October 31, 2014

Thesis

Ad-Hoc Context-Driven Changes in a Business Process Management System

Wietse Smid

Faculty of Electrical Engineering, Mathematics and Computer Science (EEMCS)
Software Engineering

Supervision:
University of Twente:
Dr. Luís Ferreira Pires
Dr. Eduardo Gonçalves da Silva
Capgemini:
Arjan Nieuwenhuis, MSc

A thesis submitted in partial fulfillment for the degree of Master of Science
Final Version

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

UNIVERSITY OF TWENTE.

ABSTRACT

Business Process Management Systems are used to keep track of business processes and the participation of (among others) employees in them. At the same time, smartphones are more prevalent in our daily lives than ever, bringing with them lots of knowledge about their context.

This available contextual information can be used to automatically change business processes to suit the context of their users better. To achieve this, ad-hoc changes to a business process can be defined in terms of change primitives and adaptation patterns. These changes are enacted through decisions made by a complex event processor, based on the contextual information sent by the employee's devices.

We design a software architecture that allows for these ad-hoc changes to be applied to the process instances in a Business Process Management System. We then use this architecture to develop a proof of concept implementation, and wevalidate this proof of concept through the application of typical examples of ad-hoc changes.

We conclude that our proposed architecture is suitable for the design of a BPMS that supports the application of ad-hoc changes based on contextual information. This information may be delivered for various sources and mobile devices of end users are a suitable source of information. However, more research is needed for validation on a full scale.

To improve is to change;
to be perfect is to change often.

WINSTON LEONARD
SPENCER-CHURCHILL

PREFACE

As early as I can remember I always liked to solve puzzles. When I was young it would be jigsaw puzzles or building with LEGO, but as I grew older more complex puzzles took my interests, such as chess, illusions and computer games (although some knowledge of these might have been lost to time and/or beer). This combined with the fact that I was good at the exact sciences in school lead to me choosing, unsurprisingly, to study Computer Science at the University of Twente. During my bachelor I grew an interest in solving larger and more abstract puzzles, getting involved into software architecture. This in turn lead to the, again unsurprising, choice to do a masters in software engineering.

This thesis is the result of eight months of work at the University of Twente and Capgemini. After looking for a research topic for a while it dawned on me that I had no real pressing issues that I wanted to research myself. With the help of the folks at Capgemini I stumbled onto the idea of making business processes less rigid. I had worked with business processes and business process management systems before at my part time jobs, and noticed that improvements could be made. Using contextual information in mobile devices was not my first idea, but when I heard the idea it made sense to me, and I started my research. Eight months later, the result is this thesis.

I would like to thank my supervisors for their support in my research. Luís, who's office door I could always barge in, announced beforehand of course, for helping me at the start, putting me on the right path. For always making me think and answer the questions that I had asked him myself only moments before, and for his fast replies in reviewing sections of my thesis, again with lots of *"Think about what you really want to say here"*-type comments written in the margins, which were always helpful.

I would also like to thank Arjan, for keeping me on track and reminding me that research can also have a value for the industry, not just academics. Discussions with him, specifically in the latter stages when I moved to Utrecht, were always of value and made me look at problems from different angles. I would also like to thank Eduardo, who may have only be involved in the later stages of the game, but provided invaluable feedback from even more different angles.

Besides my daily supervision there are other people that made this thesis possible. My girlfriend Nienke, whom I could always trust to kick my ass whenever I wasn't working when I should have been. My father Nico, who may not be a computer scientist but is a doctor, for his trove of knowledge of academia in general, and academic English specifically. I could always call and ask for his advice. Christoph Bockisch, for being my track mentor during my masters and helping me with the formalities of writing a masters thesis. And last but not least Paul Sijbers, who was my first point of contact at Capgemini in the early days, for putting his faith in me (multiple times) when standardized tests and unexpected planning issues failed me.

As I have said more than once to multiple persons: I am not an academic researcher at heart. But I think that may have made this whole experience even more valuable. I hope you enjoy reading this thesis as much as I have had writing it.

Contents

Chapter 1

Introduction

In this chapter we introduce the concepts of Context in Mobile Devices and Business Process Management, and we show the possibilities for synergy between them. We then define our research goal with accompanying research question and objectives. Finally a structure for the remainder of the report is given.

1.1 Context in Mobile Devices

In today's world more and more people have a smartphone. Smartphone penetration is expected to reach 1.75 billion in 2014, rising to 2.5 billion as soon as 2017 [1]. How we use our smartphones has been the subject of research for some time. It has been shown that the time we spend using our smartphones already exceeds the time we spend online on our desktop computers [2]. We use our smartphones when we are at work, to read and reply to email, to make calls, to take notes and to keep our schedule close and up-to-date. As a result, our smartphones have access to a lot of information, which can be used for lots of different applications.

When a device has knowledge about itself and its surroundings it can use this information to react to a situation, as well as adapt to it if necessary. This type of systems are normally called *Context-Aware System*s (CAS). A CAS can have knowledge of this type of information about their users, the devices they run on, as well as their surroundings [3]. They can make decisions and take actions based on that information. Some examples of CAS' are: an automatic sliding door (a very simple context-aware system) and a car with Lane-Assist technology that warns the driver when he is leaving a lane.

A CAS can make decisions in a number of ways. A simple construct consisting of if-then-else rules can govern how a system reacts to its context, while on the other end of the spectrum some form of

artificial intelligence may be utilized for the decision making. However, most often CAS react and adapt to their environment because of a change in that environment. When multiple changes occur in the environment simultaneously, one might infer that something larger is happening. For example, if the level of light detected is dropping, we would not directly be able to determine its cause, but combining this with the information that there is no more GPS signal could lead us to infer the possible cause, namely that we walked in-doors. A technology that can assist in these situations is called *Complex Event Processing* (CEP), namely by combining multiple sources of information about a changing context to reach decisions [4].

1.2 Business Process Management

Business Processes capture the way a company does its business, often as a combination of tasks and transitions between tasks. Data may also be exchanged as a part of these transitions. The tasks in a business process then require only a transition to be chosen to reach the next task(s) [5]. To accurately keep track of these business processes they are defined in a *Business Process Model*. This exact definition of a business process, together with a record of its executions, allows for analysis and comparison.

Companies are constantly trying to optimize their business processes with respect to various factors. In *Business Process Management* (BPM), the business processes that a company uses can be maintained in a *Business Process Management System* (BPMS) [6]. A BPMS keeps track of the business processes and all the relevant factors, such as progression through the process and documents associated to a business process, throughout time. A BPMS also supports analytical methods that use this information to allow iterative improvement of business processes in the *Business Process Management Life Cycle*.

1.3 Synergies

The fact that we use our smartphones increasingly often in our daily lives opens up increasing possibilities to make use of this contextual information they have access to. Combining contextual information in mobile devices with complex event processing allows one to infer situations and take decisions based on the context captured by these devices.

We can use this contextual information to help support business processes. A business process is subject to change through the business process management life-cycle. The information that causes these changes is based on the analysis of recorded instances of the business process. Some changes

that are temporary, or some that might be needed directly, can thus be realized without having to go through the iterative process. These changes can be done in an ad-hoc manner to allow the BPMS to change processes instances directly based on current contextual information.

1.4 Goal

The goal of this research is: *To identify and demonstrate the benefits of combining Context-Awareness in Mobile Devices with Business Process Management, and design an architecture that can use these benefits to apply ad-hoc changes to Business Processes*.

1.4.1 Research Question and Objectives

To help achieve the goal of this research we have set out a research question and two research objectives:

RQ**:** What is the state of the art in the fields of Context-Awareness in mobile devices and Business Process Management, and how can they be combined?

RO_1**:** Design a software architecture for a BPMS that can use these benefits to apply ad-hoc changes.

RO_2**:** Implement a Proof of Concept and validate the developed architecture.

To answer the research question we have performed a literature study of the relevant fields leading to a conclusion how they can be combined. The first objective asks for the design of a software architecture, the answer to the research question will allow us to develop this architecture. For the second objective, we used the designed architecture to implement a proof of concept, with which we can validate the architecture.

The primary goal of this research is then achieved through the design and evaluation of an architecture for a BPMS that supports the application of ad-hoc changes driven by contextual information from user mobile devices. To show the viability of such an architecture we have built a simple proof-of-concept.

1.5 Report Structure

The remainder of this thesis is structured as follows: Chapter 2 discusses the background on context-awareness, context information in mobile devices, complex event processing and business process management. Chapter 3 presents the principles developed to support ad-hoc changes, outlines an

architecture for a system that is capable of coping with context-driven ad-hoc changes and discusses a few examples where the technology may be applied. Chapter 4 concentrates on the design choices and implementation of the proof of concept, and uses it to validate the proposed architecture. Chapter 6 discusses the results and our conclusions, and identifies opportunities for further work.

Chapter 2

Background Information

This chapter gives the necessary background information for our research. We discuss the concept of context-awareness, contextual information in mobile devices, complex event processing and business process management. We conclude this chapter with a discussion on how these areas of research have been combined, and can be combined in the future.

2.1 Context-Awareness

When using the term *Context-Aware System*s (CAS), the meaning of the word Context should be defined. This section investigates how most authors define Context, which kinds of context exist in Information and Communication Technology (ICT) and how such contextual information can be represented in a computer system. We then discuss our definition of the term Context-Awareness and how context-aware systems make use of sensors to build a representation of their context.

2.1.1 What is Context?

In ICT, context plays an important role in case devices are designed to make autonomous decisions, because such autonomous decisions are often based on the context of the device. Several definitions of context exist. Schilit [3] defines context as: *location, nearby people, hosts and devices, and changes to those things over time*. Thus, context is all about the information that is relevant to the application. However we can extend this definition, since not only external factors come into play when making a decision. The internal state of a device might be relevant to the autonomous decision making process as well [7]. Ryan [6] categorizes context as: *locations, states of sensors and computers, imaginary companions and temporal events*.

Because context is can be complex, real world context is always transformed into a digital representation. This representation might be only accurate to a specified degree, and such partial accuracy might even result in the representation of the context being fundamentally different from the actual context [8]. When talking about context we thus always have to take into account that the device is dealing with a constructed representation or model of the context and not the actual context. Furthermore, this representation is tailored to the application it is used for, and other contextual information irrelevant to the application is normally not considered.

2.1.2 Context Types

The concept of Context may be taken in a very broad sense. While the location of a device is relevant for many applications in terms of contextual information, some other forms of context might not be as easy to recognize. The categories given by Schilit and Ryan as listed in the last section can be extended. Dey et al. [9] divide context into 4 different categories: location, identity, activity, and time. These categories loosely correlate with other categorizations reported in similar research, such as physical, user, computing and time [10]. Below we discuss these four categories defined by Dey et al.

Location Context: The location of device can be important information for decision making. Knowing where a device is allows us to decide what can be done there. Furthermore, the location allows us to infer information about the physical things around us. By knowing which room we are in, we can infer what other items are present, all of which might influence our behavior. A location might be familiar to us or not, and different locations may elicit different responses.

Identity Context: Who we are, and what is around us can also be a part of context. Knowing that there are books on the table is often not enough, we might also want to know which books they are and maybe even who owns them. There are important differences between specific instances of objects. If we see our own cup of coffee on the table, picking it up and taking a sip is an option, but when it is not our cup our reaction might be different. We also want to react differently to different actors, and whether we know someone personally or not will affect our interactions too. Objects thus have an identity and are not by definition necessarily humans. Furthermore, more than one identity can be associated with an object, and we assume that each object has at least one identity associated with it.

Activity Context: In a hypothetical static world we could plan all our actions far in advance, but in reality the environment around us is changing, and we have to take this into consideration when making decisions. All these things happening around us comprise the activity context. Depending on changes in this activity context, we have to react differently. For example, a different action can be taken de-

pending on what some other actor is doing. The difference between speeding up or slowing down while driving a car towards someone is quite important when trying to decide which action has to be taken in order to avoid a collision.

Another example can be seen in smartphones, where multiple programs can be run at the same time. The fact that other programs are running can be important contextual information. For example, when a call comes in and the music player is running, an action can be taken to pause or mute the music player and let the call come through.

Temporal Context: Some things happen at predetermined times. A meeting, for example, might be scheduled from 1:00 pm to 2:30 pm. It is therefore important to know what the current time is if we want to make decisions based on things relative to time. This is most likely the local time, UTC or some other coordinated timeframe. A system that records lectures should only be active while a lecture is being given. This can be derived from the lecture time schedule, and the knowledge of the current timeframe relative to that schedule. Another example of temporal context being used, is when the automatic doors to an office only open during working hours.

Primary and Secondary Context

The context types discussed in this section can be mapped, respectively, to the questions *Where?, Who?, What?* and *When?*. We call these types of context information primary context [9]. This contextual information is retrieved through context sources; which are sub-systems that give access to context information. We can precisely define any context as a combination of these primary contexts. Furthermore, if we have a central system where information is stored, then we can use these primary contexts as anchor points to retrieve more information, and we can infer secondary context from known primary context [9]. This secondary context entails other information related to primary context, for example, the e-mail address and job related to an identity. We can also infer primary contextual information about other entities. For example, by knowing the location of a device we can infer the other identities of other entities (people or objects) known to be in the same location.

We can now define context as a two-tiered system of primary and secondary contextual information. It is also possible that secondary contextual information is inferred from multiple source of primary contextual information. For example, a weather forecast system requires both the location and the time of the desired forecast to be known.

Contextual Representation

Reasoning about context with computers presupposes a digital representation. This representation is only a model or an approximation of reality for two reasons. In the first place, our sensors have limited capabilities, such as speed and accuracy. This introduces a difference between the actual context and its representation. Secondly, it would not be feasible to build a completely accurate representation of the world, since there is simply too much information. We thus restrict ourselves to the information that is relevant to the application. This means that any reasoning about a contextual representation by necessity has to take into account that this information may be both incomplete and inaccurate. Figure 2.1 shows the duality between the real world, and the internal contextual representation.

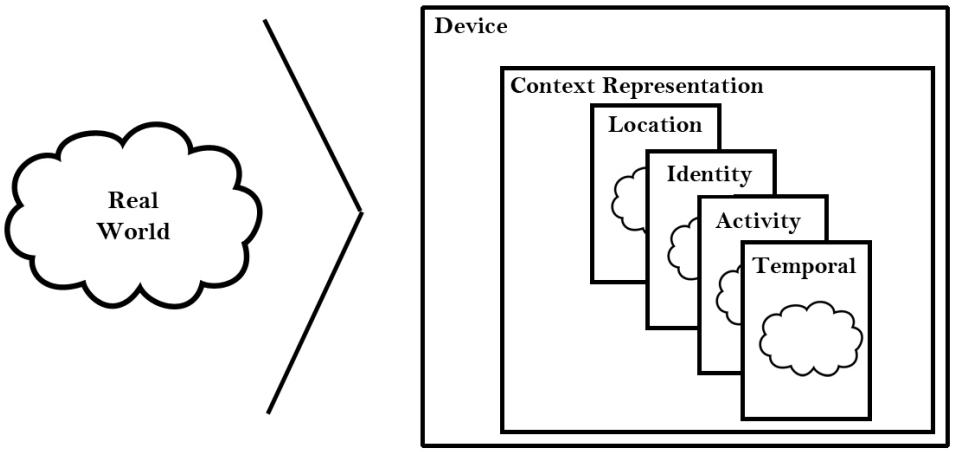

Figure 2.1: A device with the real observable world and its internal representation

2.1.3 Defining Context-Awareness

Schilit et al. [3] defined Context-Aware Systems as systems that adapt according to their context. A lot of different definitions of context-awareness have been given by various authors. Dey et.al. discuss the various definitions of context [9], first defining the difference between using context and adapting to context. When using context, various inputs are used (human, sensory) to gain information about the context, and this information is then used as input for the application. In adaptation, the contextual information is used to perform actions that change the state of the application, such as changing the graphical interface of a program or opening an automatic door.

Dey et al. define context-awareness as follows:

A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the users task [9].

This definition stresses the importance of the duality of using context and adapting to it: when providing information, a system only utilizes the context, and when providing services, a system performs actions based on the context. Furthermore, the definition stresses the relevance of the context to the task at hand. A context-aware system thus reacts and adapts to the context in a way that is useful to the user, and therefore being aware of context that is irrelevant to the task at hand does not comply with our definition of context-awareness. The relevance of a piece of contextual information, however, might change during operation.

Sensor Types

To obtain information about the context a system needs sensors, since sensors allow measurements of some form to be done about the context. Each type of context maps to one or more different types of sensors [11]. In this view, sensors are not restricted to hardware devices like microphones, but they can also be some software or even a service, such as, for example, a database of all postal codes and geographical locations.

