

A modern approach aiming for dynamic process management is Control Flow Intervention (CFI) [MS10]. CFI is an approach that allows for a flexible and semantic failure handling for composite services. It supports a dynamic replacement of failed services in processes at run time. The failed services of a process specification are replaced by dynamically selected, semantically equivalent or semantically similar services (optimistic forward recovery). To find semantically equivalent or similar services the semantics of those services have to be available. The availability of the semantics allows for a reasoning with the goal to find equivalent or similar

services. Therefore, the services have to be described in terms of functional (input data, output data, preconditions, effects) and eventually in terms of non-functional properties (performance, costs, reliability, etc.). After failure and recovery, the execution of the process continues with a modified specification. The new specification has the same goal as the original specification but uses different services to achieve the goal. Apart from the completely different techniques used by CFI and its focus on composite services, one difference to Flexwoman is that CFI reacts or intervenes after the occurrence of a failure. Flexwoman does not wait for a failure but reacts proactively and adapts the process according to relevant context changes. Thus, we usually do not have to interrupt process execution for process adaptation.

6.1.2 Context-aware WfMSs

Dynamic WfMSs offer the possibility to flexible react to exceptions: e.g., errors, and adapt workflows according to these exceptions. In contrast, context-aware WfMSs offer the possibility to react to context changes with the adaptation of a workflow. The difference is the point of view. In principle we can see exceptions as a result of a context change. Thus, context-aware WfMSs are dynamic WfMSs which use context to find the right workflow execution path. Context-aware systems need to sense context to be aware of a context changes and to adapt the workflow according to it. The additional requirements are reflected in the architecture of context-aware systems that additionally need components for sensing and analyzing context changes. Another difference of dynamic and context-aware systems is that context-aware systems usually apply context changes automatically and traditional dynamic WfMSs use manual intervention.

The CAWE (Context Aware Workflow Execution) framework (see [AFG+07] and [AFG+12]) is a system for managing of context-aware applications, whose business logic is implemented by context-aware workflows. CAWE aims at context-aware adaptation of workflows and user interfaces. It is a flexible workflow adaptation approach as for the implementation of dynamics in workflows it uses partly defined hierarchical workflows where unknown parts are modeled as abstract activities. The abstract activities are associated with one or more sub workflow that is context-dependently chosen to complete the abstract activities. For sub workflows, interfaces are defined that have to match with those of the abstract activity and that fit to the current context. The conditions for selection of the most convenient sub workflow are declaratively specified in rules. That is, a rule engine is used to evaluate the adaptation policies. The approach does not require changes to the workflow engine but embeds executable code in the definition of abstract activities. The embedded code calls the CAWE framework to find a convenient sub workflow dependent on context. The proof of concept prototype of CAWE was implemented on the top of the WfMS jBPM and uses the rule engine Jess. Just like in Flexwoman, in CAWE convenient sub workflows that replace activities are selected based on context. The specification of the replacement is done in rules, just like in Flexwoman. The difference of both system is, that Flexwoman does not rely on abstract activities for adaptations but every task can be replaced

by a sub workflow/fragment. In CAWE there is executable code embedded in the abstract activities. This limits the flexibility of the system because it limits the possibility to react to context changes in advance. It is expected that the current context is not requested in advance and the adaptations are done when the workflow execution encounters an abstract activity. This can slow down the workflow execution time. Additionally, the embedding of code avoids changes to the workflow engine but creates a dependency to the WfMS jBPM. From an implementation point of view we do not base on jBPM and Jess but on CPNTools and drools.

One of the most well known WfMS dealing with dynamic workflow execution is called YAWL [AtHVDAE07]. This system was developed to support the workflow language YAWL (Yet another Workflow Language) [vdAADtH04]. From a dynamic and context-aware WFMS point of view, YAWL offers the Worklet Service that consists of the two sub services: i) Selection Service and ii) Exception Service [Ada10]. The Selection Service allows for replacing a work item[4] with a dynamically selected worklet (a small, discrete workflow process) at run time. The selection is based on context data. It is possible to design a worklet and invoke it at run time of a process instance. Each task of a workflow specification has an extensible repertoire of worklets. The Exception Service handles expected and unexpected processes at run time. The exception handling is done by a combination of so called exlets (self-contained exception handling processes) and worklets. Exlets are selected at run time when an exception occurs. The selection depends on the workflow instance and the context of the exception. For an efficient selection process (of the Selection and the Exception Service), modified Ripple Down Rules (RDR) are used[5]. RDR are hierarchical organized rules with associated exceptions. The advantage of RDR is that the defined rules can be easily refined later. This allows for starting with general rules that can be e.g. refined by learning. YAWL supports context-aware exception handling and avoids a mixing of business logic and exception handling to ease their verification and modification. The concepts and ideas of making YAWL flexible are similar to ours. The differences of our work and YAWL are that YAWL extends CPNs while Flexwoman uses standard CPNs. YAWL requires from the workflow designer the handling of a complex tool, and learning of how to specify e.g. exlets. In Flexwoman the designer only has to deal with CPNs and easily understandable rule definitions. While the sub processes Flexwoman uses to react to unforeseen events are connected and handled by the CPN tool, a worklet in YAWL and its parent process are unrelated cases. The Worklet Service handles relationships, data mappings and synchronization between cases [Ada10]. YAWL does not come automatically with tools to acquire and analyze context from outside the WfMS. Therefore, interfaces of YAWL have to be extended and the needed tools have to be created and implemented. Thus, YAWL is not out of the box context-aware in the sense as used in this work.

[4] In YAWL, a task instance is referred to as work item [Fou14].

[5] RDR were introduced in [CJ90].

6.2 AI techniques aiming for flexible workflow management

Beside the concepts that have their origin in the workflow community, there are concepts that have their origin in the AI community. The idea to use AI techniques is not new (see [JMS+99]) and their application created big expectations [RMBCO07]. AI techniques applied to workflow management are planning and scheduling task but also learning techniques.