For the *location context*, the most popular sensor is GPS, which allows location tracking with an accuracy of up to 5 meters [12]. Other techniques augment and improve GPS and increase its accuracy. Example of this augmentation are WLAN-based location that uses nearby WLAN access points with known location as references, as well as cell tower based location that uses the cellular network towers. Most of these techniques are less effective indoors because of physical barriers that block radio signals. Different techniques can then be used to infer the user's position, such as an accelerometer to count steps and an internal compass to determine orientation, as well as specifically placed base stations to allow triangulation.

To establish the *identity context* and find the identity of different objects, we can also rely on several sensors. In most wireless communications, identity is established by each communicating party. This is true for Bluetooth and WLAN, and to some extent also for RFID. One can also use the camera to determine object identities indirectly, possibly with the use of QR-codes [12]. Identification via audio, such as voice recognition, can also be used to determine if a specific person is nearby to some location.

The *activity context* defines what changes are happening and what is being done at the moment, both in the world around the device, and on the device itself by the user. We can sense this context, for example, via a camera to capture moving objects. In addition to physical changes, we can also detect changes in the software state of a system. For example, we can detect which programs are running or which services are available.

The system clock is the basic mechanism concerning the *temporal context*. Through synchronization via the cellular network we can know the current time with reasonable accuracy. To know the relative times of different context changes, offsets to a fixed time, called timestamps, can be used. Timestamps identify the point in time that a change in context has occurred or will occur. This allows us to reason about these changes afterwards, and also to anticipate on changes in the future.

2.2 Contextual Information in Mobile Devices

Context-aware computing applications have been in use for a long time. Most of the existing applications center around the use of ubiquitous sensing, where a large array of devices and sensors work together to gain information about the context of a situation and whereupon another set of devices and actuators act accordingly [3][13]. These networks of sensors normally have a fixed location, but might spread over a larger geographical area, allowing for a context-aware system to cover, for example, an entire building. An example is an indoor navigation system comprised of stationary displays and user *Radio-Frequency Identification* (RFID) cards [14].

Another idea is to use the contextual information that is available on mobile devices to implement context-aware systems [11]. Although the reach of a distributed sensor network is not attainable with a single mobile device, other information sources are available. A mobile device can provide a wealth of contextual information that allows context-aware systems to adapt to the situation. For example, from the changing location (location context), a music player and a *Global Positioning System* (GPS) tracker being active (both activity context), a context-aware system may infer that the user is jogging or running. This information can then be used to adapt the system, for example by not allowing phone calls to go through, with perhaps the exception of some predefined phone numbers. This section discusses which concrete information, that may be acquired by mobile devices, is useful for decision making.

2.2.1 Sources

To consider the kind of information available to a mobile device we first discuss what sources are available on a mobile device such as a smartphone. We refer to the sensor types from Section 2.1.

Applications

A large source of contextual information can be found in applications. A running application can keep track of virtually anything related to the user. Information may be acquired from the following types of applications:

1. **Calendar application** Our schedule largely dictates our working life. Where we are, and with whom we are engaging in activities can provide us information about our availability.

2. **Time sheet application** When the hours that are spent working are logged in a time sheet application, information about spent time and available time can be inferred. This allows decisions to be made in terms of planning and preventing overtime working.

3. **Employee information application** When a company has an application that keeps track of all its employees' information, then this information can be obtained. Simple things like their place of residence, but also more complex and secondary contextual information such as qualifications or temporary certifications can be obtained.

All these applications expose information. Calendar applications provide activity context, temporal context and maybe even locational context, for example, where a meeting is held. Time sheet and employee applications provide identity and temporal context, telling us more about the employee himself, in terms of availability and capabilities.

Because the software on a mobile device can be as diverse as any software, there are endless possibilities for context information to be accessed by a mobile device, even more so with internet connectivity. To give a more specific example: any information about the person is contextual identity information. Theoretically, an application could do an automated web search for this kind of information. This shows that there are different levels of accessibility when it comes to contextual information, and care must be taken to specify the relevance of different sources of contextual information.

Locational

1. **GPS:** The Global Positioning System can provide a position almost anywhere on earth.

2. **Wi-Fi:** When locations of wireless networks are known a location can be determined based on which wireless networks are in range.

3. **RFID:** Using RFID and fixed receiver beacons we can derive a location for the device.

Each of the above methods provides a location for a device, with differing degrees of accuracy and for different circumstances. What they have in common is the provisioning of updates in intervals. GPS and Wi-Fi do so on a regular basis, RFID's will give updates as soon as they are scanned. Locational data can be used to provide the location of a device, and with such information, nearby objects, persons and activities can be acquired.

Other Sensors

Other sensors exist on a mobile device: camera's, temperature sensors, microphones and more. However, all these have limited effect on a business process, although examples can be thought of. For example, a temperature sensor can alter a business process to check for frozen railway switches more often. But such precisely localized temperature information is seldom required for business processes. Thus, while such sensors can in fact be used, their effective uses are limited.

2.2.2 Collection

The contextual information that all these sources generate, has to be aggregated to be useful for decision making. A separate piece of software is needed to retrieve all the relevant information, and to make it available for use.

There are two major ways to structure the retrieval of this information: polling and events, although a combination is also possible [15]. With polling, sensors are queried periodically for information, time between information points is fixed, but the value might not change. With events, sensors report changes when they occur, timing differs, but there is always a change in value reported.

We argue that the simplest way is also the most effective: every change in value of some contextual information source should correspond to an event-notification. Because a small change might be of no informational value in continuous sensors, for example, the changing of the GPS position by 1 meter, we can define limits within which sensor values can fluctuate without triggering a change notification event. Simpler and discrete sensors, such as a calendar application, already have their own discrete events and those can therefore be collected. These streams of events can then be used in context-aware applications.

2.3 Complex Event Processing

In Event Processing (EP), information is regarded as a stream of events. An event is a representation of a change in context. In Complex Event Processing (CEP), we consider multiple event streams simultaneously and combine the information from these streams to try to understand some specific situation. (C)EP can thus be used to do several things, for example: reporting these streams of events as an aggregated information source, creating understandable visualizations or making autonomous decisions [4]. As with context, what is actually obtained is a stream of representations, and we have to deal with the fact that these representations are only an approximation of the changes in the real world, with all

the pitfalls associated with an approximation.

This section discusses how the information in event streams can be queried via Event Processing Languages (EPLs), and what their properties are. It further discusses how a stream processor uses event streams and queries in an EPL to automatically reach a decision about those event streams.

2.3.1 Event Processing Language

To understand and manipulate event streams, an Event Processing Language (EPL) is often used [16]. Such a language defines how events are described, and how they should be processed. Several of such languages exist, and examples are the SASE language that is specifically designed to analyze real-time streams of RFID readings [16] as well as the EP-SPARQL that combines on-the-fly analysis of streams with background knowledge and reasoning [17].

EPLs define rules according to the Event-Condition-Action (ECA) paradigm: *"on Event when Condition do Action"* [18]. These rules describe on which events and under what conditions an action has to be taken. This type of rule can be extended with extra attributes, such as the time, a post condition or some complex term that can be evaluated or executed at a later time. In Listing 2.1 an ECA rule is given in the syntax of Paschke et. al. that describes that every 10 seconds a check is done to determine whether a service is requested by a customer. If this request is detected, then a server is searched, and if the server is found, the service is loaded. No particular post condition is given, and the notification operation here is enacted if the server cannot be found, in which case a message is sent to the customer [18].

```
eca(
     every10Sec(),                                          % time
     detect(request(Customer, Service),T),                  % event
     find(Server),                                          % condition
     load(Server, Service),                                 % action
     !,                                                     % postcondition
     notify(Customer, "Service request temporarily rejected"). % complex term
}.
```

Listing 2.1: an ECA-type rule showing a service availability check

When considering ECA rules with respect to real-time stream analysis, we can conclude that the time attribute is not necessary, as rules are triggered directly upon incoming events. However, the general Event-Condition-Action constructs are still applicable to real-time stream analysis. In both EP-SPARQL

and SASE a form of query language is used to match events, albeit with some differences [16][17].

EP-SPARQL and SASE both use selectors to retrieve certain information from a matched event, event clauses to describe which events are to be matched, and condition clauses to filter events based on certain conditions. The event clauses enable the matching of sequential events, conjunction, disjunction or absence of events, and concepts such as nonoccurrence of events between two other events. SASE also has an explicit construct to filter for time ranges.

In Listing 2.2 a simple example of a SASE rule definition is given that checks if the patient 'John' has taken an overdose of antibiotics in the last four hours, by matching two MEDICINE-TAKEN events in sequence and applying filters to check if the situation has occurred [16]. SASE was specifically designed to match events on real-time event streams. The action is therefore not defined in the SASE language itself but is left as an external construct [16].

```
EVENT     SEQ(MEDICINE_TAKEN x, MEDICINE_TAKEN y)
WHERE     [name='John'] ∧ [medicine='Antibiotics'] ∧
              (x.amount + y.amount) > 1000
WITHIN    4 hours
```

Listing 2.2: a SASE rule checking for a medicine overdose of a certain patient

2.3.2 Stream Processors

A Stream Processor is a component that processes event streams. It takes event streams as input, together with rule definitions in an EPL, and then enacts the rules defined on the input streams. Depending on the EPL, the decisions can be defined in the rules themselves or can be given in another format to allow separation of event matching and decision making. [16][17].

A simple Stream Processor takes an event stream as input, together with some form of decision mechanism to reason about the event stream. A Complex Stream Processor has multiple event streams as input, as well as an EPL definition on how to analyze them. Figure 2.2 shows how a Complex Stream Processor connects multiple input streams and uses an EPL specification to take actions. Generally these event streams are generated by sensors, but this is not strictly necessary, since stream processors can be used to analyze different kinds of streams, both on-the-fly and on data sets generated from, for example, logging.

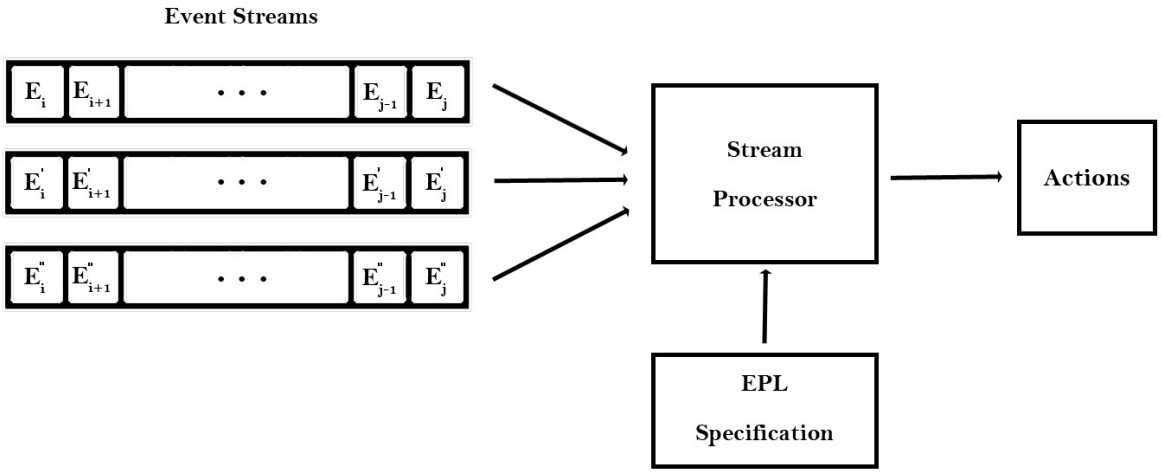

Figure 2.2: Interactions of a Stream Processor

2.4 Business Process Management

Many companies try to improve their business processes by using Information Technology. *Business Process Management* (BPM) is an area that studies a company's internal business processes. These are identified and represented in process descriptions, and subsequently managed and optimized in a continuous monitoring and reviewing cycle [6]. Business Process Management states that business processes are characterized by a number of activities that are performed by computing systems, humans, or their collaboration [19].

In this section we discuss how these business processes are continuously improved through the BPM life-cycle as well as show ways of representing and managing business process models.

2.4.1 Business Process Management Life-cycle

The BPM life-cycle is used to govern the process of designing, enacting, managing and analyzing of operational business processes. The phases of the BPM life-cycle might overlap, but are sequential, and the whole process is iterative [20]. The four BPM life-cycle phases are:

1. Process Design In this phase a model of either a current process or a desired process is produced. To be able to model all aspects of a process, this model can have several different perspectives, such as control-flow, data-flow, organizational and operational perspectives. In this part of the life-cycle, the model is changed according to knowledge about how the process works in practice.

2. System Configuration In this phase, the created process model is used to configure a system that

can keep track of the execution of that model. Models are often constructed in an editor, as a result they can often be used as input to automatically create such a system configuration.

3. **Process Enactment** During this phase, the created system is actually used to manage the instances of the process model. Users can connect and interact with the system to complete tasks from the process model and advance it. Information about the executed process instances is stored in the system for later use.

4. **Diagnosis** The information gained during the enactment phase is analyzed in this phase. Here it may be decided whether the process model needs to be changed, based on the information about the executed process instances stored in the system. This phase therefore leads to a new Process Design phase, and the cycle starts over again.

Through the Business Process Management life-cycle business processes can be continuously improved. The iterative nature of this life-cycle can be seen in Figure 2.3.

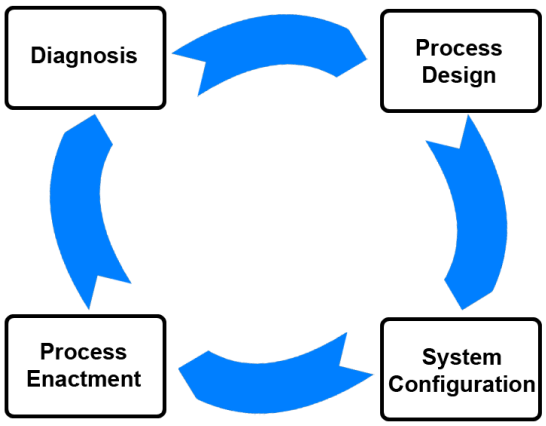

Figure 2.3: The BPM Lifecycle by van der Aalst [20]

2.4.2 Business Process Modeling Notation

To store and handle business process models we need a way to represent them. Business Process Modeling Notation (BPMN) was developed by the Business Process Management Initiative to provide a notation that is readily understandable by all business users [21]. In BPMN, a Business Process Diagram (BPD) is defined based on so called flowcharts that can represent a process as a series of nodes and connecting arrows. A business process diagram is thus a graphical representation of the process with activities (nodes) and flow controls (arrows). BPMN provides several language constructs that facilitate the construction of process models for various purposes, from the creation of first drafts

of business process models, up to technical designs for developers who implement the business processes.

A BPD consists of several elements, connections and constructs [21]. The BPMN elements are discussed below.

Flow Objects There are three types of flow objects in BPMN:

1. **Event** An event is represented by a circle and means that something happens. They usually have a cause (trigger) or an impact (result). Events are split into three types: start events, intermediate events and end events.
2. **Activity** There are two types of activities: task and sub-process. Activities represent the work performed by a company and is shown as a rounded rectangle. A task is a single atomic activity and a sub-process can contain multiple activities, events and gateways connected through flow objects.
3. **Gateway** A gateway is shown as a diamond and can control the sequence flow through decisions, forking, merging and joining of paths.

Connecting Objects The three types of flow objects are connected via three possible connecting objects:

1. **Sequence Flow** A sequence flow shows which activity is to be performed next, it thus defines the order in which activities occur in a process. It is represented as a solid line with a solid arrowhead.
2. **Message Flow** The flow of a message between two participants is shown as a dashed line with an open arrowhead. Participants are separate business entities or business roles, and are represented by two separate pools (Swim lanes).
3. **Association** An association is shown as a dotted line with an open arrowhead. It is used to associate data, text and other artifacts with flow objects.

Swim lanes With swim lanes we can visualize activities together as a region. BPMN has two types of swim lanes:

1. **Pool** A pool represents a participant in a process. Multiple pools can partition the activities according to multiple participants to the process. A pool is represented as a rectangle with a name.

2. **Lane** A pool can be subdivided in lanes. Lanes are used to categorize and organize activities. Lanes have their own names.

Artifacts Besides the functional elements, BPMN has a few artifacts that offer the option of annotating the BPD without changing the process it represents. Artifacts can be added to a BPD to provide extra information about the business process that is modeled. BPMN defines the following Artifacts:

1. **Data Object** Data objects are connected to activities via associations. They represent the data that is required or produced by an activity.
2. **Group** A group is shown as a rectangle with a dashed border. It can be added for documentation or analysis purposes, but it does not affect the Sequence Flow.
3. **Annotation** Annotations allow textual comments to be added to the BPD.

Figure 2.4 shows an example of a BPD in BPMN for the customer request of a new account for a service. It shows the swimlanes indicating different users, as well as the general processing of the request through events (circles), actions (squares) and gateways (diamonds). They are connected through sequence flows (solid arrows), message flows (dashed arrows) and data flows (dotted arrows). The pool has its own start and end events which are accompanied by message flows with the customer.

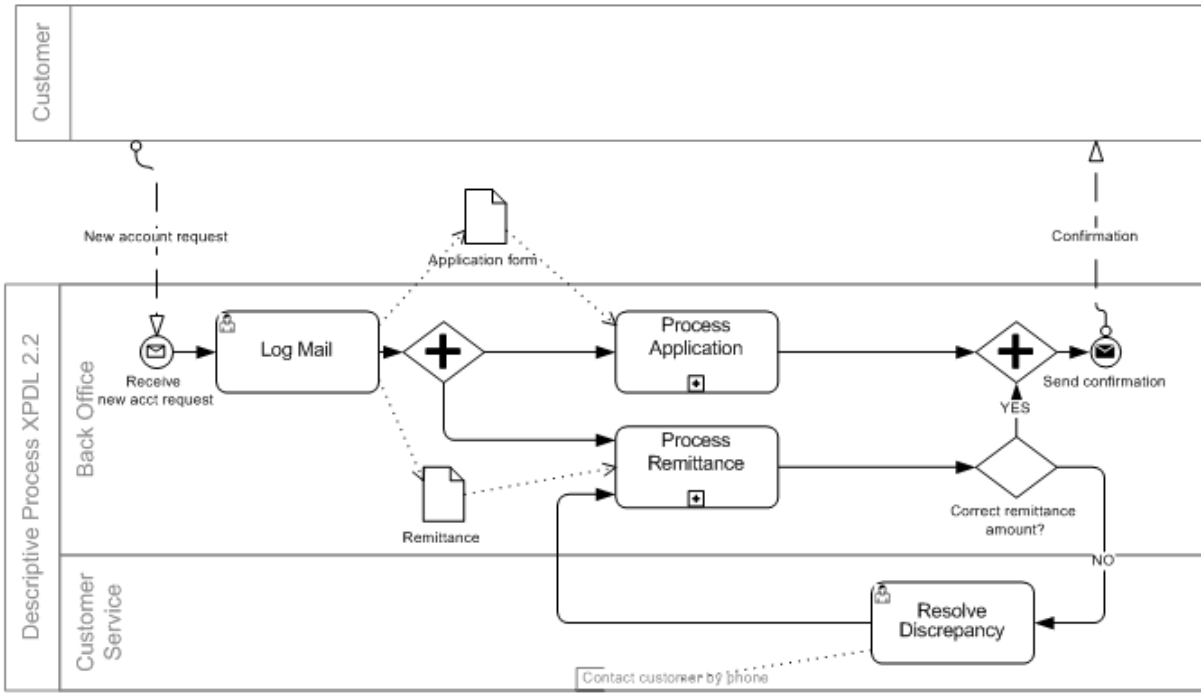

Figure 2.4: Example of a Business Process Diagram in BPMN-notation [22]

2.4.3 Business Process Management System

A Business Process Management System (BPMS) supports the entire BPM life-cycle. A BPMS consists of several software components that support modeling, configuration, enactment and analysis of Business Processes. A BPMS delivers a consistent process management experience, enabling business managers to monitor, analyze and refine the execution of a company's business processes [23].

A Business Process Management System typically consists of the following components:

1. **Process Modeling Tools** The process modeling tools support the creation and modification of BPD's that can then be executed by the process engine.
2. **Process Engine** The process engine allows for the execution of business processes and keeps track of all the running process instances.
3. **Business Analytics** The business analytics component gives insight to managers about completed and running business process instances. It can show problems, trends and opportunities through reports and dashboards.
4. **Content Management** The content management system stores electronic documents, images and files relevant to the business processes.
5. **Collaboration Tools** The collaboration tools allow employees to communicate and collaborate with each other.

2.5 Contextual Information in a BPMS

Some research has been done into using contextual information in a BPMS, and several approaches to the concept have been taken. In this section, we discuss illustrative parts of this research and show the breadth of possibilities for using contextual information in a BPMS.

In broadly defined business processes, multiple routes can be taken from start event to end event. Van der Aalst [24] uses contextual information to navigate through such a business process, feeding the information to a path finding algorithm found in car navigation software as a means to steer the business process execution. Most of the contextual information used is retrieved from event logs of previous executions.

One approach uses contextual information as a source of information in a business process [25]. It envisions, for example, a process where the choice made in a gateway in the process hinges on a

decision. Rather than only using information available in the process, contextual information can be taken into account as well in that decision. Reseman et. al. [25] give an example of an airline check-in process. Multiple routes through the process are defined and a decision to take a specific route is based upon contextual information, such as the availability of an internet connection.

Another approach is to extend the BPMN specification with contextualized elements. These mobility events are designed to be triggered upon the reaching of a position by a participant, possibly supplemented by an additional condition. This allows for business processes to be sensitive to the contextual location data of its participants [26].

In the research done by Santos et. al. [27] contextual information is gathered during the execution of a process. This information is recorded together with an instance. Using Non-Functional Requirements (NFRs) that define quality and constraints, the authors describe the desired goal of a business process. They then develop a framework where using these NFRs, possible variations in a business process are computed based on contextual information. These variations are then applied to the process model, and new instances be started with these variations on the business process model.

The concept discussed in the research by Santos et. al. [27] can be expanded. The mobile devices of end users of a BPMS have access to contextual information. With this contextual information, we can change the business processes not just automatically every iteration of the business process management life-cycle, but all the time, and ad-hoc. These ad-hoc change have to be defined in advance, but which actual changes are applied is decided at runtime based on contextual information.

2.6 Conclusion

In this chapter, we discussed what context is, and how it can be viewed as a layered system that can have digital representation. We discussed how devices are aware of their context through sensors, and how context can be viewed with regards to mobile devices. We then discussed how we can use multiple sources of contextual information to make decisions based on patterns.

We described what a Business Process Management System is, and how business processes are designed and executed. We also discussed several concepts on using contextual information to augment a BPMS, and concluded that there are opportunities for the non-trivial use of contextual information.

We concluded from the above that contextual information should be gathered from the mobile devices of users, and sent to the BPMS in the form of event streams, where through complex event processing the changes described in an event processing language can be applied.

Chapter 3

A BPMS for Ad-hoc changes

Business Process Management Systems have been around since the early 2000's, and they have evolved to a point where the basic architecture is generally agreed upon [28]. In this chapter we discuss a technique to represent and apply ad-hoc changes on process instances. We investigate how the current BPMS architecture can be adapted to accommodate for a BPMS in which contextual information can be used to support ad-hoc changes on a process model or a process instance. Finally we present some examples to show the applicability of a BPMS that supports ad-hoc changes.

3.1 Ad-hoc changes

To design an architecture for a BPMS that supports ad-hoc changes we first have to define what ad-hoc changes exactly are and how we can structure and use them. We discuss what types of ad-hoc changes can be applied to a process model and how these changes can be combined and structured. Because we want to prevent changes that might eventually result in unexpected errors, we look at the problems that might arise when changing a process model. We discuss the concepts of soundness, state compliance and dynamic change correctness, and how they can be used to determine which ad-hoc changes can be applied and which cannot.

3.1.1 Structure

To define ad-hoc changes for business processes we descend to the most elementary level of changing a process. Reichert and Weber [29] call this the level of Change Primitives. By combining these elementary changes, larger and reusable Adaptation Patterns can be constructed. Such a pattern describes which Change Primitives are executed and in what order. We can further combine change primitives and adaptation patterns to form larger patterns, and an hierarchical system for defining changes to a business process.

Change Primitives

We can effectively change a process model by adding or removing any element of this process model: activities, connections, gateways etc. Because a process model can be viewed as a directed graph, and each element is either a node or an edge, we restrict ourselves semantically to the following Change Primitives: *add node*, *remove node*, *add edge* and *remove edge*.

Adaptation Patterns

Because single Change Primitives carry little semantic meaning we use a method to combine them into high-level change operations, such as the insertion, deletion or manipulation of an entire process fragment. We use the term *process fragment* to describe a connected component of the process graph with a single input and output. High-level changes are thus effectively structured as a series of Change Primitives. Figure 3.1 shows the high-level change operation of parallelizing two sequential activities by adding two gateways. The individual change primitives that were applied to achieve this change are:

Remove Edge Start - Activity A
Remove Edge Activity A - Activity B
Add Node Parallel Gateway (id:par-gate-1)
Add Edge Start - par-gate-1
Add Edge par-gate-1 - Activity A
Add Edge par-gate-1 - Activity B
Remove Edge Activity B - Activity C
Add Node Parallel Gateway (id:par-gate-2)
Add Edge Activity A - par-gate-2
Add Edge Activity B - par-gate-2
Add Edge par-gate-2 - Activity C

High-level changes such as adding a process fragment or removing one can be seen as the building blocks for ad-hoc changes. These recurring changes that can be described in an abstract way are called Adaptation Patterns. Table 3.1 lists the thirteen adaptation patterns for modifying a process model that Reichert and Weber [29] classified, in five categories.

3.1.2 Constraints

When applying an ad-hoc change there are a number of things that need to be taken into consideration to prevent the occurrence of errors. We discuss the concepts of soundness, state compliance and

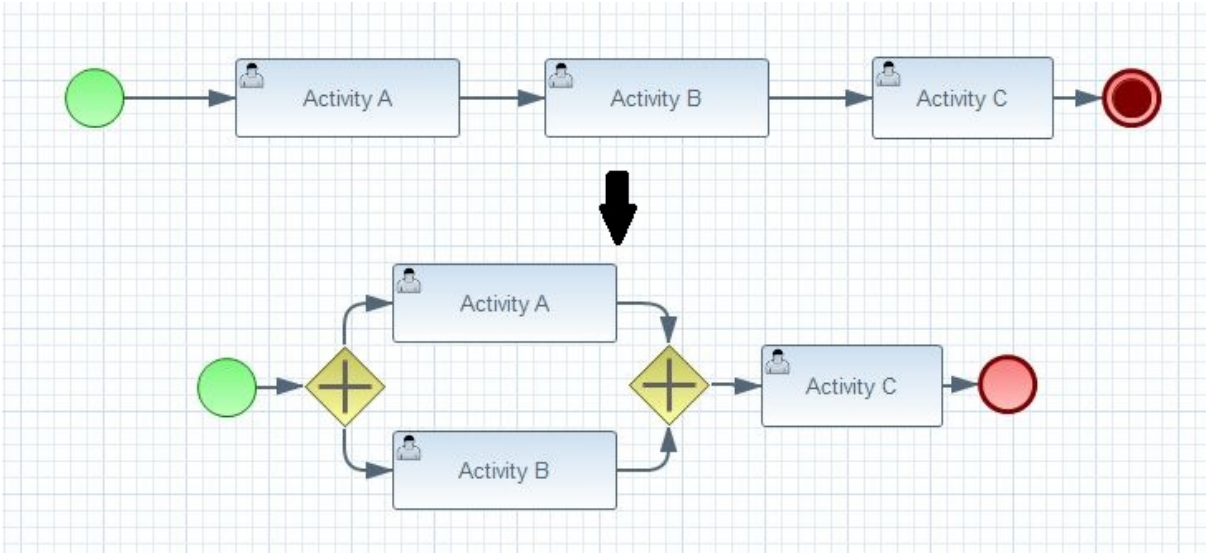

Figure 3.1: The high-level change operation to parallelize Activities A and B.

Pattern Category	**Pattern**	
Adding or deleting Process Fragments	AP1:	Insert Process Fragment
	AP2:	Delete Process Fragment
Moving or replacing Process Fragments	AP3:	Move Process Fragment
	AP4:	Replace Process Fragment
	AP5:	Swap Process Fragment
	AP14:	Copy Process Fragment
Adding or removing Process levels	AP6:	Extract Subprocess
	AP7:	Inline Subprocess
Adapting control dependencies	AP8:	Embed Process Fragment in loop
	AP9:	Parallelize Process Fragments
	AP10:	Embed Process Fragment in conditional branch
	AP11:	Add control dependency
	AP12:	Remove control dependency
Change transition condition	AP13:	Update condition

Table 3.1: Adaptation Patterns for Changes in Process Models [29].

dynamic change correctness. Together, these ensure that ad-hoc changes have no erroneous side-effects on the process instance and models. Ad-hoc changes shall only be applied when they preserve these properties.

Soundness

A sound business process is one that adheres to the syntactic rules of BPMN as defined in the meta-model [30]. For example: edges have one starting node and one ending node, and all nodes have exactly one input edge and one output edge, with the exception of gateways that may have more. Furthermore, in a sound business process we can always advance to another node, deadlocks do not exist, and every node has a trace to a terminating node [29] [31]. The soundness of a business process resulting from an ad-hoc change is vital if we want to avoid run-time errors.

State Compliance

The state of a process instance includes which activities have been completed, which are enabled, and which are running. Together they form the execution trace for a process instance. When applying an ad-hoc change, we have to make sure that the execution trace of the process instance is compatible with its model, since the record of how the process was executed cannot be modified. To enforce this immutability, we use the definition of State Compliance:

State Compliance *"Let I be a process instance with execution trace σ_I. Further, let T be a process model. Then, I is state-compliant with T if σ_I is reproducible on T."* [29]

If we only apply ad-hoc changes that adhere to state compliance, then we can ensure that the history of process instances is not changed. In Figure 3.2 we can see the process model of a treatment and its ad-hoc change. It can be seen that in I_1 there is no problem as the execution trace can be executed fine. In I_2 we see that the injection of the activity "Test for Allergies" invalidates the execution trace. Thus resulting in a process that is not state compliant, and therefore the ad-hoc change cannot be executed. Instance I_3 has no problem recreating the execution trace and is thus state compliant. However, the insertion of the "Test for Allergies" activity means that the "Prepare Patient" activity is no longer the next activity in line, and thus the collection of enabled activities needs to be changed.

Dynamic Change Correctness

If an ad-hoc change moves activities through a process instance, then we must take care to preserve the soundness of that process instance. In Figure 3.1, if we assume that Activity A was enabled, then after the change, Activity A will be enabled but Activity B will not, leading to a deadlock or a Dynamic Change Bug. Any ad-hoc change can only be applied if the resulting instance is a valid instance of the changed underlying process model. For example, an ad-hoc change cannot be applied to a process instance that has progressed too far, where applying the ad-hoc change would introduce a dynamic change bug. We give below the definition of Dynamic Change Correctness [29]:

Dynamic Change Correctness *"Let I = (S, σ_I) be a process instance running on a sound process model S and having execution trace σ_I. Assume further that S is transformed into another sound process model S' by applying change δ, i.e. $S[\delta > S'$. Then:*

- *δ can be correctly applied to I if I is state compliant with S'.*
- *Assume I is state compliant with S'. When applying δ to I, correct activity states of I on S' can be obtained by applying σ_I to S'; i.e., by logically replaying the events captured by σ_I on S' in their original order."*

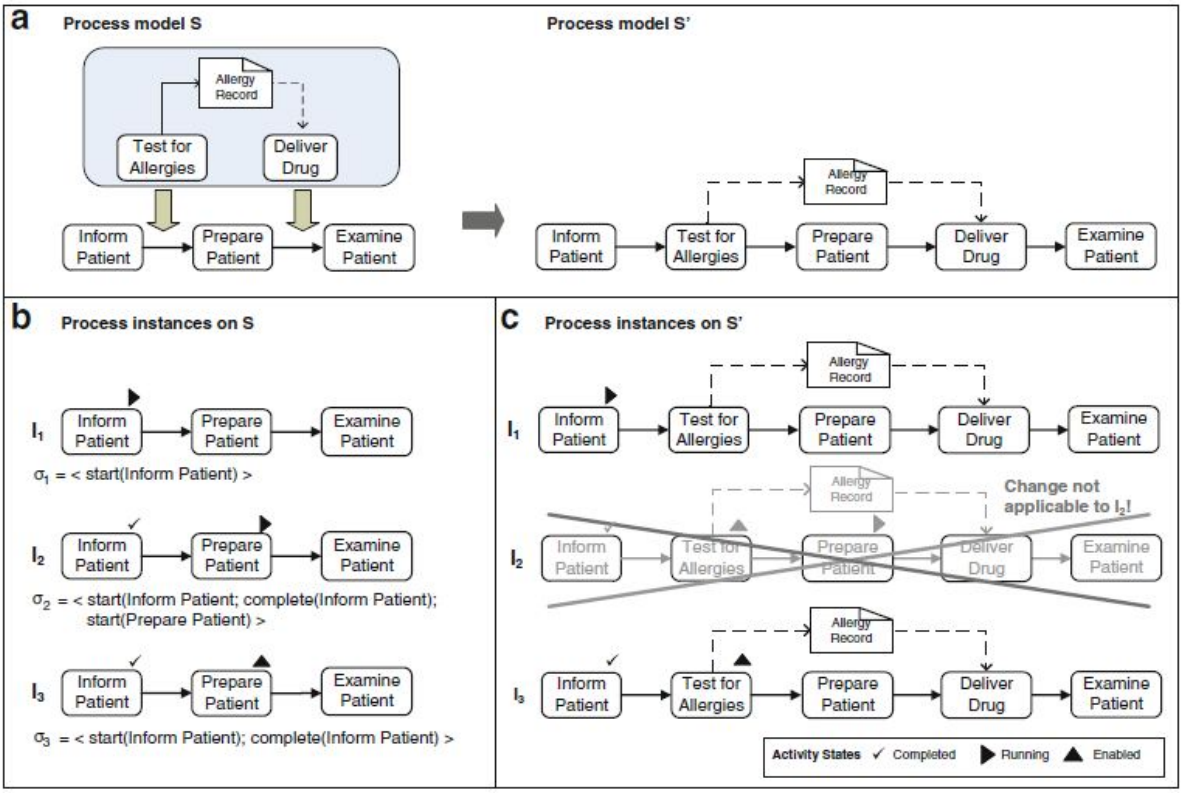

Figure 3.2: An image showing how an ad-hoc change is enacted or not because of state compliance and. [29]

Dynamic Change Correctness thus ensures that an ad-hoc change can only be executed if the result of that change is a valid state for the underlying changed process model. With these two notions we can make sure that ad-hoc changes will only take effect when they result in valid process instances, preventing run-time errors that disturb the operation of the BPMS.