Workflow planning with AI techniques requires a modeling of the domain, that is, following elements: i) tasks, and ii) a model of the problem (initial state and goals) [RMBCO07]. The modeled elements are the input of a planning tool that creates the plan for process execution. In case the process has to be adapted, a new plan can be created by updating the problem model. Certainly, with the application of planners, a WfMS can achieve a degree of automation that is much higher than that reached without applying planning techniques. Often, the high flexibility is reached at the expense of time.
Approaches such as [FF06] and [RMBCO07] have to stop workflow execution when an exception occurs (see Marrella et al. [MMR11]). After stopping the planner is invoked. The planner generates a new process taking the current process state as initial state. After the new plan is generated process execution is resumed. With such a procedure time is lost because the complete process execution is stopped when an exception occurs and the planning is done at run time. Additionally, the domain modeling used as base for applying planning techniques is more difficult to understand for humans than the modeling of processes with a WfMS. Dynamic and context-aware WfMSs, and thus Flexwoman, aim for easing the usage of them and minimizing the time needed for adaptations at run time.

Marella et al. [MMR11] introduces a solution to the unwanted time loss for workflow adaptation. Their approach changes only the process parts that have to be adapted and keep the other parts stable. Additionally, it does not stop every task of the process while computing a new plan. It is a non blocking approach.

6.3 WfMSs in eHealth

Most of the approaches for dynamic workflow adaption are generic and can be applied in diverse application areas. Nevertheless, their different properties imply a usage for different application scenarios. In this work we are dealing with processes in eHealth. EHealth is a broad field. The character of the processes is usually dynamic, but ranges from non complex and static, such as the process of making an appointment with a physician, to complex and highly dynamic, such as the patient treatment processes. In this work we focus on dynamic processes in eHealth. Especially, the highly dynamic processes in eHealth have to be correct and do not allow for long interruptions needed for process adaptations. Thus, it is often not meaningful to apply planner based techniques for the dynamic process adaptation to new context information. Further, it

is often difficult to find the right approach of dynamic workflow adaptation for the respective eHealth scenario. In [RRvdGK10] a framework is developed that supports to take the decision, which approach to use[6]. Their work shows that there is the request to use dynamic workflows in eHealth but it exists a uncertainty of what approach and which tool is to use.

Workflow systems that explore the applicability of dynamic workflow management in eHealth, introduced before are e.g.: ADEPT [RRD03] that researched amongst others how hospital processes in a Women's hospital can be supported by dynamic workflows, AgentWork [MGR04] that aims for supporting workflows for cancer treatment, CAWE [AFG+07], that aims for supporting home assistance (monitoring) of patients that are affected by heart diseases and treated with blood thinners.

Additionally, there is e.g. the Perikles Project that aims for dynamic handling of operating room management [SMBH12]. This project bases on YAWL and extends it with observer tasks that monitor data changes and generator tasks that combine so called bricklets (building blocks consisting of tasks) to build subnets according to so called construction rules. The system supports late modeling and handles anticipated changes, for which no strategies can be defined at design time. In this project additional modeling constructs are added to an existing modeling language to achieve more flexibility. For execution, a mapping from the extended language to YAWL is needed. In Flexwoman we do not touch modeling constructs.
Mans et al. describe in [MRvdA+10] how so called Proclets can be used in gynecological oncology healthcare processes. Proclets are lightweigth processes that interact via channels to transport messages from one Proclet to another Proclet. Proclets support environments characterized by fragmented processes that have to interact. They can be used to dissolve complex and interlaced processes into simple fragments to emphasis aspects related to the interaction of workflows. [vdAMR09] This work focuses on different aspects and different kinds of workflows than we do.

6.4 Summary

In this chapter we discussed work related to Flexwoman. We explained that there are two communities that are working at overcoming the problem of inflexibility of workflows caused by a strong differentiation between build time modeling and run time execution. The workflow community strives for flexible processes that are correct and do not need much time for planning at run time. The degree of run time flexibility they want to reach is at medium level, and they do not necessarily aim for a completely automated adaptation process. In contrast, the AI community aims for a very high degree of run time flexibility. But they accept that the complete planning and replanning work is done at run time what can cause a blocking of the overall process execution at run time, and what can slow down the run time performance.

[6] They rely in their work on a different classification of approaches than we do.

We pointed out that our application area eHealth is diverse and the flexibility of eHealth processes can range from static to highly dynamic. We described that we focus on flexible processes, with special attention being paid to flexible processes in the eHealth area that are characterized by a pressure of time. The time needed at run time for planning and replanning has to be as short as possible. Thus, approaches developed by the workflow community are usually more convenient for that special application area. On the other hand, eHealth is also characterized by routine work that influences the health state of patients. Routine work done by humans increases the risk of errors. Thus, an automation of workflow adaptations is desirable for routine work. By means of methods developed by the AI community incorporated in modern approaches of the workflow community such an automation can be achieved.

Chapter 7

Conclusion and Outlook

Today many application domains, such as emergency management and eHealth, have constantly to deal with processes that are not completely plannable in advance. That is, they have to deal with processes characterized by: (i) unforeseen changes, or (ii) changes from which it is not clear when they arise (flexible processes), or (iii) an in advance unknown or only partly known order of execution (unstructured processes).
Other application domains, such as automobile industry and banking, have to offer customized solutions today. Thus, they have to manage (partly) rare or never repeated processes.
Applications from the above mentioned areas benefit of processes that can automatically be adapted to a changed or a not until run time known environment. That is, they benefit of context-aware processes. This is especially true for processes underlying mobile applications, because mobile applications are affected by frequent changes of the environment.

But when looking at WfMSs, the traditional tools to handle processes, it is getting clear that these systems cannot manage processes that are not completely plannable in advance. The handling of customized processes generates too large and complex process definitions. Additionally, WfMSs cannot change processes according to changes in the environment. Traditional WfMSs are created for and are effective in the handling of highly structured processes, which are rarely changed, not affected by unforeseen changes and not affected by foreseen changes from which the time of occurrence is unpredictable.

On the other hand, WfMSs have advantages, flexible, unstructured and context-aware processes can benefit from. WfMSs allow for the *(i) automation of processes* as they coordinate process execution. Process models, specified by the modeling components of WfMSs, are not only the prerequisite for process execution but also *(ii) process documentation* and a mean to improve *(iii) process understandability*. The controlling component of WfMSs allows for the *(iv) controlling of the correct process execution*. WfMSs explicitly represent managed processes and separate them from the application logic. The separation is an advantage over problem specific proprietary applications [RMBCO07] as it eases the understanding of the process logic as well as it simplifies problem detection and *(v) process optimization*. Consequently, the usage

of WfMSs can significantly increase the quality of processes and therefore the competitiveness of organizations that manage their processes with WfMSs.

Consequently, in this work we developed the tool Flexwoman. Flexwoman is a generic concept for a flexible and context-aware WfMS. It supports (i) the manual adaptation of processes that are not completely in advance plannable and (ii) the automated context-aware adaptation of processes. Flexwoman keeps the advantages of conventional WfMSs and adds the flexibility to react on context changes with workflow adaptations. It is a generic tool that can be used to support all kinds of application domains. We decided to use an application scenario from the eHealth domain to explain our work because it illustrates the functionality of Flexwoman particularly well.

Flexwoman uses RBSs to make WfMSs context-aware and CPNs to make them flexible. The development of Flexwoman covers the formal description of the main concepts of Flexwoman with HCPNs, the design of an architecture, the prototypically implementation of the architecture, and the evaluation of the functionality the prototype offers as well as the evaluation of the scalability.

The main problem of handling flexible and unstructured processes in traditional WfMSs is the gap between build time modelling and run time execution. Parts of the workflow that are not designed at build time cannot be executed at run time. One class of approaches that aims to solve this problem are flexible workflow systems. Our work ties in with the work about flexible workflow systems. Instead of using an existing concept we selected the concept of the so called open-point approach and extended it to reach an independence of changes to predefined change regions and to support context-aware workflow adaptations. Additionally, Flexwoman realizes a refinement of workflows over more than one level, the option to replace a workflow part by a sub workflow that is selected out of a set of several different sub workflows with the same interface as the replaced part of the workflow, and the reuse of sub workflows for the replacement of different workflow parts.

The architecture we designed bases on the well known workflow reference model and extends it by following components: (i) the *context collection tool* for capturing and storing context, (ii) the *context analysis tool* for analyzing context changes of their relevance, (iii) the *workflow adaptation component* for adapting the workflow instances to relevant context changes, and (iv) *controlling of runtime changes* to check the correctness of the applied changes.
The context collection tool does not only sense and store context but it unifies the context data collected from different sources and thus allows for a unified analysis of the context data. For context data analysis as well as workflow adaptation we use a rule-based system. The rules can be changed during run time, they are easy understandable for users that have to apply run time changes and they can be executed automatically and efficiently by rule-based systems. The correctness checks of the adapted workflows are executed by a Hierarchical Colored Petri net tool. We also use this HCPN tool to executed workflows, which are internally in Flexwoman described by HCPNs. We have chosen HCPNs as internal representation for the workflows as they offer a deterministic execution of workflows and can represent workflows on

different levels of detail.

In a qualitative evaluation the prove of concept prototype Flexwoman has been successfully checked for the correct implementation of the concepts of Flexwoman. We could show that our concepts are meaningful and that Flexwoman is a flexible WfMS that supports the management of foreseen and unforeseen events. In a quantitative evaluation we successfully checked the scalability of the system when the rule base increases. We could show that the system performs in complex application scenarios.

Outlook

We evaluated existing approaches of dynamic workflow management in this work. In the evaluation we argued that our application scenario eHealth does not need the flexibility the approach late composition needs. Furthermore, we aimed for avoiding the possible run time overhead of late composition as eHealth is time critical. However, late composition can be seen as AI planning problem. In this area there have been a lot of advances in the last years. Planning (and re-planning) is getting faster and faster. Thus, future work could research the use of late composition for workflow adaptation in Flexwoman. Additionally, late composition in combination with a learning component that learns from earlier workflow adaptations can increase the automation of workflow adaptation. Hence, workflow improvement and maintenance can be simplified.

In this work we rely on the assumption that the interfaces of the replaced and the replacing workflow parts are in accordance. For the user it would be meaningful if Flexwoman would support appropriate interface adaptations of data types automatically or if Flexwoman would offer a graphical support for interface adaptations.

In this work we learned that visualization plays an important role for the modeling of workflows and for the understanding of problems in workflow adaptation. Flexwoman is a tool that has to be adapted to the special scenarios it is applied to. Users have to adapt (i) the database when it is not enough to react on the given context types, (ii) the database trigger that filters the relevant context changes if needed, (iii) the rules that categorize context changes and trigger adaptations. Additionally, users have to create and maintain the workflows needed for the adaptation work. For evaluation work, in case of error messages and for illustration of our thesis we rely on the graphical user interface of CPNTools to visualize the adapted processes. But for practical use it would be meaningful to design an IDE that allows an easy and uncomplicated adaptation of the tool Flexwoman to another application scenario. In case of adaptation errors such an IDE would avoid a loss of time. The IDE could abstract from CPNs and use the slightly easier syntax of workflows. Additionally, a graphical visualization of the workflow execution like offered by CPNTools could be provided.

In this work we focused on functionality and not on performance. When we consider the results of the quantitative evaluations we see that the adaptation of the CPNs dominates the run time of change operations in normal mode. Thus, it could be meaningful, to optimize the adaptation of the CPNs.

Appendix A

Comparison of tools that implement CPNs

There is a large set of tools that were developed to implement Colored Petri Nets. The approach to find such a tool is to use an existing comparison or to make an own comparison. A well done, but outdated comparison of high level Petri net tools is [Stö98]. We borrowed some ideas regarding the structure of the comparison from this paper. We could not find any up-to-date comparison or list of CPN tools. Therefore, at first we had to compile a representative list of CPN tools, more precisely a list of Hierarchical Colored Petri Net tools, as the support of hierarchies is necessary for implementing our concept. The starting point for our search was the Petri Net Tools Database[1]. This database aims to give an overview of existing Petri nets tools. It does not focus on Colored Petri nets but on Petri nets in general. This database is comprehensive collection and well done. The main problem is, that the data base is partly outdated. It contains fresh entries from 2013, but older entries are not maintained. Some of the older entries no longer exist. Thus, the database entries are not reliable and have to be checked. The database has a public available WWW interface. This interface only allows for the search of high level petri nets but not for HCPNs. Thus, we compiled a list of high level petri nets from the Petri Net Tools database. We extended this list with tools found by internet search.
In summary, we used following tools in our comparison:

Access/CPN, ALPHA/Sim, AlPiNA, Artifex, Coopn Builder, COSA BPM, CPN-AMI, CPN Tools (former Design/CPN), ePNK, ExSpect, F-net, GDToolkit, Geist3D, GreatSPN, Helena, HiQPN-Tool, HISIm, INA, Income Suite, JFern, JPNDE, Kontinuum, LoLA, Maria, MISS-RDP, MISTA, The Model-Checking Kit, P3, PACE, PEP, Petri Net Kernel, Petruchio, PnetLab, PNMLFramework, PNSim, PNtalk 2.0 (former PNtalk), Poses++ (former Poses), Predator, PROD, ProM framework, PROTOS, QPME, Renew, SEA, Sepia, Simulaworks, SNAKES, Snoopy, SPNP, StpnPlay, SYROCO, TimeNET, YAWL

[1] http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools/db.html

First iteration: From the tools we found we excluded in a first iteration step:

- tools that are no more available or do not run on current systems,
- tools that do not support both, modeling and simulation of CPNs,
- tools where CPNs are only a base and their focus are e.g. workflows,
- tools that are no longer maintained, and
- commercial tools (also that are free for non commercial use).