3.2 BPMS Components

To incorporate the functionality of ad-hoc changes into a BPMS, we look at the different elements of a standard BPMS, and determine if they are affected by this change, and if so, how they are affected. Figure 3.3 shows the architecture of a standard BPMS. We then look at any additional elements that may be necessary to incorporate ad-hoc changes.

Process Modeling Tools

Standard Process Modeling Tools provide a way to design process models. For our BPMS with ad-hoc changes there are no fundamental aspects that need to be adjusted in the Process Modeling Tool, as we still have to design BPD's within their normal constraints. Furthermore, the effect that ad-hoc changes have on a process model or instance always result in a sound process model or instance. Therefore,

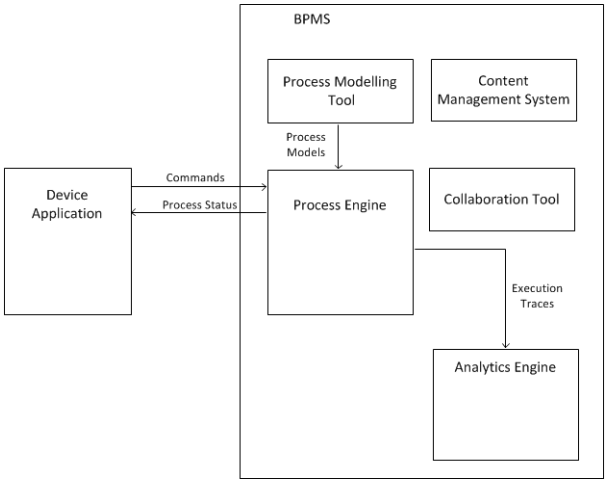

Figure 3.3: The architecture for a standard BPMS

at every point in the execution of a process instance, the BPD should be sound. What changes can happen and under which circumstances can be informed separately to the Process Engine.

In Figure 3.4 we see a screenshot of the Eclipse IDE plug-in from the Open-Source JBoss jBPM BPMS. The simple drag-and-drop interface provides an easy way to construct BPDs. It stores its BPDs as XML files according to the BPMN meta-model defined by the OMG [30]. The interface and standardized format suit our needs for a Process Modeling Tool. As a result, no changes are required in the Process Modeling Tool to function with ad-hoc changes.

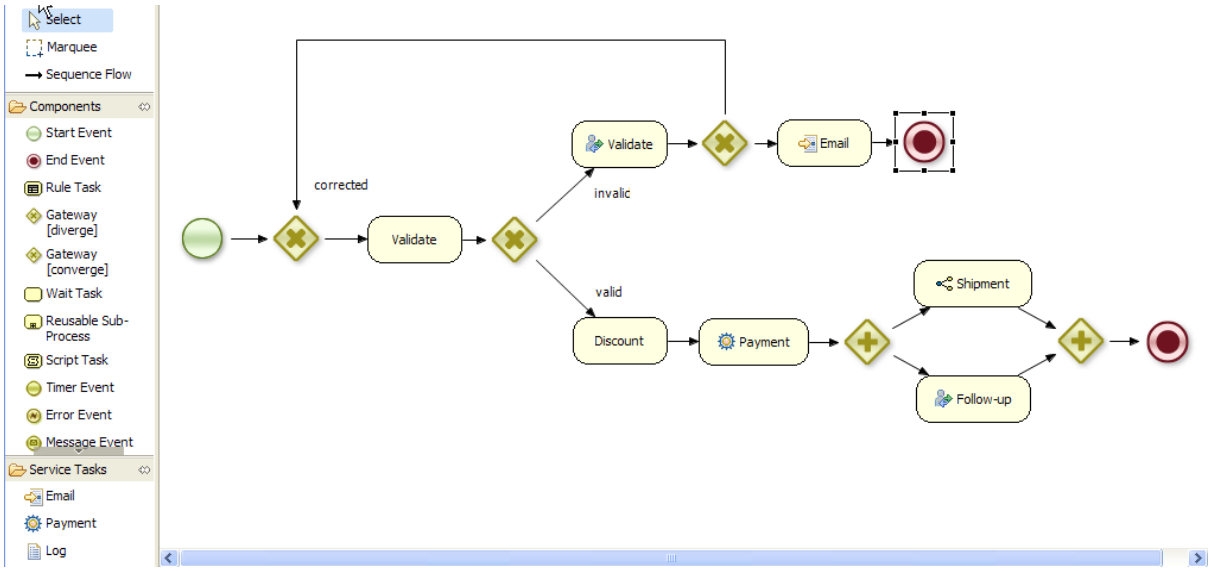

Figure 3.4: The jBPM Process Modeling Eclipse plug-in [32]

Process Engine

In a standard BPMS the Process Engine spawns new process instances from defined process models and keeps track of their execution. Participants can perform activities and the process advances. When the capability to apply ad-hoc changes on the process instances is added, a few extra requirements arise:

1. Defining Possible Changes To define possible changes we make use of an Event Processing Language (EPL) as described in Section 2.3. This allows us to define a change and under what circumstances this change needs to be applied. This is done via the Event-Condition-Action pattern. Listing 3.1 shows an example of an ECA definition of a change in pseudo code.

Therefore we need to add an interface to the Process Engine that executes these changes when called upon. These changes should be stored in a way that facilitates easy comparison and analysis. A suitable way to do this is to store them as incremental changes to the initial BPD.

2. Applying Ad-hoc Changes: To decide when to apply ad-hoc changes to a process instance, a Stream Processor is added to the BPMS. This Stream Processor uses information streams and the change definitions described in the EPL to decide when a change should be applied. It then calls the Process Engine, which in turn can execute the changes on the relevant process instance(s).

```
Event:
  Calendar event has started
Condition:
  Employee A is present at the calendar event
Action:
  Disable the activity "Meet with Employee A"
```

Listing 3.1: An example ECA rule

Furthermore we have to comply with the principles of soundness, state compliance and dynamic change correctness introduced in Section 3.1. The responsibility of applying ad-hoc changes only when these concepts are not violated lies with the process engine, which must reject any ad-hoc changes that do not comply with these principles.

Business Analytics

We assume that process instances remain sound throughout execution, even if ad-hoc changes are applied. We therefore have no extra requirements for the Business Analytics, and can use existing

Business Analytics tooling. Different traces through process instances that have had the exact same ad-hoc changes applied, and thus have the exact same BPD graph, may still be analyzed as equal processes.

However, this might lead to problems when a business process is defined with many possible ad-hoc changes. As we increase the number ad-hoc changes, the total number of possible unique BPDs resulting from these changes can grow exponentially with the number of ad-hoc changes. Having this many different unique types of BPDs can render the analytics useless, as even in a large organization there might not be enough process instances which a specific BPD to perform meaningful analyses.

There are, however, options for meaningful analyses even when two process instances do not have the same underlying BPD. In Figure 3.5, P_1 and P_2 are the result of two different ad-hoc changes applied to *P*. Since their BPDs are different, instances of P_1 are not comparable to instances of P_2, and no analysis can be done. But if we look more closely, then we can see that the first parts of the BPDs of P_1 and P_2 are the same. As a result we can still perform some analysis on those parts of process instances that have BPDs that are equal. This requires that the analytics tooling can infer which parts of two instances are equal.

Therefore, the analysis methods used by the Analysis Tools in a standard BPMS are viable for our BPMS with ad-hoc changes. However, more specific methods can be developed to provide even better analysis for process instances with differently changed BPDs.

Content Management System

The requirements for a Content Management System do not change for a BPMS with ad-hoc changes. Files and information are still stored for each process model and process instance when required. We assume that the identity of a process instance does not change, split or merge when executing an ad-hoc change, therefore, we do not require any extra functionality from the Content Management System.

Collaboration Tools

Similar to the Content Management System, the Collaboration Tools need no adjustment either. Participants in process instances can still communicate and collaborate with each other. An ad-hoc change made to a process instance has no effect on the ongoing collaboration between participants.

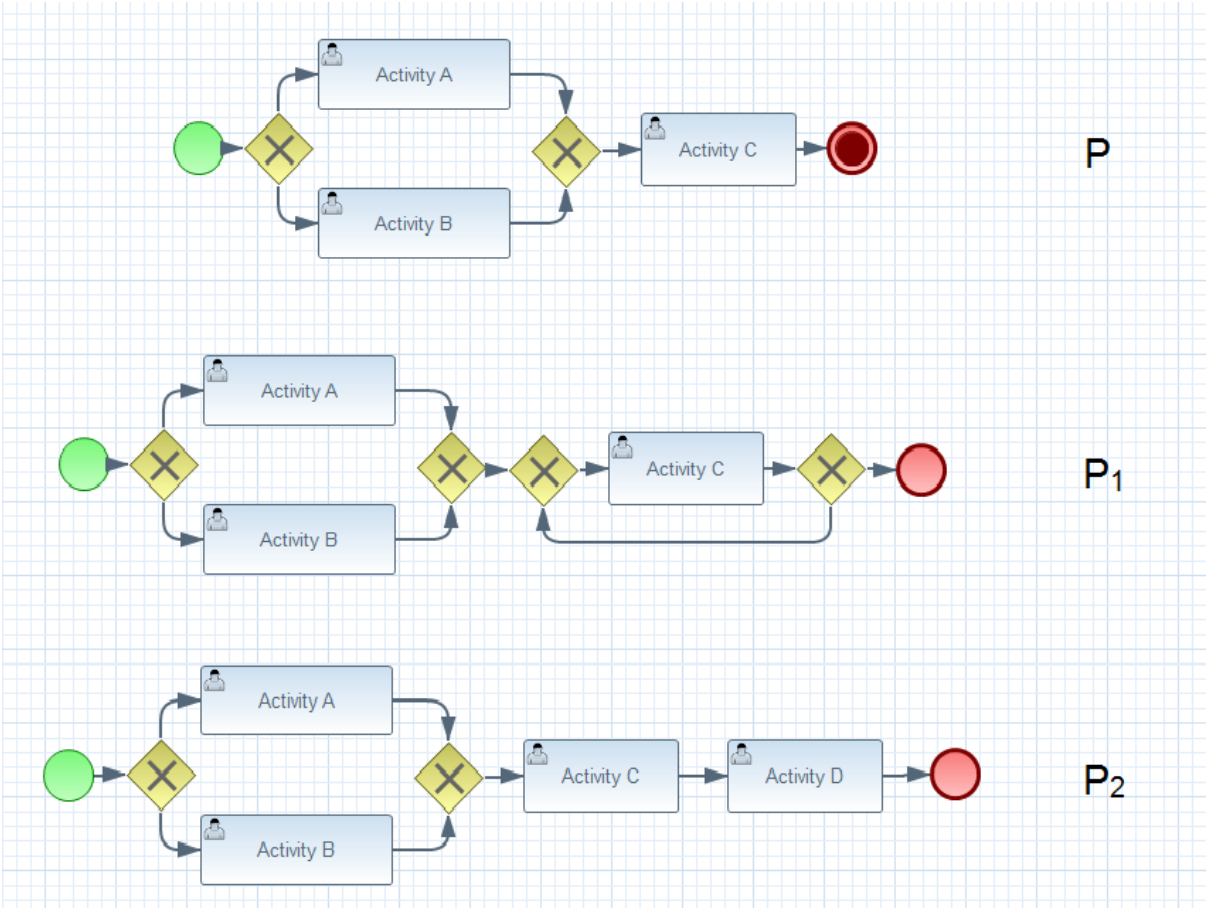

Figure 3.5: A Process P with two possible changed processes P1 and P2.

Stream Processor

A Stream Processor is necessary in a BPMS with the functionality to apply ad-hoc changes. In Section 2.3, we state that the Stream Processor receives streams of events generated by contextual changes. These streams can be supplied by the BPMS itself, but more useful information can be retrieved from other sources, such as the mobile devices of the participants. The Stream Processor uses a predefined EPL description to apply the defined ad-hoc changes to the process engine based on a stream of events.

Device Application

In a regular BPMS there are many ways for the users that are involved in a process to interact with the system. Normally, users interact with the BPMS through an application, which may be through a web interface or in a specially designed mobile application.

In our BPMS we collect contextual information from the mobile devices of the users. The device application thus needs to be extended to include some information gathering functionality.

3.3 High-level Architecture

Our ad-hoc BPMS has been designed as an extension of a regular BPMS. Figure 3.6 shows that only a few extra components are necessary, and a few components have to be modified (in green), and that the architecture is fully backwards compatible with a regular BPMS (in black).

The *Device Application* is situated on the user mobile device. Current BPMS might even have such an application with an integrated user interface. In our ad-hoc BPMS this application also aggregates and relays the contextual information. The device application can be opened at will to interact with the system and complete tasks. In the background, information is continuously gathered and sent to the system. The ad-hoc BPMS further adds a *Stream Processor* that receives the information streams that the *Device Application* sends, and acts according to the given ad-hoc change specification.

The *Process Engine* is modified to allow for ad-hoc changes of process models and process instances. The *Analytics Software* is extended to deal with modified process instances. Since both the *Process Engine* and *Analytics Software* only add functionality to their regular counterparts, this architecture is a proper extension of the regular BPMS.

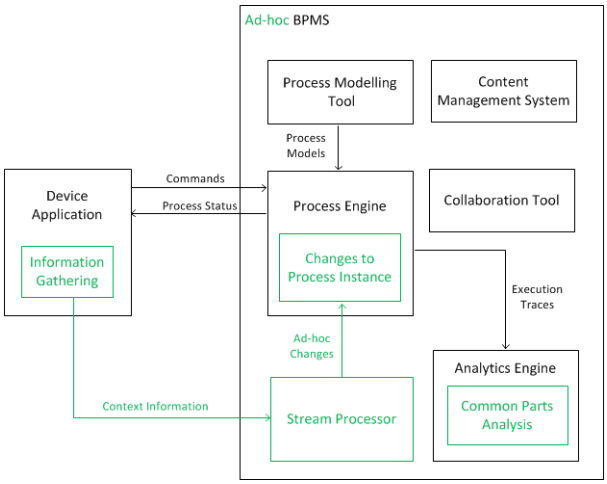

Figure 3.6: A high-level architecture for an ad-hoc BPMS.

3.4 Examples

In Section 3.2 we discussed how BPDs do not fundamentally change when used in a BPMS that can apply ad-hoc changes. However, it can be beneficial to develop the BPD and the accompanying ad-hoc changes in parallel, as this gives much more control over the process. In this section we highlight a few examples of possible ad-hoc changes supported by contextual information.

It is not a coincidence that these examples mainly involve the inclusion or exclusion of employees based on contextual information. Mobile devices are user-centric and thus have quite a lot of information about their users and their direct environments.

Office Availability

In this example an employee has to get approval for some activity by any one of his three superiors. When his request is approved, the employee can continue and carry out that activity. In Figure 3.7 we can see this example where the employee has to get approval from Superior A, B or C. In this case, there might be an ad-hoc change description that describes the deletion of an activity if the relevant superior is not available at the moment. In the case of Figure 3.7, Superior B is unavailable, thus his option is removed from the process instance at runtime.

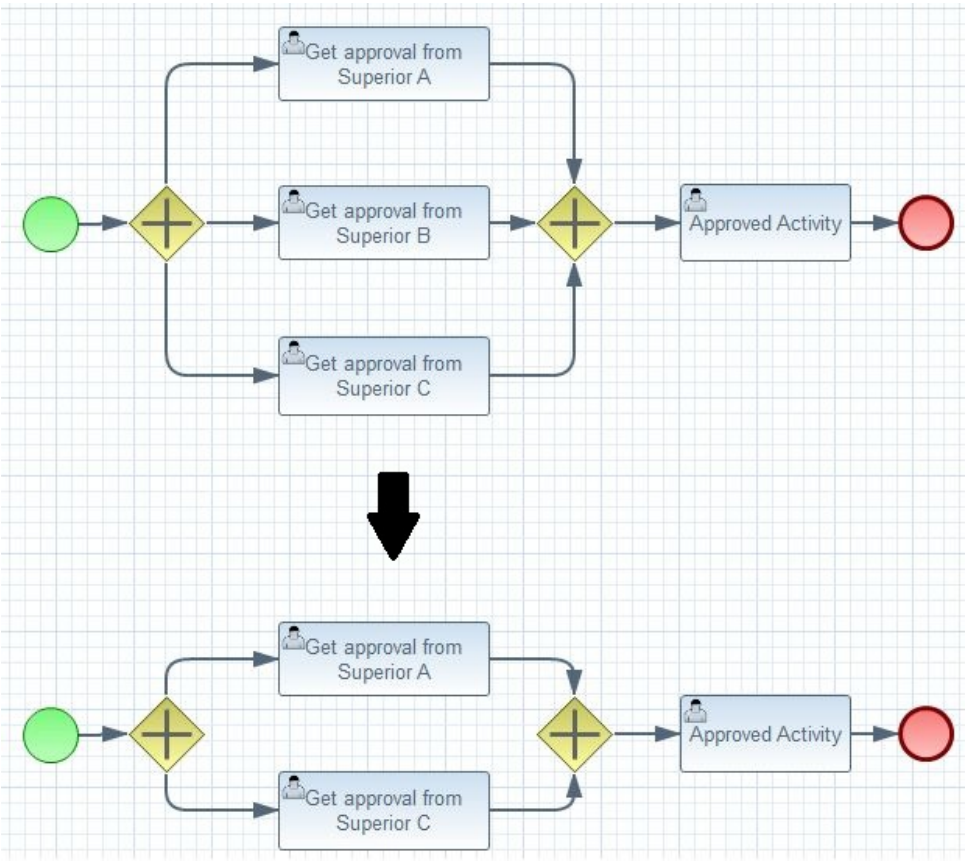

Figure 3.7: Ad-hoc change for the Office Availability example

Task Assignment

A manager assigns an activity to an employee. The employee can always refuse the activity, prompting the manager to assign a new employee. However, if the employee is not sufficiently trained, he might

follow a training, allowing him to complete the activity. Figure 3.8 shows an example where we can see that the process instance is modified with an activity for the employee to follow training, after which the process is restored to its original state, and the original activity can be completed.

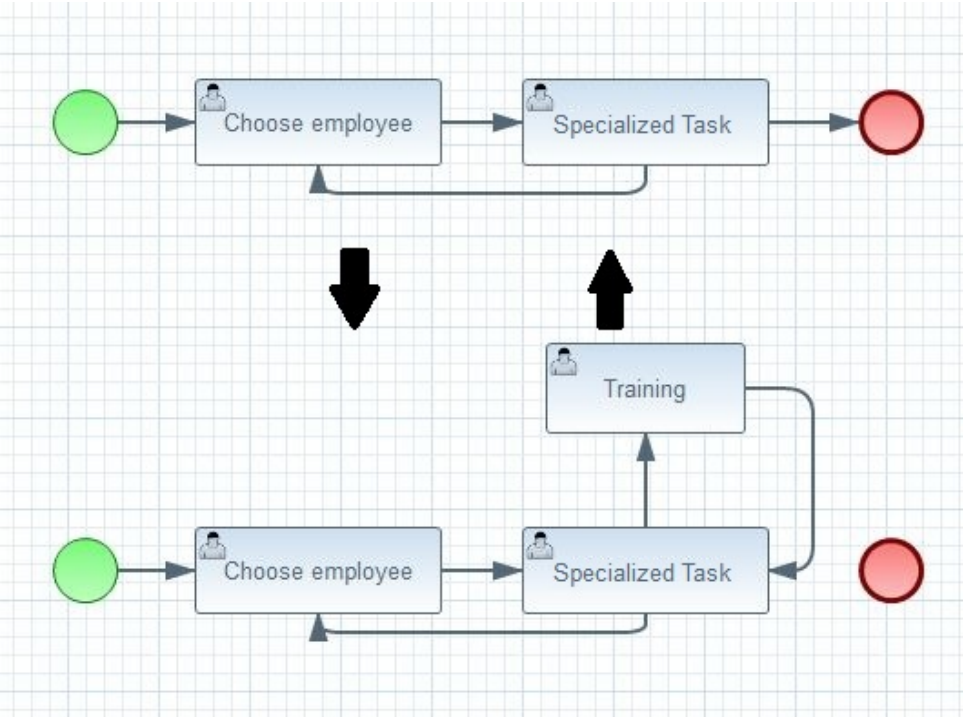

Figure 3.8: Ad-hoc change for the Task Assignment example

Meeting Location

If a meeting has to take place between two parties to sign a contract, then we could imagine a business process instructing both parties to meet and sign the contract. In the example of Figure 3.9, locational data has shown that both parties that are supposed to meet are far away from each other, and even though the actual signing still has to be done personally, the discussion about the contents of the contract might possibly be done via teleconference.

An ad-hoc change can be defined, that offers another separate option to scheduling a meeting and signing the contract. Figure 3.9 shows that an extra path is added by which instead of a meeting to discuss and sign the contract, the discussion is done via teleconferencing. The actual formality of signing is postponed to a later date as a separate action.

Emergency Response Volunteering

In an emergency response unit, volunteers might be called to the scene of an accident. A business process can be used to call on the volunteers who should respond to the scene. If too many volunteers

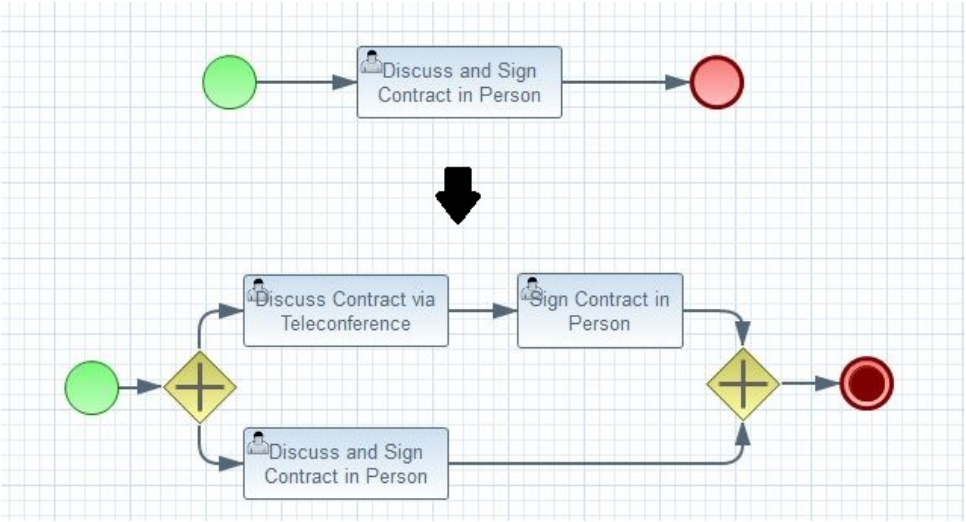

Figure 3.9: Ad-hoc change for the Meeting Location example

are called, the location may become too crowded to be effective, and if too few are called, insufficient manpower may be available to support the professional staff. Individual volunteers thus have to be called to the scene, and this can be done manually by phone or automatically via push messages.

Given a number of volunteers, we can construct a business process that requires them all to be called to the scene. We can then define an ad-hoc change that removes any ineligible volunteers based upon contextual information such as location, availability and training. Furthermore we can remove or add as many volunteers as we need to reach a desired number, possibly using margins to overcome unknown variables.

Figure 3.10 shows how such a business process would be transformed to reach the desired outcome. When five volunteers are on record and only 2 are needed, 3 can be eliminated from the process by any number of reasons. In this example, John might be ill, and Maggie and Mike are just not needed at the time.

3.5 Conclusion

In this chapter we discussed a predefined structure to hierarchically define ad-hoc changes, both in its minimal form as Change Primitives as in an abstracted form in terms of Adaptation Patterns. The principles of state compliance and dynamic change correctness can be used to ensure that any ad-hoc changes that are executed on a process instance guarantee the soundness and correctness of that process instance.

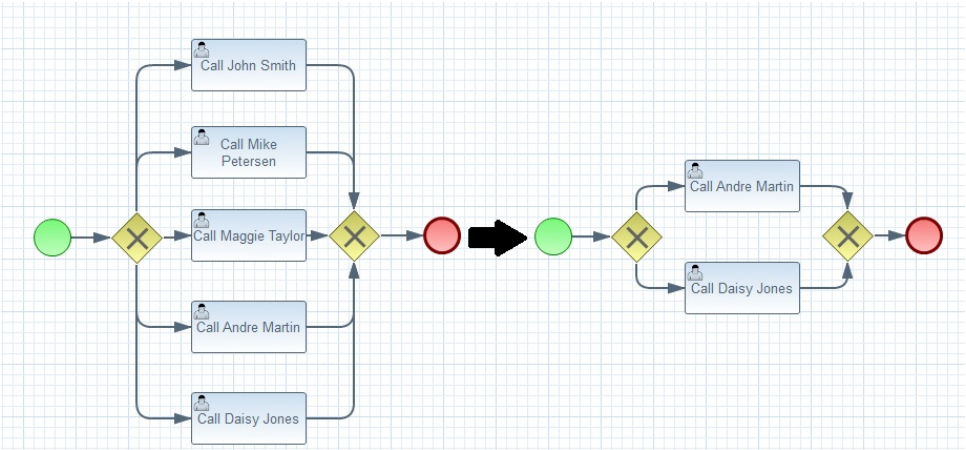

Figure 3.10: Ad-hoc change for the Emergency Response Volunteering example

We further looked at the components of a regular BPMS and assessed how they would change when the ad-hoc changes functionality is added to the system. We showed that a regular BPMS can be extended to support ad-hoc changes. Most notably the Process Engine requires an extension to deal with changing process models and instances. Furthermore, to drive these changes, the device application gathers contextual information and sends it to an additional component called Stream Processor. This Stream Processor executes the desired ad-hoc changes on the Process Engine. Finally an extension to the Analytics Engine can be made to improve the possible analysis of execution traces of changed process instances.

We finally gave a few examples to show how this new functionality can be used to help adapt business processes to changes in the environment. Whilst current business processes might not be directly ready for ad-hoc changes, they can be rewritten to be receptive to ad-hoc changes. It is also shown that a wide variety of changes is possible, even in a seemingly straightforward environment as an office with meetings. Furthermore, even though the examples are centered around contextual information from mobile devices used by users, the concept of ad-hoc changes can be taken in a broader view, using other sources of contextual information.

Chapter 4

AdHoc BPMS

To validate our approach and high-level architecture, a proof of concept was developed, which we called the AdHoc BPMS. In this chapter we detail the system requirements of the proof of concept implementation, show several use cases and present the architecture of the proof of concept. We show how the individual software components are designed and implemented, and give an example showing how the final system works. We conclude the chapter by presenting some limitations of the developed proof of concept, and argue why these limitations do not pose a problem when a full implementation is considered.

4.1 Scope

Because this proof of concept is developed to validate the concept and architecture of ad-hoc changes in a BPMS, we limit the scope of this proof of concept to those elements that are influenced by ad-hoc changes. Figure 4.1 shows the boundaries of this proof of concept in a blue dashed line.

Contained within this proof of concept are the core components necessary to facilitate the application of ad-hoc changes to process instances: the mobile application, the process engine and the stream processor. Each of these are discussed in more detail in this chapter.

Outside of the scope of this proof of concept are the collaboration tool and the content management tool as they are not coupled to the process engine or mobile application. We do not develop our own process modeling tool either, since we chose to use the jBoss jBPM process modeling Eclipse IDE plug-in [32]. This tool can output XML files of the modeled BPD's that we then use as input for our proof of concept.

Finally we leave the analytics engine out of the scope of this proof of concept. However, as the recorded information on business process instance execution is fundamentally the same as without ad-hoc changes, any standard business process analysis should be able to do the job. The only caveat is that the data generated by the process engine is delivered in a format that can be imported by the analysis tool of choice.

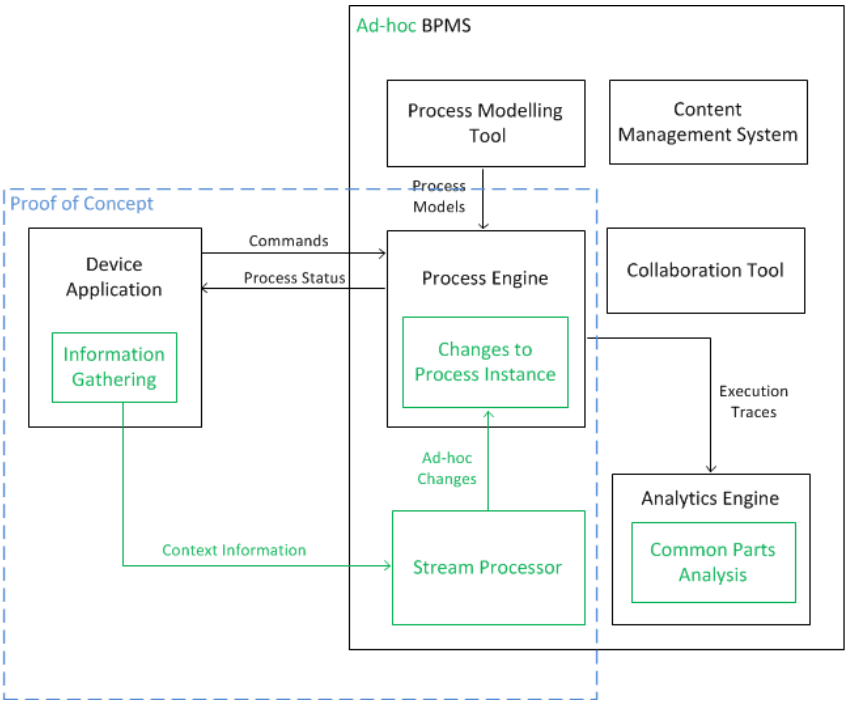

Figure 4.1: The scope of the proof of concept compared to the complete architecture.

4.2 System Requirements

To develop a proof of concept we first specify the system requirements. Because we are building an extension to a standard BPMS we start by specifying the system requirements for a standard BPMS. We then specify the extra requirements that are necessary to support the ad-hoc changes capabilities.

4.2.1 BPMS Requirements

Figure 4.1 shows the two existing components that we are going to extend in black, namely the process engine and the mobile application. We discuss the functionalities that these two components need to provide and from those we derive the system requirements.

The *Process Engine* is the central component of a BPMS, it governs the running instances of pre-

defined business processes. It does this by starting new instances, managing the activities that are possible at any one time, and allowing users to complete activities on instances. We thus come to the following system requirements:

R1: *The process engine shall be able to import business processes designed by the modeling tool.*

R2: *The process engine shall provide a way to start new process instances.*

R3: *The process engine shall provide an interface to expose actions on running process instances.*

R4: *The process engine shall provide an interface to receive actions that are to be enacted on running process instances.*

R5: *The process engine shall change the state of a process instance according to the received actions.*

The *Mobile Application* functions as the end user interface to the system. As such it provides a way to log in to the system. Furthermore, to interact with the system we must be able to receive all activities that are relevant to the user that is logged in from the process engine, as well as being able to enact those activities and send those actions to the process engine. We thus come to the following system requirements:

R6: *The mobile application shall provide a way for a user to log in to the system.*

R7: *The mobile application shall provide a way to retrieve all possible actions relevant to the logged in user from the process engine, and present them in a graphical user interface.*

R8: *The mobile application shall provide a way for a user to take an action, and send it to the process engine.*

4.2.2 Ad-hoc BPMS Requirements

We next add the system requirements that are necessary to support context-driven ad-hoc changes. Figure 4.1 shows the components that are needed to do so in green. We therefore have to define extra system requirements for the process engine and the mobile application, and define system requirements for the stream processor.

To be able to apply ad-hoc changes, the *Process Engine* needs to be able to change process instances.

R9: *The process engine shall be able to apply ad-hoc changes on process instances.*

The contextual information used to drive ad-hoc changes is collected in the *Mobile Application*. It needs to periodically collect the contextual information and send it to the stream processor. In this proof of concept we restrict ourselves to calendar and location information.

R10 *The mobile application shall be able to aggregate contextual information and send it to the stream processor.*

The *Stream Processor* receives contextual information from the Mobile Applications of the BPMS end users. It matches the contextual information to a predefined EPL specification, and sends the ad-hoc changes defined in the EPL to the process engine.

R11: *The stream processor shall be able to import an EPL description.*

R12: *The stream processor shall be able to match incoming contextual information to ad-hoc changes described in the EPL description.*

R13: *The stream processor shall be able to send ad-hoc changes to the process engine.*

4.3 Use Cases

We define several use cases to further describe the functionality of the system. We loosely followed the format described by Fowler [33], namely: give a description for each use case, a number of steps that dictate the interaction between actors and the system, and a possible list of extension steps. We show a use case diagram, and relate use cases to the requirements specified in Section 4.2.1.