After the first round of the comparison our list only contains tools, that offer means to model and simulate HCPNs as we aim to build on the top of a HCPN tool to implement our ideas. We limit the comparison to tools, that are open source, because this allows us to understand the source code and to apply changes and/or extensions to the source code that are needed to implement our ideas. The systems have to be ready to use, to avoid spending too much time with installation work or bug fixing.

The following box shows the tools that pass the first round of our comparison.

Access/CPN, CPN-AMI, CPN Tools, ePNK, JFern, JPNDE, PnetLab, PNtalk 2.0, QPME, Renew, Sepia, SNAKES, Snoopy

Second iteration: With the first round, we could exclude a lot of tools, but there are almost 20 tools left. Thus, in a second step we looked whether the tools meet following requirements, to offer a good background for Flexwoman:

- The programming language should be Java because that makes it easier to build applications on the top of it that run on mobile devices.
- They should support ISO/IEC-15909 (PNML).
- They should be well documented.
- They should have a community working with it.

In this second round we wanted to find out whether it is possible to use the left CPN tools as base for a flexible workflow system. We compared the tools according to the above requirements, because: A Java based system has advantages in mobile settings. A support of PNML (Petri Net Markup Language) – the standard interchange format for Petri net tools – allows e.g. for a cooperation with other tools and a reuse of available petri net models. A good documentation (manuals, Javadoc,..) is meaningful to understand the code. In case of questions, a large and active community is desirable. A large community makes a tool future-proof.
To rank the tools we give points for fulfilling the requirements as follows: 2 points for Java as PL

(because this fact eases implementation work in mobile settings a lot), 1 for PNML implementation, 1 for documentation and 1 of an active community. In table A.1 the points given for each tool are summarized in brackets after the name of the tool. In the columns the concrete points given per tool and property are listed in brackets.

Tools	PL Java	PNML	Documentation	Community
Access/CPN (3.5)	Java (2)	yes (1)	small technical documentation, few code documentation, good examples, benefits from documentation of CPNTools (0.5)	forms for support, bugs, feature requests, benefits from CPNTools (0)
CPN-AMI (3)	Ada, C, Java and more (1)	yes (1)	web site, user manual, videos, examples, but not sufficient for this large tool (0.5)	bug report, FAQ (0.5)
CPNTools (3)	simulator: Standard ML, gui: Beta (0)	yes (1)	excellent user documentation at the web site, video tutorials, examples (1)	forms for support, bugs, feature requests, newsletter, facebook, twitter (1)
ePNK (5)	Java (2)	yes (1)	website, manual, examples (1)	email for comments, suggestions, error reports (1)
JFern (3.5)	Java (2)	no (0)	manual in download, examples, video, good code documentation (1)	mailing list, bug report (no activities on both) (0.5)
JPNDE (3)	Java (2)	yes (1)	no documentation (0)	no community (0)
PnetLab (2)	simulation engine: C++, gui: Java (0.5)	yes (1)	basic web site, user manual (0.5)	no community (0) support
PNtalk 2.0 (2.5)	Smalltalk (0)	yes? (1)	website, documentation in download (1)	email for feedback (0.5)

Tools	PL Java	PNML	Documentation	Community
QPME (4)	Java (2)	XML schema based on PNML (0.5)	web site, user documentation (0.5)	Mailing List, News, Forum, Bug Tracker, Feature Requests (1)
Renew (4.5)	Java (2)	yes (1)	small web site, user documentation, FAQ, good code documentation (1)	email for contact with developers (0.5)
Sepia (3)	Java (2)	PNML parsing (0.5)	good code documentation, tests, no user manual (0.5)	no community support (0)
SNAKES (1.5)	Python (0)	only correct handled for low-level nets (0.5)	reference manual (0.5)	small wiki, few entries to issue tracker (0.5)
Snoopy (2)	C++ (0)	only export (0.5)	web site, manuals and tutorials, FAQ (1)	mailing list, bug report (0.5)

Table A.1 – Second round of comparison of CPN tools

Third iteration: In a third step we evaluated only the most interesting tools ($>=$3.5 points). We compared the following systems in detail:
Access/CPN, ePNK, JFern, QPME, Renew

- The tool has to be stable/robust.
- The tools should be resource preserving because it should be used in mobile settings.
- The tools should be easy to install and intuitive in usage.
- A well known and easy to use inscription language is an advantage.

Tools	re-sources	stable	easy in-stallation	intuitive usage	inscription language	miscella-neous
Access/CPN	210 MB eclipse + 320 MB Ac-cess/CPN	yes	yes	yes	SML	—

Tools	re-sources	stable	easy in-stallation	intuitive usage	inscription language	miscella-neous
ePNK		simulator is not stable	yes	partly (tree view as starting point not intuitive)	proprietary, syntax resembles that of CPNTools	simulator is in an experimental state
JFern	7.9 MB Disk	reset, exam-ples, hierar-chies do not work	yes	no, and very short manual, but a video	Java, Kawa, BeanShell and Clojure	hierarchies imple-mented but bugs avoid a usage out of the box
QPME	41.9 MB Disk	yes	yes	mainly yes		no step-to-step simulation, not so suitable for usual CPNs
Renew	5.9 MB Disk	yes	yes	yes	Java	hierarchies are not im-plemented as expected