4.3.1 U1: System Startup

Description: At the startup of the system, BPMN and EPL specifications are supplied to the system. The system starts up the process engine and the stream processor, and populates them with the BPMN and EPL specifications, respectively.

Steps:

1. *Administrator User* locates the BPMN and EPL files that are necessary for the BPMS.
2. *Administrator User* starts the system and provides the BPMN and EPL files.

4.3.2 U2: Starting a Process Instance

Description: When the system is started, an administrative user can use the back-end interface to start a process instance for any known business process description.

Steps:

1. *Administrator User* opens the interface to the BPMS.
2. *Administrator User* starts a new instance of the desired process.

4.3.3 U3: Logging In

Description: When an end user starts the Mobile Application, he is prompted for his name, and the location (IP address and port number) of the BPMS server. After entering the information and sending it to the server, the user is logged in.

Steps:

1. *End User* opens the Mobile Application.
2. *End User* enters his name, and the location of the BPMS Server.
3. *End User* connects to the Server and is logged in.

4.3.4 U4: Task Completion

Description: A logged in user can ask the process engine to list all available activities. He is first prompted to select a process for which he wants to complete an activity. After choosing a process, all available activities for that process are shown. The user can then select an activity and send a completion request to the process engine. The process engine receives this request and modifies the corresponding process instance accordingly.

Steps:

1. *End User* opens the Mobile Application.
2. *End User* selects the desired process from the list.
3. *End User* confirms the selected process
4. *End User* selects the desired action from a list.
5. *End User* confirms the completion of the selected action.

Extension:

1a. *End User* logs in to the server.

4.3.5 U5: Context Information Update

Whenever a context source detects a change in the context of a user device that is running the mobile application, a change notification is sent to the stream processor. This process is automated and requires no user intervention. Whenever an update is received by the stream processor, it is checked against the list of predefined changes. If the conditions of the change are met, then the corresponding change is applied to the process instance in the process engine.

Steps:

1. *Context Source* detects a context change.
2. *Mobile Application* sends change event to *Stream Processor*.
3. *Stream Processor* tries to match event to the defined EPL specifications.

Extension:

4a. *Stream Processor* applies an ad-hoc change defined in the EPL specification to the *Process Engine*.

4b. *Process Engine* changes the relevant process instance.

4.3.6 Use Case Diagram

Figure 4.2 shows the Use Case Diagram for the AdHoc BPMS System. The Administrator is linked to Use Cases 1 and 2 and the End User is linked to Use Cases 3 and 4. The actor for the Context Source is linked to Use Case 5.

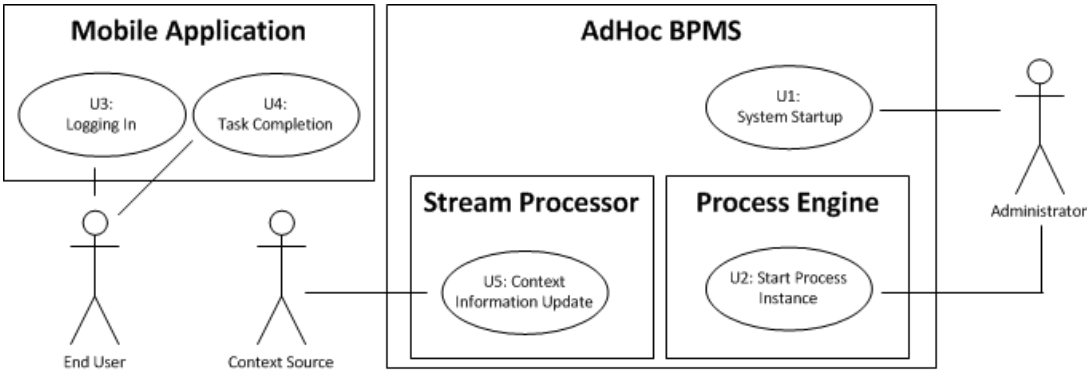

Figure 4.2: Use Case Diagram for the AdHoc BPMS

4.3.7 Relating Requirements and Use Cases

We can relate the system requirements and the defined use cases. Table 4.1 shows how each use case corresponds to at least one requirement, and how all requirements correspond to at least one use case.

	R1	R2	R3	R4	R5	R6	R7	R8	R9	R10	R11	R12	R13
U1	X										X		
U2		X											
U3						X							
U4			X	X	X		X	X					
U5									X	X		X	X

Table 4.1: A relation between the defined requirements and use cases.

4.4 Software Architecture

Our proof of concept implementation consists of a client and a server. The client is an Android application, and the server is a Java program. In this section we detail the overall architecture of these two components. We chose to develop a new mobile application as well as a new BPMS server, there are a few reasons why we chose to do so. Mainly, existing solutions have full BPMN implementation, and this would require us to support a full BPMN implementation as well. Furthermore, existing solutions may also have extra capabilities that are irrelevant to this research, and handling those extra capabilities would needlessly complicate the proof of concept.

A new mobile application was built, it is a simple Android application that allows interaction with the server in two ways. First, it provides the regular BPMS functionalities to retrieve and complete activities. Second, it aggregates contextual information and sends it to the server. Figure 4.3 shows the architecture of the mobile application in a class diagram. There are three parts to the mobile application: The activities to handle the regular interaction, the context sources that gather contextual information, and an interface to provide the connection to the server.

The server consists of three components in itself, Figure 4.4 shows the architecture for server in terms of a class diagram. It shows the outline of the three components: the process engine, the stream processor, and the interfaces that provide interaction to the server, and how they interact.

Rather than extending an existing process engine, we built our own process engine that supports ad-hoc changes and a subset of BPMN: tasks, events, gateways and sequence flows.We also built our own stream processor. This component is new in a BPMS and therefore it was necessary to develop

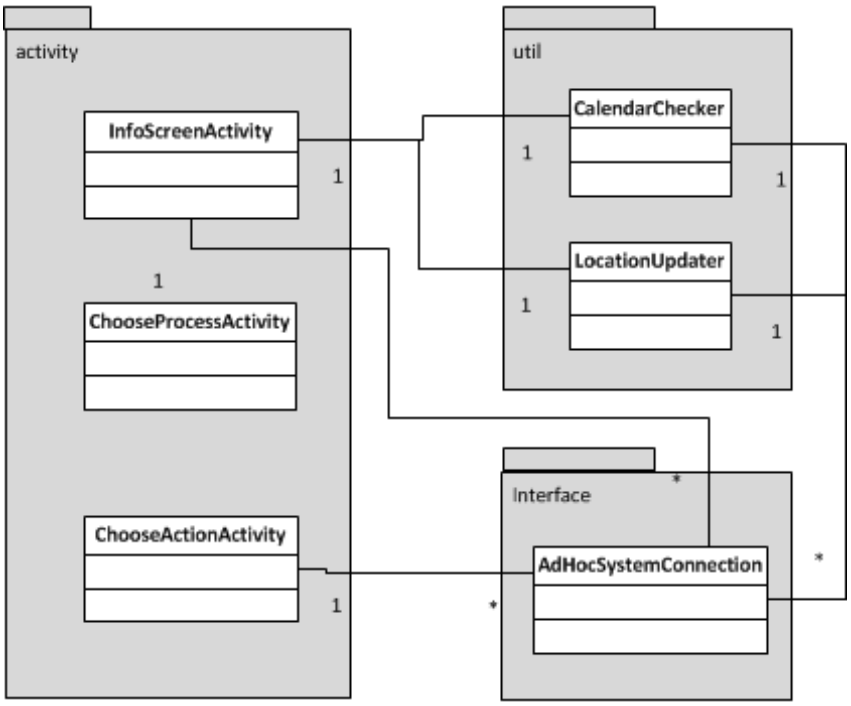

Figure 4.3: Detailed class diagram of the AdHoc mobile application.

our own.

4.5 Implementation

In this section we discuss the design and implementation of the individual parts of the proof of concept: the mobile application, the process engine and the stream processor.

4.5.1 Mobile Application

The Mobile Application serves a twofold purpose. First, it is an app interface to the ad-hoc BPMS. A user inputs his identity, and with this identity all processes and tasks that are assigned to this identity can be retrieved by the application. A simple user interface then lets the user complete any of these tasks, and sends the results to the server. Figure 4.5 shows the process from startup to the completion of an activity. The initial activity requests the username and server location (IP/port). The other two activities allow the user to choose a process and an action through drop-down selectors.

Second, the mobile application is an information harvester. It periodically gathers updates about contextual information and sends these to the server to be processed. This information is then fed

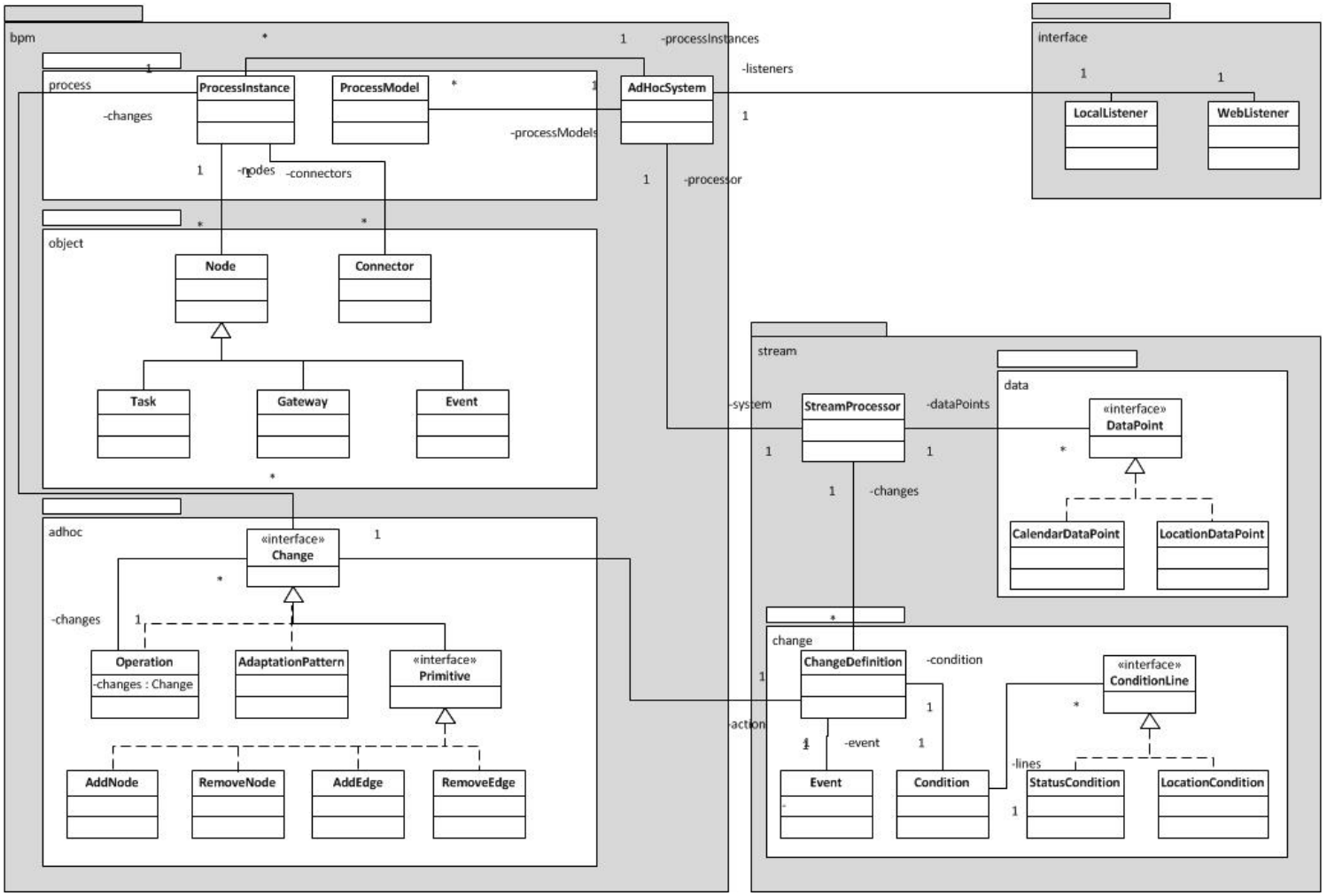

Figure 4.4: Detailed class diagram of the AdHoc system server.

into the stream processor. For the proof of concept, the Mobile Application harvests two sources of information: the user location, which makes use of Android's LocationManager API to retrieve the best locational fix from *GPS*, *Wi-Fi* or *Network*, and the user availability, where the current status of the user is taken from the Calendar Context API and sent to the server as either *busy* or *free*.

Figure 4.5: The general process flow in the AdHoc mobile application.

To send information to and receive information from the server, the application uses standard Java sockets. The types of context information sources and the way they provide their information can be fine-tuned to the requirements of the system, but for this proof of concept we deem location and availability data as sufficient.

4.5.2 Process Engine

The largest extension necessary for a regular BPMS to support ad-hoc changes is in the Process Engine. Whereas a normal Process Engine supports the representation of a business process as a graph, and the ability to keep track of the execution trace that is taken on this process, in the AdHoc system we have to deal with changeable business pocesses, and thus with changable graph representations.

We designed a new way to represent business processes to accommodate for these ad-hoc changes. The primary representation of the BPD of a process instances is therefore not a graph of nodes and edges, but rather a list of immutable change primitives. The initial BPD is represented by the initial changes. Subsequent applied ad-hoc changes are then stored together with the list of previous changes. The execution trace is stored as a list of nodes, and the initial nodes are defined as the start

events in the BPD.

The benefit we achieve from using this list of immutable change primitives, is the ability to construct the current BPD graph by applying the list of changes to an empty graph. In addition, we can easily create a deep copy of the graph where no dependence exists between the original graph and the copy.

Regular BPM execution is achieved by building the representative graph from the list of changes by applying them in sequence. Initially this consists of only the base BPD definition given in the BPMN description, but because ad-hoc changes themselves are also modeled as immutable lists of change primitives and adaptation patterns, throughout execution this list is appended with the applied ad-hoc changes. This graph together with the execution trace allows us to infer possible next nodes in the business process in the same way a regular BPMS would. Completed tasks can be stored in the execution trace, and a new set of possible next nodes can be inferred.

Ad-hoc changes are applied in a step-by-step fail-fast process to ensure the resulting process instance is sound and adheres to state compliance and dynamic change correctness as discussed in Section 3.1. This is achieved as follows:

1: A copy of the BPD graph is made and the ad-hoc change is applied to it; if an adaptation pattern is used the change primitives are inferred here. This reveals any graph integrity erros, such as for example, removing non-existing nodes and edges, adding duplicate nodes and adding edges to non-existing nodes.

2: The soundness of the resulting BPD is checked by ensuring that every node has exactly one incoming and one outgoing edge (except for gateways), as well as checking complete forward and backward reachability through a fixed-point iteration.

3: The original trace is applied and checked on the BPD resulting from the ad-hoc change. To achieve this, the starting point for the new BPD is generated from start events, and the existing trace is applied node by node. If a node existing in the trace does not appear as a viable choice in the new BPD, then this can be traced back to a state incompliance. After applying the change, dynamic change bugs can be identified through the verification of paths between parallel and exclusive gateways.

4: After ensuring that the process resulting from applying the ad-hoc change on the copy is be sound and adheres to state compliance and dynamic change correctness, the ad-hoc change is applied to the actual BPD graph of the desired process instance.

If an incompliance is encountered anywhere in the process, then the application aborted and the resulting change is not applied, leaving the original process instance intact.

4.5.3 Stream Processor

New in the AdHoc BPMS with regard to a standard BPMS is the Stream Processor. At startup, the EPL definitions are fed to the stream processor. These ECA-type changes are then stored, the event and condition are stored as their respective matching variables, and the action, in this case the ad-hoc change, is stored as an immutable list of change-primitives and adaptation patterns.

During execution, the stream processor receives context information updates from users. Upon an incoming update event, the information is stored in the information storage of the stream processor. The stream processor then tries to apply all the known change definitions to all the known process instances. Similar to the process engine, this is a step-by-step fail-fast process to ensure speedy execution. Changes are applied as follows:

1: The name of the process instance is matched against the name of the process in the change definition.

2: The event is matched against the event in the change definition. This encompasses checking the type and the originating user.

3: The condition is matched against the information stored in the stream processor. This includes both availability statuses and distance checks.

4: The ad-hoc change is applied to the process instance in the process engine as described in Section 4.5.2.

The AdHoc Event Processing Language

To describe the Ad-hoc changes to a process, we defined a simple event processing language, which we called AdHoc. This language can be used to describe Event-Condition-Action changes that can be executed on the Process Engine. In this section we discuss the EBNF for this language and describe each of its elements.