Table A.2 – Third round of comparison of CPN tools

The third round of the comparison shows that JFern is a smart little tool that would perfect fit our purpose when it would be stable and well documented. It supports Java as inscription language and it is small. Unfortunately, it does not work out of the box: The biggest problem we found (for our work) is that the hierarchy concept cannot be used because of bugs in the implementations. Furthermore, examples belonging to the tool documentation are not working and the resetting of a simulation is not possible. The documentation is extremely small and not sufficient.
Renew is the smallest tool in our comparison and it supports Java as inscription language. But our evaluation showed that hierarchies are implemented in an unexpected way. After folding of a net to a transition or place, the unfolded net is no longer available. A refinement of a place is a splitting of the place into two (similar for transitions). The old version of the net is no longer available.
QPME is adapted to a usage for Queuing Petri nets and difficult to use for conventional CPNs. It does not support a step-by-step simulation. Therefore, we cannot use it.
ePNK is a tool with a large footprint. It is a Petri net platform and all needed functionality can be

plugged in. Unfortunately, the offered simulator is in an experimental state. Because we rely on a good working simulator we have to exclude this tool.
The last tool we evaluated is Access/CPN. It was developed as Java interface to CPNTools but it can be used without the graphical interface of CPNTools. The most important disadvantage is the footprint of this tool, that bases on eclipse. Another disadvantage is the inscription language SML. The main advantages are stability, the easy installation and usage. Additionally, it benefits from the usage of the elaborated simulator of CPNTools and thus the well done documentation of CPNTools.
We decided to use Access/CPN for our work mainly because of its stability and its stable implementation of hierarchies.

List of Definitions

List of Tables

List of Figures

Bibliography

[AB92] Doris E. Altenkrüger and Winfried Büttner. *Wissensbasierte Systeme: Architektur, Entwicklung, Echtzeit-Anwendungen*. Vieweg, 1992. [in German].

[Ada10] Michael Adams. The Worklet Service. In Arthur H. M. Hofstede, Wil M. P. Aalst, Michael Adams, and Nick Russell, editors, *Modern Business Process Automation*, pages 291–326. Springer Berlin Heidelberg, 2010.

[AEtH03] Michael J. Adams, David Edmond, and Arthur H.M. ter Hofstede. The Application of Activity Theory to Dynamic Workflow Adaptation Issues. In *The 7th Pacific Asia Conference on Information Systems (PACIS 2003)*, pages 1836–1852, Adelaide, Australia, 2003.

[AFG+07] Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan. A framework for the management of context-aware workflow systems. In *Proc. of WEBIST 2007 - Third International Conference on Web Information Systems and Technologies*, pages 80–87, Barcelona, Spain, 2007.

[AFG+12] Liliana Ardissono, Roberto Furnari, Anna Goy, Giovanna Petrone, and Marino Segnan. The context aware workflow execution framework. *Int. J. Auton. Adapt. Commun. Syst.*, 5(1):58–76, January 2012.

[AH98] Mark S. Ackerman and Christine Halverson. Considering an Organization's Memory. In *Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work*, CSCW '98, pages 39–48, New York, NY, USA, 1998. ACM.

[AS10] R. Akerkar and P. Sajja. *Knowledge-Based Systems*. Jones & Bartlett Learning, 2010.

[AtHVDAE07] Michael Adams, Arthur H. M. ter Hofstede, Wil M. P. Van Der Aalst, and David Edmond. Dynamic, extensible and context-aware exception handling for workflows. In *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS*, pages 95–112. Springer, 2007.

[Bat94] Don Batory. The LEAPS Algorithms. Technical report, Department of Computer Science, University of Texas at Austin, Austin, TX, USA, 1994.

[BCF+08] Gert Brettlecker, Cesar Caceres, Alberto Fernandez, Nadine Fröhlich, Ari Kinnunen, Sascha Ossowski, Heiko Schuldt, and Matteo Vasirani. *CASCOM: Intelligent Service Coordination in the Semantic Web*, chapter Technology in Healthcare, pages 125–139. Birkhäuser, 2008.

[BGV91] Wilfried Brauer, Robert Gold, and Walter Vogler. *A survey of behaviour and equivalence preserving refinements of petri nets*. Springer-Verlag, London, UK, UK, 1991.

[CCPP96] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. In *Data and Knowledge Engineering*, pages 438–455. Springer Verlag, 1996.

[CDMF07] Stefano Ceri, Florian Daniel, Maristella Matera, and Federico M. Facca. Model-driven development of context-aware web applications. *ACM Trans. Inter. Tech.*, 7:2, 2007.

[CH94] Søren Christensen and Niels Damgaard Hansen. Coloured Petri Nets Extended with Channels for Synchronous Communication. In Robert Valette, editor, *Application and Theory of Petri Nets 1994, Proc. of 15th Intern. Conference*, volume 815 of *Lecture Notes in Computer Science*, pages 159–178. Springer, 1994.

[CJ90] P. Compton and R. Jansen. Knowledge in context: A strategy for expert system maintenance. In Christopher J. Barter and Michael J. Brooks, editors, *AI '88*, volume 406 of *Lecture Notes in Computer Science*, pages 292–306. Springer Berlin Heidelberg, 1990.

[CK00] Guanling Chen and David Kotz. A Survey of Context-Aware Mobile Computing Research. Technical report, Dept. of Computer Science, Dartmouth College, Hanover, NH, USA, 2000.

[CMP08] Christine Choppy, Micaela Mayero, and Laure Petrucci. Experimenting Formal Proofs of Petri Nets Refinements. *Electronic Notes in Theoretical Computer Science*, 214:231–254, June 2008.

[Cor93] Meta Software Corporation. *Design/CPN Reference Manual for X-Windows*. Meta Software Corporation, 1993.

[DA93] René David and Hassane Alla. Autonomous and timed continuous petri nets. *Advances in Petri Nets 1993*, pages 71–90, 1993.

[DA00] Anind K. Dey and G. D. Abowd. Towards a better understanding of context and context-awareness. In *Computer Human Interaction 2000 Workshop on the What, Who, Where, When, Why and How of Context-Awareness*, 2000.

[DA05] René David and Hassane Alla. *Discrete, Continuous, and Hybrid Petri Nets*. Springer-Verlag, 2005.

[DDDG08] Gero Decker, Remco Dijkman, Marlon Dumas, and Luciano García-Bañuelos. Transforming BPMN Diagrams into YAWL Nets. In *Proceedings of the 6th International Conference on Business Process Management*, BPM '08, pages 386–389, Berlin, Heidelberg, 2008. Springer-Verlag.

[Dic09] Editors Of The American Heritage Dictionaries, editor. *The American Heritage Dictionary of the English Language*, volume 4. Houghton Mifflin Company, 2009.

[DK76] F. DeRemer and H.H. Kron. Programming-in-the-Large Versus Programming-in-the-Small. *IEEE Transactions on Software Engineering*, 2:80–86, 1976.