Each chane definition file is defined as a $<$*ChangeFile*$>$ clause. It defines the name of the process to which the changes should be applied, as well as the changes themselves. Each change is defined

in a <*ChangeRule*> clause, which in turn is represented as <*Event*>, <*Condition*> and <*Action*> clauses.

⟨*ChangeFile*⟩ ::= `Process` NEWLINE IDENTIFIER NEWLINE (⟨*ChangeRule*⟩+)

⟨*ChangeRule*⟩ ::= ⟨*Event*⟩ ⟨*Condition*⟩ ⟨*Action*⟩ NEWLINE

The <*Event*> clause describes an event. More specifically, it lists which type of event should be matched, and the source of the event.

⟨*Event*⟩ ::= `Event` NEWLINE ⟨*EventLine*⟩

⟨*EventLine*⟩ ::= (`Calendar` | `Location`) IDENTIFIER NEWLINE

The <*Condition*> clause describes the condition that has to be matched for the change to be applied. Any number of <*ConditionLine*> clauses can be used, and the condition will be satisfied when all of the <*ConditionLine*> clauses are satisfied. The AdHoc EPL has two types of ConditionLine clauses: <*StatusLine*> and <*DistanceLine*>. A <*StatusLine*> describes whether a user must be free or busy, and a <*DistanceLine*> describes whether two users should be either closer, or further apart than a specified distance (e.g. in meters).

⟨*Condition*⟩ ::= `Condition` NEWLINE (⟨*ConditionLine*⟩+)

⟨*ConditionLine*⟩ ::= (⟨*StatusLine*⟩ | ⟨*DistanceLine*⟩) NEWLINE

⟨*StatusLine*⟩ ::= `Status` (`Busy` | `Free`) IDENTIFIER

⟨*DistanceLine*⟩ ::= `Distance` (LARGER | SMALLER) INTEGER IDENTIFIER IDENTIFIER

The <*Action*> clause describes the actual ad-hoc changes that can be applied. It is composed of any number of <*ActionLine*> clauses, which will be executed in the order in which they are defined. Each <*ActionLine*> is either a <*PrimitiveLine*> or a <*PatternLine*>.

⟨*Action*⟩ ::= `Action` (⟨*ActionLine*⟩+) NEWLINE

⟨*ActionLine*⟩ ::= (⟨*PrimitiveLine*⟩ | ⟨*PatternLine*⟩) NEWLINE

The <*PrimitiveLine*> clause describes any of the four change primitives. The <*RemoveEdge*> and <*AddEdge*> clauses describe the removal and addition, respectively, of a connection between two nodes. The <*RemoveNode*> clause describes the removal of a node. These are all defined by their internal identifiers.

⟨*PrimitiveLine*⟩ ::= ⟨*RemoveEdge*⟩ | ⟨*AddEdge*⟩ | ⟨*RemoveNode*⟩ | ⟨*AddNode*⟩

⟨*RemoveEdge*⟩ ::= `RemoveEdge` IDENTIFIER IDENTIFIER

⟨*AddEdge*⟩ ::= `AddEdge` IDENTIFIER IDENTIFIER

⟨*RemoveNode*⟩ ::= `RemoveNode` IDENTIFIER

The <*AddNode*> clause is the most complex change primitive rule, as it requires extra information to be present. It describes the internal identifier for the new node, its <*NodeType*>, a <*Description*> and the <*Actors*> associated with the new node. It is also possible to assign no <*Actors*> to a node when all actors should be able to complete it, or when assigning an actor has no semantic value, such as for example, a gateway node.

⟨*AddNode*⟩ ::= `AddNode` IDENTIFIER ⟨*NodeType*⟩ NEWLINE ⟨*Description*⟩ NEWLINE ⟨*Actors*⟩

⟨*Description*⟩ ::= `Description` IDENTIFIER

⟨*Actors*⟩ ::= `Actors` (IDENTIFIER*)

Adaptation patterns are described in <*PatternLine*> clause. The AdHoc EPL supports three adaptation patterns: <*Insert*>, <*Delete*> and <*Swap*>. The <*Delete*> and <*Swap*> clauses use internal identifiers to define which nodes should be deleted or swapped, respectively. The <*Insert*> clause works similarly to the <*AddNode*> clause, but also defines the two nodes between which the new node should be inserted.

⟨*PatternLine*⟩ ::= ⟨*Insert*⟩ | ⟨*Delete*⟩ | ⟨*Swap*⟩

⟨*Insert*⟩ ::= `Insert` IDENTIFIER IDENTIFIER IDENTIFIER ⟨*NodeType*⟩ NEWLINE ⟨*Description*⟩ NEWLINE ⟨*Actors*⟩

⟨*Delete*⟩ ::= `Delete` IDENTIFIER

⟨*Swap*⟩ ::= `Swap` IDENTIFIER IDENTIFIER

Lastly, the <*Nodetype*> clause defines a fixed set of node types that can be used.

⟨*NodeType*⟩ ::= `Task` | `ExGateway` | `ParGateway` | `Event` | `StartEvent` | `EndEvent`

4.6 Conclusion

With the architecture from Chapter 3 a proof of concept was designed and implemented. The chosen scope allowed us to restrict the proof of concept and to focus on the core concepts for ad-hoc change capabilities. Through requirements and use case analyses we defined the systems behaviour of both a regular BPMS, and of one that is capable of applying ad-hoc changes.

We described the software architecture and implementation of the server side and the mobile application. We also designed a simple EPL to support the description of the ad-hoc changes, and showed how ad-hoc changes could be defined in this EPL.

Chapter 5

Validation

To validate our proof of concept, we have defined an example business process and EPL description and use it to validate the capabilities of the AdHoc system. In this chapter we discuss several possible ad-hoc changes to this BPD by showing their EPL description, reasoning when they are applied, or not, and how the BPD changes as a result of their application. We then discuss the liimitations of this proof of concept, and how they translate to a full scale implementation.

5.1 Example Changes

Figure 5.1 shows the graphical representation of the BPD and it is annotated to show the internal id's of the individual elements. To validate the possible changes we have developed an EPL description for the example process with four possible changes. Below we discuss this EPL description and its effects in detail. Listing 5.1 shows the first part of the EPL file, describing that this EPL file contains changes for the example process.

```
Process
    example
```

Listing 5.1: Start of the EPL description for the example process

5.1.1 Removing a Node

This example describes the removal of the node labeled *Task 1* when the (fictional) user *ExampleUser1* is busy. The change is triggered when a *Calendar* event is sent in by *ExampleUser1*, and the condition specifies that *ExampleUser1* must be busy for the action to be applied. The action itself removes the

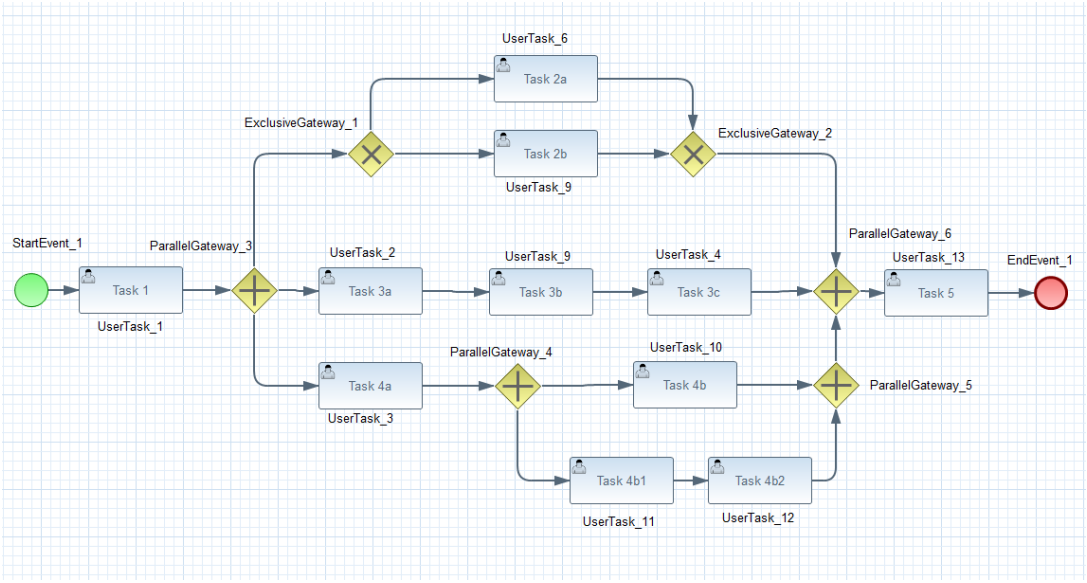

Figure 5.1: Example BPD.

UserTask_1 node and its incoming and outgoing edges, it then connects the nodes *StartEvent_1* and *ParallelGateway_3* directly.

```
Event
    Calendar ExampleUser1
Condition
    Status Busy ExampleUser1
Action
    RemoveNode UserTask_1
    RemoveEdge StartEvent_1 UserTask_1
    RemoveEdge UserTask_1 ParallelGateway_3
    AddEdge StartEvent_1 ParallelGateway_3
```

Listing 5.2: EPL description of Example 1: Removing a Node

As a result of this change, the node labeled *Task 1* is no longer an option, and the choice of actions presented to the user are now changed as seen in Figure 5.3.

If *UserTask_1* is already in the execution trace, then the application of this change fails, as removing a node from the execution trace results in a state incompliance. The resulting BPD after the change is applied correctly can be seen in Figure 5.2.

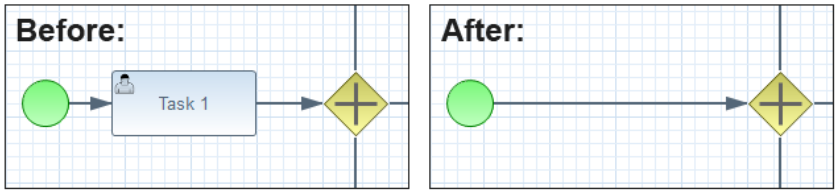

Figure 5.2: BPD change for example 1.

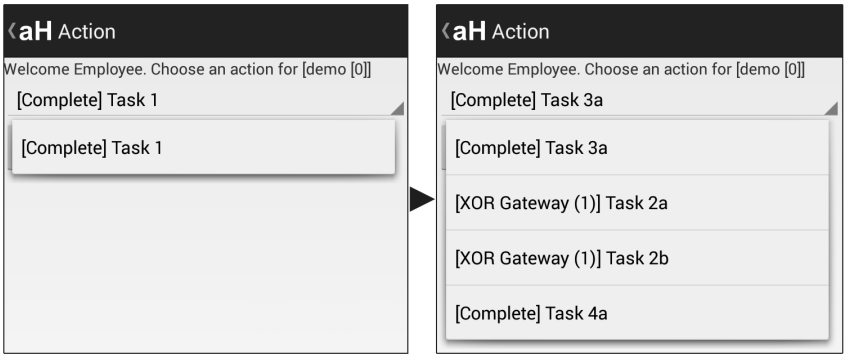

Figure 5.3: Application change for example 1.

5.1.2 Adding a Node

This example describes the addition of a node labeled *Task 2c (new)* to the exclusive gateway already containing the two traces with the tasks *Task 2a* and *Task 2b*. The change is triggered when a *Location* event is sent by *ExampleUser2*, and the condition specifies that the distance between *ExampleUser2* and *Employee* is shorter than 1000 meters. The action itself adds the *UserTask_2c* node to the system, with the description *Task 2c (new)* and the actor *Employee* that can enact it. It furthermore adds the two edges connecting the node to the exclusive gateway nodes.

```
Event
    Location ExampleUser2
Condition
    Distance < 1000 ExampleUser2 Employee
Action
    AddNode UserTask_2c Task
        Description Task 2c (new)
        Actors Employee
    AddEdge ExclusiveGateway_1 UserTask_2c
    AddEdge UserTask_2c ExclusiveGateway_2
```

Listing 5.3: EPL description of example 2: Adding a Node

As a result of this change, the task labeled *Task 2c (new)* is added to the graph as shown in Figure 5.4. Figure 5.5 shows the change in the possible actions that can be chosen in the mobile application.

If *UserTask_6* or *UserTask_9* is already in the execution trace, then this node is still added, as it does not violate soundness, state compliance or dynamic change correctness. However, it will be to no effect, as the exclusive trace between *ExclusiveGateway_1* and *ExclusiveGateway_2* has already been chosen.

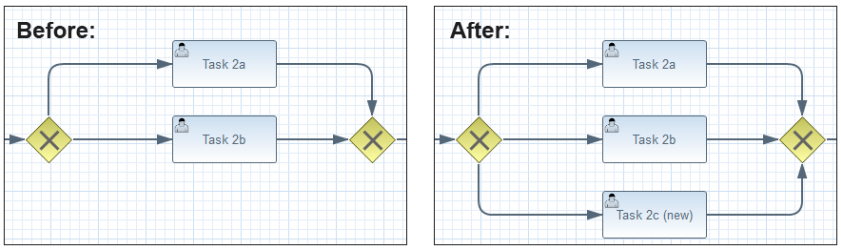

Figure 5.4: BPD change for example 2.

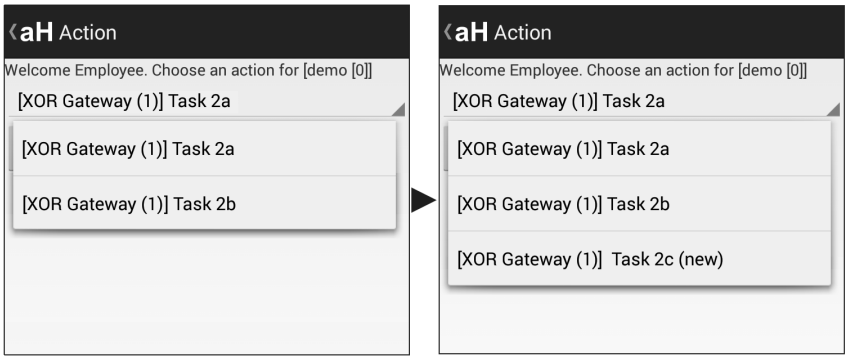

Figure 5.5: Application change for example 2.

5.1.3 Exclusion of two Nodes

This example describes the exclusion of two sequential nodes labeled *Task 3b* and *Task 3c*. The change is triggered when a *Calendar* event is sent in by *ExampleUser3*, and the condition specifies that *ExampleUser3* should be free. The change itself does a number of things: first it removes all the edges between *UserTask_2* and *ParallelGateway_6*, it then adds two gateway nodes, and finally it connects them all as shown in Figure 5.6.

As a result, *Task 3b* and *Task 3c* are now in an exclusive relation, where only one of the two can be executed. The result of this in the mobile application is shown in Figure 5.7, where they exclude eachother, and only one can be chosen.

If only *Task 3b* is in the execution trace, then this change can be applied with no problems. This change, however, excludes *Task 3c* from ever being executed, as *Task 3b* has blocked that option. If

both *Task 3b* and *Task 3c* are in the execution trace, then this change can not be applied. The new BPD would be state incompliant, as there is no way to run the execution trace on the new BPD with the exclusive gateway.

```
Event
    Calendar ExampleUser3
Condition
    Status Free ExampleUser3
Action
    RemoveEdge UserTask_2 UserTask_9
    RemoveEdge UserTask_4 ParallelGateway_6
    RemoveEdge UserTask_9 UserTask_4
    AddNode ExclusiveGateway_in ExGateway
        Description ExclusiveGateway_in
        Actors
    AddNode ExclusiveGateway_out ExGateway
        Description ExclusiveGateway_out
        Actors
    AddEdge UserTask_2 ExclusiveGateway_in
    AddEdge ExclusiveGateway_in UserTask_9
    AddEdge ExclusiveGateway_in UserTask_4
    AddEdge UserTask_9 ExclusiveGateway_out
    AddEdge UserTask_4 ExclusiveGateway_out
    AddEdge ExclusiveGateway_out ParallelGateway_6
```

Listing 5.4: EPL description of example 3: Exclusion of two Nodes

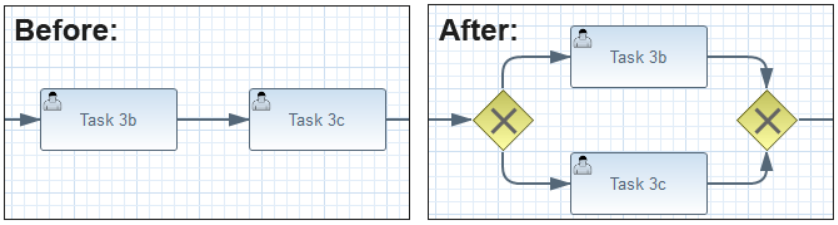

Figure 5.6: BPD change for example 3.

5.1.4 Adaptation Pattern (Swap)

This example describes the swapping of the two nodes labeled *Task 4b1* and *Task 4b2*. The change is triggered when a *Calendar* event is sent in by *ExampleUser4*, and the condition specifies that *ExampleUser4* should be busy. How the change is executed depends on the state of the BPD at the moment of application.