[EKR95] Clarence Ellis, Karim Keddara, and Grzegorz Rozenberg. Dynamic change within workflow systems. In *COCS '95: Proceedings of conference on Organizational computing systems*, pages 10–21, New York, NY, USA, 1995. ACM.

[EN94] Javier Esparza and Mogens Nielsen. Decidability Issues for Petri Nets - a survey. *Elektronische Informationsverarbeitung und Kybernetik*, 30(3):143–160, 1994.

[Feh93] Rainer Fehling. A Concept of Hierarchical Petri Nets with Building Blocks. In *Papers from the 12th International Conference on Applications and Theory of Petri Nets: Advances in Petri Nets 1993*, pages 148–168, London, UK, UK, 1993. Springer-Verlag.

[FF06] Hugo M. Ferreira and Diogo R. Ferreira. An integrated life cycle for workflow management based on learning and planning. *International Journal of Cooperative Information Systems*, 15(04):485–505, 2006.

[FMM+09] Nadine Fröhlich, Andreas Meier, Thorsten Möller, Marco Savini, Heiko Schuldt, and Joël Vogt. LoCa–Towards a Context-aware Infrastructure for eHealth Applications. In *Proceedings of the 15th International Conference on Distributed Multimedia Systems (DMS09)*, Redwood City, San Francisco Bay, USA, 9 2009.

[For79] Charles L. Forgy. *On the efficient implementation of production systems.* PhD thesis, Carnegie Mellon University, Pittsburgh, PA, USA, 1979. AAI7919143.

[For82] Charles L. Forgy. Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19:17–37, 1982.

[Fou14] The YAWL Foundation. *YAWL - User Manual, version 3.0*, 2014. `http://www.yawlfoundation.org/manuals/YAWLUserManual3.0.pdf`.

[FRMS10] Nadine Fröhlich, Steven Rose, Thorsten Möller, and Heiko Schuldt. A Benchmark for Context Data Management in Mobile Context-Aware Applications. In *Proceedings of the 4th International Workshop on Personalized Access, Profile Management, and Context Awareness in Databases (PersDB 2010)*, Singapur, 9 2010.

[Gal79] Zvi Galil. On improving the worst case running time of the boyer-moore string matching algorithm. *Commun. ACM*, 22(9):505–508, September 1979.

[GBH+05] Matthias Grossmann, Martin Bauer, Nicola Honle, Uwe-Philipp Kappeler, Daniela Nicklas, and Thomas Schwarz. Efficiently Managing Context Information for Large-Scale Scenarios. In *PERCOM '05: Proceedings of the Third IEEE International Conference on Pervasive Computing and Communications*, pages 331–340, Washington, DC, USA, 2005. IEEE Computer Society.

[GL81] Hartmann J. Genrich and Kurt Lautenbach. System Modelling with High-Level Petri Nets. *Theoretical Computer Science*, 13:109–136, 1981.

[GOR11] Gregor Grambow, Roy Oberhauser, and Manfred Reichert. Contextual injection of quality measures into software engineering processes. *International Journal on Advances in Software*, 4(1 and 2):76–99, 2011.

[GR05] Joseph C. Giarratano and Gary D. Riley. *Expert Systems: Principles and Programming*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, fourth edition, 2005.

[GT00] Ataeddin Ghassemi-Tabrizi. *Realzeit-Programmierung*. Springer-Verlag, 2000.

[HJS91] Peter Huber, Kurt Jensen, and Robert M. Shapiro. Hierarchies in coloured Petri nets. In *Advances in Petri Nets 1990*, pages 313–341. Springer, 1991.

[Hol95] David Hollingsworth. Workflow Management Coalition - The Workflow Reference Model. Technical report, Workflow Management Coalition, January 1995.

[Jen81] Kurt Jensen. Coloured Petri Nets and the Invariant-Method. *Theoretical Computer Science*, 14:317–336, 1981.

[Jen89] Kurt Jensen. Coloured petri nets: A high level language for system design and analysis. In Grzegorz Rozenberg, editor, *Applications and Theory of Petri Nets*, volume 483 of *Lecture Notes in Computer Science*, pages 342–416. Springer, 1989.

[Jen92] Kurt Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Volume 1, Basic Concepts*. Springer-Verlag, 1992.

[JK09] Kurt Jensen and Lars Michael Kristensen. *Coloured Petri Nets - Modelling and Validation of Concurrent Systems*. Springer, 2009.

[JKW07] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured Petri Nets and CPN Tools for Modelling and Validation of Concurrent Systems. In *International Journal on Software Tools for Technology Transfer*, pages 213–254, 2007.

[JMS+99] Peter Jarvis, Jonathan Moore, Jussi Stader, Ann Macintosh, Andrew Casson du Mont, and Paul Chung. Exploiting AI Technologies to Realise Adaptive Workflow Systems. In *Proceedings of the Workshop on Agent Base Systems in the Business Context, held during AAAI-99.*, pages 342–416, 1999.

[KLB+08] Marek Kowalkiewicz, Ruopeng Lu, Stefan Bäuerle, Marita Krümpelmann, and Sonia Lippe. Weak dependencies in business process models. In *Business Information Systems*, pages 177–188. Springer, 2008.

[Lak93] C. A. Lakos. The Role of Substitution Places in Hierarchical Coloured Petri Nets. Technical Report TR93-7, Computer Science Department, University of Tasmania, August 1993.

[Lak97] Charles Lakos. On the Abstraction of Coloured Petri Nets. In *Proceedings of Petri Net Conference 97*, pages 42–61. Springer-Verlag, 1997.

[Lak00] Charles Lakos. Composing Abstractions of Coloured Petri Nets. In Mogens Nielsen and Dan Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 323–342. Springer Berlin / Heidelberg, 2000.

[LLC] Dictionary.com LLC. context. http://dictionary.reference.com/browse/context. [Online; accessed 27-August-2012].

[Lug05] G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson education. Addison-Wesley, fifth edition, 2005.

[Mat96] R. Mattmann. *Rapid-Prototyping eingebetteter Systeme*. TIK-Schriftenreihe. vdf, Hochschulverlag AG an der ETH Zürich, 1996.

[MBLG90] Daniel P. Miranker, David A. Brant, Bernie Lofaso, and David Gadbois. On the performance of lazy matching in production systems. In *Proceedings of the eighth National conference on Artificial intelligence - Volume 1*, AAAI'90, pages 685–692. AAAI Press, 1990.