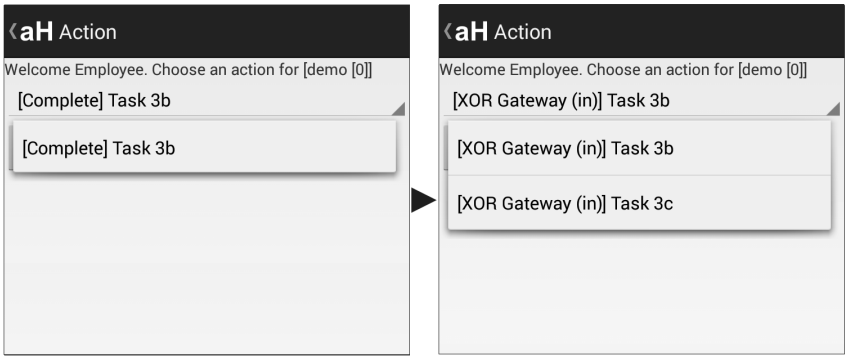

Figure 5.7: Application change for example 3.

```
Event
  Calendar ExampleUser4
Condition
  Status Busy ExampleUser4
Action
  Swap UserTask_11 UserTask_12
```

Listing 5.5: EPL description of example 4: Swapping two Nodes

Adaptation Patterns define semantic changes without explicitly stating which change primitives should be executed. These change primitives are therefore derived at runtime. For this example, to swap two nodes, the incoming and outgoing edges are found for each node, and they are swapped.

Listing 5.6 shows the resulting change primitives that are derived in this situation. We can see that the incoming and outgoing edges for *UserTask_11* and *UserTask_12* are removed, and that the new edges are added. It is worth noting is that the edge that connects *UserTask_11* and *UserTask_12* is not removed and added twice.

```
  RemoveEdge ParallelGateway_4 UserTask_11
  RemoveEdge UserTask_11 UserTask_12
  RemoveEdge UserTask_12 ParallelGateway_5
  AddEdge ParallelGateway_4 UserTask_12
  AddEdge UserTask_12 UserTask_11
  AddEdge UserTask_11 ParallelGateway_5
```

Listing 5.6: EPL description of example 4: Swapping two Nodes

As a result, the BPD is now changed as shown in Figure 5.8. Figure 5.9 shows how the possible options in the mobile application change as well.

In other situations, for example, when a node has been added in between *Task 4b1* and *Task 4b2*, or when the *Task 4b* node and the parallel gateways have been removed, another list of change primitives would have been derived, and subsequently, executed.

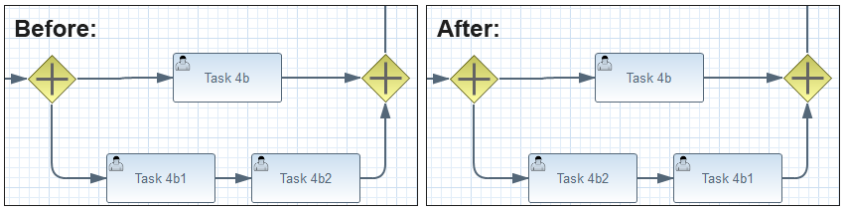

Figure 5.8: BPD change for example 4.

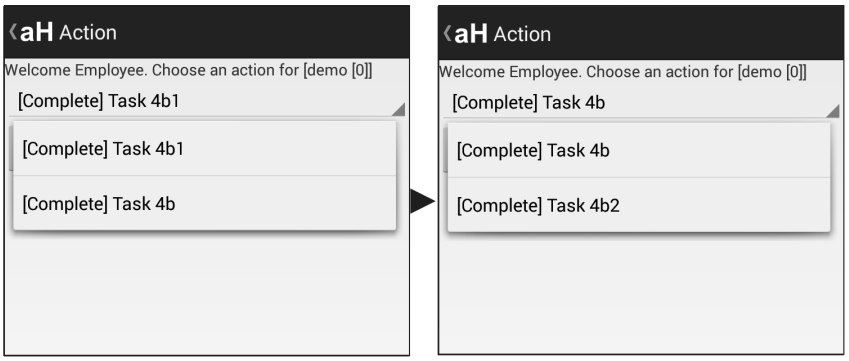

Figure 5.9: Application change for example 4.

5.2 Limitations

Since we built a simple proof of concept BPMS, this system has some limitations. We address these limitations here and argue how they do not pose any unknown problems when considering a full scale implementation.

Contextual Information

The Mobile Application supplies the server with two types of information: location and calendar status. This suffices for demonstration purposes as it allows us to make choices based on the contextual information sent from one or multiple users and one or multiple sources. In a full scale implementation, more information can be reported to the server through various sensors. This opens up more opportunities, as more information allows for more elaborate reasoning about decisions. However, we are confident that the core concepts of this proof of concept in gathering and processing contexual information are the same in a full scale implementation as the only real addition is the capacity to understand more context information types.

BPMN

This proof of concept uses a subset of BPMN. The entire BPMN specification consists of over 50 types of elements, and has numerous ways to connect and group them [30]. In our proof of concept we use only tasks, exclusive and parallel gateways, and start, end and regular events. We restrict the connection types to sequence flows. The theory behind changing BPMN graphs including more of these elements is discussed by Reichert et. al. [29]. The concepts of state compliance and dynamic change correctness defined there apply to our proof of concept implementation as well as a full implementation. Furthermore, any additional BPMN elements would still be subject to the four change primivites.

Adding BPMN elements poses little extra requirements above adding them to the language and modifying the process engine to recognize them. Some extra checks have to be implemented in the checking of soundness, state compliance and dynamic change correctness, but extra BPMN elements comply with these concepts. We therefore argue that a full scale implementation does not require more extensions in this area.

Adaptation Patterns

We implemented three adaptation patterns in our proof of concept: adding a process fragment. removing a process fragment and swapping two process fragments. However, adaptation patterns are just a sequence of change primitives derived from a process instance and variables. The derivation of these primitives is not always trivial, but is disconnected from the use of contextual information to enact changes.

Because our proof of concept supports all four change primitives, it is in fact a full implementation in this regard. Adaptation patterns are merely defined by the change primitives they generate, and can be arbitrarily defined. The set given by Reichert et. al. is an excellent minimum starting set, but by no means prevents anyone from constructing their own adaptation patterns [29].

Security

Our proof of concept implementation has no real security measures to prevent abuse. Implementation of a proper authentication system as well as secure connections would be desirable for a full scale implementation. However, these and other security measures do not interact with the application of ad-hoc changes, and therefore they do not pose a problem when implementing on full scale.

5.3 Conclusion

We validated our proof of concept through an example containing four different types of ad-hoc changes. This example demonstrated the capabilities of the proof of concept and validated the application of our proof of concept as a (small) BPMS that uses contextual information to drive ad-hoc changes. Furthermore we discussed the limitations of our proof of concept and how these would restrict a full scale implementation, and can concluded that these do not pose any fundamental problems.

Chapter 6

Conclusions

In this chapter, we answer the research questions posed in Chapter 1 one by one. We then use these answers to discuss the research objective. Finally we identify opportunities for further research based on this research.

6.1 Results

RQ**:** *What is the state of the art in the fields of Context-Awareness in mobile devices and Business Process Management, and how can they be combined?*

In Chapter 2 we gave an overview of context-awareness, specifically for mobile devices. We discussed complex event processing, a technique that is used to base decisions on streams of information, and we discussed business process management as a way to guide the business processes of a company. We concluded that contextual information can be used to supplement business process management in various way, by using context for simple variable inputs, by gathering contextual information to use in better analysis and by enabling changes to be made to the business process on-the-spot, or ad-hoc.

RO_1**:** *Design a software architecture for a BPMS that can use these benefits to apply ad-hoc changes.*

In Chapter 3 we designed a software architecture for a BPMS that can apply ad-hoc changes. We showed that this architecture can be defined as an extension of current BPMS and it is therefore fully backwards compatible. We further showed how ad-hoc changes should be structured in terms of change primitives and adaptation patterns, and how we can preserve soundness, state compliance and dynamic change correctness. We then showed how the capability to apply ad-hoc changes affects

each of the components, and discussed a few examples to illustrate how the concept of ad-hoc changes can be used to steer business processes.

We thus conclude that the designed architecture is a proper architecture for a BPMS that uses the benefits of context-awareness to apply ad-hoc changes to business processes.

RO_2**:** *Implement a Proof of Concept and validate the developed architecture.*

In Chapter 4 we developed a proof of concept to validate the architecture designed in Chapter 3. Through the development of requirements and use cases we designed a detailed software architecture for the newly developed components: the process engine and the stream processor. Furthermore a small device application was developed to accompany the process engine and stream processor. We furthermore developed a small language to describe ad-hoc changes in, that is used as input for the stream processor.

In Chapter 5 we validated the developed proof of concept through an example on the completed system and argued that the limitations of this proof of concept do not hamper the development of a full scale implementation.

The goal of this research was:

To identify the benefits of combining Context-Awareness in mobile devices with Business Process Management, and design an architecture that can use these benefits to apply ad-hoc changes to Business Processes.

We conclude that we achieved the goal of this research by designing an architecture for a BPMS that can use contextual information to apply ad-hoc changes, and then validating that architecture through the implementation of a proof of concept.

6.2 Future Work

In the course of this research, extra research opportunities have arisen that either transcended the scope of this research, or required to much time to be worked out and could not be explored within the time frame of this project. In this section we identify those research opportunities and discuss their possibilities.

Full BPMN Implementation

In our proof of concept we opted to implement only a subset of BPMN for the sake of brevity and clarity. We argued that a ad-hoc changes work with a full implementation as well, as they operate at the level of nodes and edges in a graph. Additional research could use the developed architecture, requirements and use cases to develop a full BPMN implementation to further validate the claims this research makes.

Extensive Adaptation Patterns

In our research we restricted ourselves to three simple adaptation patterns. We argued that adaptation patterns are descriptions of a high-level operation by which a sequence of change primitives is generated and executed. Further research can investigate more elaborate adaptation patterns, as well as efficient ways to describe adaptation patterns, and explore different options to generate change primitives from adaptation patterns.

Intelligent Decision Making

In our proof of concept we used an EPL designed according to the Event-Condition-Action paradigm. This simple form of change definitions allows us to make decisions based on the information and events. A more complex and elegant system can be developed to support decision making. This can even apply neural networks and artificial intelligence to use multiple sources of contextual information to reach a decision.

Another aspect could be to automatically derive possible changes based on previous executions and contextual information, instead of predefining them.

Advanced Business Analytics

As discussed in Section 3.2, even more detailed analyses are possible with process instances that started out the same but were differentiated through different applications of ad-hoc changes. The realization of such an analysis method would require some theory to compare graphs (BPDs) and find identical graph segments.

The Maximum Common Subgraph problem has been investigated, and algorithms have been defined to detect if a certain graph has a subgraph that is similar or isomorphic to another graph [34]. This type of research would need to be extended to allow shared subgraphs between two graphs to be found, with these shared graphs an analysis can be performed on the common subgraph of two or more BPDs.

Bibliography

[1] eMarketer, “Smartphone users worldwide will total 1.75 billion in 2014,” Jan. 2014.

[2] Nielsen, “How smartphones are changing consumers’ daily routines around the globe,” Feb. 2014.

[3] B. Schilit, N. Adams, and R. Want, “Context-aware computing applications,” in *Mobile Computing Systems and Applications, 1994. WMCSA 1994. First Workshop on*, pp. 85–90, IEEE, 1994.

[4] A. Buchmann and B. Koldehofe, “Complex event processing,” *it-Information Technology*, vol. 51, no. 5, pp. 241–242, 2009.

[5] G. M. Giaglis, “A taxonomy of business process modeling and information systems modeling techniques,” *International Journal of Flexible Manufacturing Systems*, vol. 13, no. 2, pp. 209–228, 2001.

[6] “A computer scientist’s introductory guide to business process management (bpm),”

[7] A. Schmidt, M. Beigl, and H.-W. Gellersen, “There is more to context than location,” *Computers & Graphics*, vol. 23, no. 6, pp. 893–901, 1999.

[8] J. Grudin, “Desituating action: digital representation of context,” *Human-Computer Interaction*, vol. 16, no. 2, pp. 269–286, 2001.

[9] G. D. Abowd, A. K. Dey, P. J. Brown, N. Davies, M. Smith, and P. Steggles, “Towards a better understanding of context and context-awareness,” in *Handheld and ubiquitous computing*, pp. 304–307, Springer, 1999.

[10] G. Chen, D. Kotz, *et al.*, “A survey of context-aware mobile computing research,” tech. rep., Technical Report TR2000-381, Dept. of Computer Science, Dartmouth College, 2000.

[11] M. Raento, A. Oulasvirta, R. Petit, and H. Toivonen, “Contextphone: A prototyping platform for context-aware mobile applications,” *Pervasive Computing, IEEE*, vol. 4, no. 2, pp. 51–59, 2005.

[12] S. von Watzdorf and F. Michahelles, “Accuracy of positioning data on smartphones,” in *Proceedings of the 3rd International Workshop on Location and the Web*, p. 2, ACM, 2010.

[13] A. Wood, J. A. Stankovic, G. Virone, L. Selavo, Z. He, Q. Cao, T. Doan, Y. Wu, L. Fang, and R. Stoleru, “Context-aware wireless sensor networks for assisted living and residential monitoring,” *Network, IEEE*, vol. 22, no. 4, pp. 26–33, 2008.

[14] P. Ruppel, F. Gschwandtner, C. K. Schindhelm, and C. Linnhoff-Popien, “Indoor navigation on distributed stationary display systems,” in *Computer Software and Applications Conference, 2009. COMPSAC’09. 33rd Annual IEEE International*, vol. 1, pp. 37–44, IEEE, 2009.

[15] S. Madden and M. J. Franklin, “Fjording the stream: An architecture for queries over streaming sensor data,” in *Data Engineering, 2002. Proceedings. 18th International Conference on*, pp. 555–566, IEEE, 2002.

[16] E. Wu, Y. Diao, and S. Rizvi, “High-performance complex event processing over streams,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pp. 407–418, ACM, 2006.

[17] D. Anicic, P. Fodor, S. Rudolph, and N. Stojanovic, “Ep-sparql: a unified language for event processing and stream reasoning,” in *Proceedings of the 20th international conference on World wide web*, pp. 635–644, ACM, 2011.

[18] A. Paschke, A. Kozlenkov, and H. Boley, “A homogeneous reaction rule language for complex event processing,” 2007.

[19] R. K. Ko, S. S. Lee, and E. W. Lee, “Business process management (bpm) standards: a survey,” *Business Process Management Journal*, vol. 15, no. 5, pp. 744–791, 2009.

[20] W. M. Van Der Aalst, “Business process management demystified: A tutorial on models, systems and standards for workflow management,” in *Lectures on concurrency and Petri nets*, pp. 1–65, Springer, 2004.

[21] S. A. White, “Introduction to bpmn,” *IBM Cooperation*, vol. 2, no. 0, p. 0, 2004.

[22] “More bpmn in the cloud,” June 2014.

[23] J. B. Hill, J. Sinur, D. Flint, and M. J. Melenovsky, “Gartner’s position on business process management,” *Gartner Research G*, vol. 136533, 2006.

[24] W. M. van der Aalst, “Tomtom for business process management (tomtom4bpm),” in *Advanced Information Systems Engineering*, pp. 2–5, Springer, 2009.

[25] M. Rosemann, J. Recker, and C. Flender, “Contextualisation of business processes,” *International Journal of Business Process Integration and Management*, vol. 3, no. 1, pp. 47–60, 2008.

[26] T. Kozel, “Bpmn mobilisation,” in *Proceedings of the European Conference of Systems: World Scientific and Engineering Academy and Society, WSEAS*, 2010.

[27] E. Santos, J. Pimentel, J. Castro, and A. Finkelstein, “On the dynamic configuration of business process models,” in *Enterprise, Business-Process and Information Systems Modeling*, pp. 331–346, Springer, 2012.

[28] “Gartner on bpms,” June 2014.

[29] M. Reichert and B. Weber, *Enabling flexibility in process-aware information systems: challenges, methods, technologies*. Springer, 2012.

[30] “Documents associated with business process model and notation (bpmn) version 2.0,” Jan. 2011.

[31] O. Oanea, *Verification of soundness and other properties of business processes*. PhD thesis, PhD thesis, Eindhoven University of Technology, 2007.

[32] “jbpm eclipse plugin,” June 2014.

[33] M. Fowler, *UML distilled: a brief guide to the standard object modeling language*. Addison-Wesley Professional, 2004.

[34] J. W. Raymond and P. Willett, “Maximum common subgraph isomorphism algorithms for the matching of chemical structures,” *Journal of computer-aided molecular design*, vol. 16, no. 7, pp. 521–533, 2002.