[MGR04] Robert Müller, Ulrike Greiner, and Erhard Rahm. AGENT WORK: a workflow system supporting rule-based workflow adaptation. *Data Knowl. Eng.*, 51(2):223–256, 2004.

[Mir87] Daniel P. Miranker. TREAT: A Better Match Algorithm for AI Production Systems; Long Version. Technical report, University of Texas at Austin, Austin, TX, USA, 1987.

[MMR11] Andrea Marrella, Massimo Mecella, and Alessandro Russo. Featuring Automatic Adaptivity through Workflow Enactment and Planning. In *CollaborateCom 2011*, pages 372–381. IEEE, 2011.

[MRvdA+10] R. S. Mans, Nick C. Russell, Wil M. P. van der Aalst, Piet J. M. Bakker, Arnold J. Moleman, and Monique W. M. Jaspers. Proclets in healthcare. *Journal of Biomedical Informatics*, 43(4):632–649, 2010.

[MS10] T. Möller and H. Schuldt. OSIRIS Next: Flexible Semantic Failure Handling for Composite Web Service Execution. In *Semantic Computing (ICSC), 2010 IEEE Fourth International Conference on*, pages 212–217, Sept 2010.

[Mül11] Sarah Müller. Evaluating Rule Engines and Implementing a Context-sensitive Application for Mobile Platforms. Bachelor's thesis, University of Basel, Basel, Switzerland, January 2011.

[MvdA05] N. A. Mulyar and W. M. P. van der Aalst. *Patterns in Colored Petri Nets*. BETA publicaties. Beta, Research School for Operations Management and Logistics, 2005.

[Nar04] N. C. Narendra. Flexible Support and Management of Adaptive Workflow Processes. *Information Systems Frontiers*, 6(3):247–262, September 2004.

[Neg11] M. Negnevitsky. *Artificial intelligence: A guide to intelligent systems*. Pearson Education, Limited, third edition, 2011.

[NS63] A. Newell and H. A. Simon. GPS, a program that simulates human thought. In E. A. Feigenbaum and J. Feldman, editors, *Computers and Thought*, pages 279–293. McGraw-Hill, New York, 1963.

[OMG11] OMG. Business Process Model and Notation (BPMN) Version 2.0. http://www.omg.org/spec/BPMN/2.0, January 2011. formal/2011-01-03.

[PAK+11] Rinus Plasmeijer, Peter Achten, Pieter Koopman, Bas Lijnse, Thomas van Noort, and John van Groningen. iTasks for a Change: Type-Safe Run-Time Change in Dynamically Evolving Workflows. In *Proceedings of the 20th ACM SIGPLAN workshop on Partial evaluation and program manipulation*, PEPM '11, pages 151–160, New York, NY, USA, 2011. ACM.

[Per89] Mark W. Perlin. The match box algorithm for parallel production system match. Paper 1895, Computer Science Department, Carnegie Mellon University, 1989.

[Pet62] Carl Adam Petri. *Kommunikation mit Automaten*. PhD thesis, Darmstadt University of Technology, Germany, 1962. [in German].

[PW05] Frank Puhlmann and Mathias Weske. Using the π-calculus for formalizing workflow patterns. In *Proceedings of the 3rd international conference on Business Process Management*, BPM'05, pages 153–168, Berlin, Heidelberg, 2005. Springer-Verlag.

[RD98] Manfred Reichert and Peter Dadam. ADEPTflex: Supporting Dynamic Changes of Workflow without Loosing Control. *Journal of Intelligent Information Systems*, 10:93–129, 1998.

[RHM06] Nick Russell, Arthur H. M. Ter Hofstede, and Nataliya Mulyar. Workflow ControlFlow Patterns: A Revised View. Technical Report BPM-06-22, BPM Center, 2006.

[RM03] Erhard Rahm and Rolf Müller. Lecture: Workflow-Management und E-Services, Chapter 5: Flexibles Workflow-Management. dbs.uni-leipzig.de/skripte/WMES/PDF4/kap5-1.pdf, 2003. [Online; accessed 29-August-2012; in German].

[RMBCO07] María Dolores R-Moreno, Daniel Borrajo, Amedeo Cesta, and Angelo Oddi. Integrating planning and scheduling in workflow domains. *Expert Systems with Applications*, 33(2):389 – 406, 2007.

[RODS78] N. J. Nilsson R. O. Duda, P. E. Hart and G. L. Sutherland. Semantic Network Representations in Rule-Based Inference Systems. In *Pattern-Directed Inference Systems*, pages 203–221. Academic Press, New York, 1978.

[RRD03] Manfred Reichert, Stefanie Rinderle, and Peter Dadam. ADEPT Workflow management system: Flexible support for enterprise-wide business processes: Tool presentation. *Lecture notes in computer science*, pages 370–379, 2003.

[RRD04] Stefanie Rinderle, Manfred Reichert, and Peter Dadam. Correctness criteria for dynamic changes in workflow systems—a survey. *Data & Knowledge Engineering*, 50(1):9–34, 2004.

[RRvdGK10] Hajo A. Reijers, Nick Russell, Simone van der Geer, and Gertruud A. M. Krekels. Workflow for Healthcare: A Methodology for Realizing Flexible Medical Treatment Processes. In Stefanie Rinderle-Ma, Shazia Sadiq, and Frank Leymann, editors, *Business Process Management Workshops*, volume 43 of *Lecture Notes in Business Information Processing*, pages 593–604. Springer Berlin Heidelberg, 2010.

[San87] James C. Sanborn. A Modifiable Approach to Expert Systems Development. In *Applications of Artificial Intelligence V*, pages 99–106. International Society for Optics and Photonics, 1987.

[SBG99] Albrecht Schmidt, Michael Beigl, and Hans-W. Gellersen. There is more to context than location. *Computers and Graphics*, 23(6):893–901, 1999.

[SCD09] Ahmet Soylu, Patrick De Causmaecker, and Piet Desmet. Context and Adaptivity in Pervasive Computing Environments: Links with Software Engineering and Ontological Engineering. *JSW*, pages 992–1013, 2009.

[SHB98] Amit Sheth, Yanbo Han, and Christoph Bussler. A Taxonomy of Adaptive Workflow Management. In *CSCW-98 Workshop, Towards Adaptive Workflow Systems*, 1998.

[SMBH12] Sebastian Schick, Holger Meyer, Markus Bandt, and Andreas Heuer. Enabling YAWL to Handle Dynamic Operating Room Management. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 100 of *Lecture Notes in Business Information Processing*, pages 249–260. Springer Berlin Heidelberg, 2012.

[SSO01] Shazia W. Sadiq, Wasim Sadiq, and Maria E. Orlowska. Pockets of Flexibility in Workflow Specification. In *Proceedings of the 20th International Conference on Conceptual Modeling: Conceptual Modeling*, ER '01, pages 513–526, London, UK, UK, 2001. Springer-Verlag.

[Stö98] H. Störrle. An evaluation of high-end tools for petri nets. Bericht 9802, Institut für Informatik, Ludwig-Maximilians-Universitat München, Munich, Germany, June 1998.

[Tea13] The JBoss Drools Team. *Drools Documentation, Version 6.0.1.Final*. JBoss, http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/pdf/drools-docs.pdf, 2013.

[vdA98] Wil van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[vdA01] Wil M. P. van der Aalst. Exterminating the dynamic change bug: A concrete approach to support workflow change. *Information Systems Frontiers*, 3(3):297–317, 2001.

[vdA11] Wil M. P. van der Aalst. *Process Mining: Discovery, Conformance and Enhancement of Business Processes*. Springer Publishing Company, Incorporated, 1st edition, 2011.

[vdAADtH04] Wil M. P. van der Aalst, Lachlan Aldred, Marlon Dumas, and Arthur H. M. ter Hofstede. Design and Implementation of the YAWL System. In Anne Persson and Janis Stirna, editors, *Advanced Information Systems Engineering, 16th International Conference, CAiSE 2004, Riga, Latvia, June 7-11, 2004, Proceedings*, volume 3084 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2004.

[vdAMR09] Wil M. P. van der Aalst, R. S. Mans, and Nick C. Russell. Workflow Support Using Proclets: Divide, Interact, and Conquer. *IEEE Data Eng. Bull.*, 32(3):16–22, 2009.

[vdAtHKB03] Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[vdAtHR11] Wil van der Aalst, Arthur ter Hofstede, and Nick Russell. Workflow patterns home page. http://www.workflowpatterns.com/, 2011. [Online; accessed 19-October-2012].

[vdAvH04] Wil van der Aalst and Kees van Hee. *Workflow Management: Models, Methods, and Systems*. MIT Press, Cambridge, MA, USA, 2004.

[VHJG95] John Vlissides, Richard Helm, Ralph Johnson, and Erich Gamma. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49:120, 1995.

[Wei93] Mark Weiser. Some computer science issues in ubiquitous computing. *Commun. ACM*, 36(7):75–84, July 1993.

[Wes98] Mathias Weske. Flexible modeling and execution of workflow activities. In *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, volume 7, pages 713–722. IEEE, 1998.

[Wes11] Michael Westergaard. Access/CPN 2.0: a high-level interface to coloured petri net models. In *Applications and Theory of Petri Nets*, pages 328–337. Springer, 2011.

[Win01] Terry Winograd. Architectures for Context. *Human-Computer Interaction*, 16(2-4):401–419, 2001.

[WK09] Michael Westergaard and Lars Michael Kristensen. The Access/CPN Framework: A Tool for Interacting with the CPN Tools Simulator. In *Applications and Theory of Petri Nets*, pages 313–322. Springer, 2009.

[WRR07] B. Weber, S. B. Rinderle, and M. U. Reichert. Identifying and Evaluating Change Patterns and Change Support Features in Process-Aware Information Systems. Technical Report TR-CTIT-07-22, Centre for Telematics and Information Technology, University of Twente, Enschede, March 2007.

[WWB04] Barbara Weber, Werner Wild, and Ruth Breu. CBRFlow: Enabling Adaptive Workflow Management Through Conversational Case-Based Reasoning. In *Advances in Case-Based Reasoning: 7th European Conference, ECCBR 2004, Madrid, Spain, August 30-September 2, 2004, Proceedings*, volume 7, page 434. Springer, 2004.

[WWRD07] Barbara Weber, Werner Wild, M. U. Reichert, and Peter Dadam. ProCycle–Integrierte Unterstützung des Prozesslebenszyklus. *KI Journal*, 12(4/20):9–15, 2007. [in German].

[ZB97] W. M. Zuberek and I. Bluemke. Hierarchies Of Place/transition Refinements In Petri Nets. In *Proceedings of the 5th IEEE International Conference on Emerging on Technologies and Factory Automation*, pages 355–360. IEEE, 1997.

Curriculum Vitae

Personal Information

Name:	Nadine Fröhlich
Date of Birth:	7. September 1975
Place of Birth:	Wolfen, Germany
Citizenship:	German

Education

2005- 2014	Doctor of Philosophy in Computer Science: Doctor of Philosophy Thesis: ”Context-aware workflow management in eHealth applications”
2002	Diploma in Computer Science: Dipl. - Inf. Thesis: ”Datenintegration am Beispiel der Produktentwicklung”
1999- 2002	Study of Computer Science at Otto von Guericke University Magdeburg, Germany
1999	Diploma in Architecture: Dipl.-Ing. (FH) Thesis: ”Sanierungsplanung und Umnutzung eines 1609 in Tangermünde erbauten Pfarrhauses in Fachwerkkonstruktion”
1994-1999	Study of Architecture at Anhalt University of Applied Sciences, Germany
1990-1994	Gymnasium Gräfenhainichen, Germany, Abitur

Work Experience

Since 11/05	Research Assistant Databases and Information Systems Group, Department of Mathematics and Computer Science, University of Basel, Switzerland

- Research area: Context-Aware Workflow Management Systems
- Research projects: Cascom (Context-aware Business Application Service Co-ordination in mobile Computing Environments) and LoCa (A Location & Context-aware eHealth Infrastructure)
- Teaching Assistant in ”Datenbanken”

07/03 - 11/05	Saxony-Anhalt Postgraduate scholarship candidate

- Research areas: Pattern oriented software development, prototyping, generative programming
- Teaching Assistant in ”Entwicklung technischer Informationssysteme”

04/02 - 06/03	Research Assistant Rechnerunterstützte Ingenieursysteme Group, Department of Technical and Business Information Systems, Faculty of Computer Science, Otto von Guericke University Magdeburg, Germany

- Research area: Federated databases
- Research project ”Workbench für die Informationsfusion”
- Teaching Assistant in ”Datenbanken 1” and ”Entwicklung technischer Informationssysteme”