"Ambients makes Business Processes Mobile"

Mikkel Refsgaard Bech

Department of Computer Science
Aalborg University

February 10, 2004

Aalborg University
Department of Computer Science

Title: Ambients makes Business Processes Mobile

Author:
Mikkel Refsgaard Bech

Supervisor:
Bent Thomsen

Abstract
In a competitive world, business process outsourcing promises to reduce costs. Unfortunately business processes are not yet prepared to be moved around to take advantage of outsourcing. The stumbling point is the foundation that underlies the business processes. They do not support mobility. This thesis proposes a new foundation, mobile ambients, that enables the fine grained mobility that is needed to make outsourcing efficient. The problem of making business processes mobile is analyzed and a design based on mobile ambients is made that fulfills the requirements found. A working prototype that supports mobile business processes is implemented based on the design.

Aalborg University - Fredrik Bajers Vej 7E - DK-9220 Aalborg - Phone +45 96 35 80 80

Preface

This thesis is the result of my work with Business Processes and formally founded Mobility. The work has been ongoing between the August 1st 2003 and February 10th 2004.

Readers of this thesis is assumed to have basic knowledge of computer science. Knowledge of the topics workflow modeling and business process modeling is an advantage.

This thesis and the source code developed is available at `http://www.cs.auc.dk/~mrb/thesis/`.

I would like to thank my supervisor Bent Thomsen for his guidance, advices and encouragement. Also I want to thank Jimmy Juncker for reviewing the thesis and giving valuable feedback.

Aalborg, February 10, 2004

Mikkel Refsgaard Bech

Contents

1 Introduction **9**
- 1.1 Business Process Outsourcing . . . 9
- 1.2 Business Process Management . . . 10
- 1.3 The Future of Business Process Outsourcing . . . 11
- 1.4 Mobile Ambients . . . 12
- 1.5 Problem Statement . . . 12
- 1.6 Thesis Overview . . . 13

2 Mobile Ambients **15**
- 2.1 Implementations of Mobile Ambients . . . 17
- 2.2 Summary . . . 17

3 Business Process Management **19**
- 3.1 Dissecting Business Process Management . . . 19
- 3.2 Workflows . . . 22
- 3.3 Summary . . . 23

4 Workflow Patterns **25**
- 4.1 Description of the Workflow Patterns . . . 25
- 4.2 Summary . . . 31

5 Analysis **33**
- 5.1 Mobile Ambients . . . 33
- 5.2 Supporting the Workflow Patterns . . . 34
- 5.3 Summary . . . 39

6 Design **41**
- 6.1 The Basic System . . . 41
- 6.2 Meeting the Requirements . . . 47
- 6.3 Results of the Design . . . 50
- 6.4 Summary . . . 50

7 Implementation **53**
- 7.1 Software to Build Upon . . . 53
- 7.2 The Movement Requirement . . . 55
- 7.3 The Complete State Requirement . . . 57
- 7.4 The Execution Environment Requirement . . . 57
- 7.5 Results of the Implementation . . . 58
- 7.6 Summary . . . 58

8 Conclusion **59**
8.1 Future Works 60

Bibliography **60**

Chapter 1

Introduction

The future of business is changing. Globalization increases trade and communication which results in stronger competition. This increased competition causes a focus on price, which in turn drives focus on cutting costs. Therefore businesses look to find ways to bring down costs and one way is outsourcing of its business processes; Business Process Outsourcing (BPO).

1.1 Business Process Outsourcing

Gartner defines BPO as: *delegation of one or more IT-intensive business processes to an external provider that, in turn, owns, administrates and manages the selected process(es) based on defined and measurable performance metrics.* [4]

This definition requires some explanation, which I will give here:

To understand BPO, you have to understand what a business process (BP) is. They are processes that happen repeatedly, are supported by IT, but need human interaction. The characteristic business process targeted for outsourcing is, IT-intensive, administrative and is not a core process to the business. Examples are finance and accounting, human resource management, supply chain integration and costumer relationship services like help desks. These processes can be outsourced, this is BPO.

BPO reduces cost because the external provider has a mass production advantage. Outsourcing can help a business to focus on its core competence because it releases in-house resources and it gives the possibility to buy knowledge that does not exist in-house. Both the buyer of the service and the provider of the service are being more efficient, as they both do what they are best at.

There is more to BPO than just outsourcing of business processes. Outsourcing of a business process is not just outsourcing of the actual work *within* the process, it is also outsourcing of the work *with* the process. This means analyzing,

measuring and evolving the process to make it more efficient to execute and thus cheaper to run.

BPO is not a trend that will go away anytime soon, according to internet.com, BPO is a market in growth: *In 2003, the worldwide business process outsourcing (BPO) market is expected to grow 10.5 percent - to $122 billion - up from $110 billion in 2002, according to Gartner Inc., and the Aberdeen Group predicts 13 percent annual growth until 2005 when the market will reach $248 billion.* [1]

Outsourcing a business process requires that you have detailed knowledge of the process. Getting detailed knowledge about business processes is part of doing *business process management* (BPM).

1.2 Business Process Management

BPM is about breaking down a business in to manageable chunks called business processes, that defines: *what the business does*, *who does it* and *with what it is done*. Historically this has been done in both natural language and in ad-hoc workflow languages. Today, BP's are defined in languages that are, more or less, founded in formalisms as the pi-calculus[19] and petri-nets[20]. Examples of these languages are BPML[6] or XLANG[7]. The formalisms provide a firm foundation for workflow languages, and the workflow languages are becoming the de facto tool for describing BP's.

BP's are run on a business process management system (BPMS). A BPMS is a program that interpret BP's and execute them. It also provides interfaces to humans and interface with other software. BPMS's also support modeling and management of BP's.

Having defined the business processes, by using BPM, enables outsourcing of processes that previously were impossible to outsource because the process was not fully understood or known because of its complexity or deep integration in the organization. BPM is a prerequisite for BPO.

The current trend in BPM is to make BP's available as a *web service*. A web service is a BP which interface is made available on the Internet for others to use. This can be regarded as outsourcing, but it happens either by using someone else's business process or by moving your entire BP to an external provider, that runs them for you.

While this all-or-nothing approach is fine for many purposes its not as fine grained as it could be. Maybe you only want a part of your BP run by an external provider because you want to protect your intellectual property as a business process can be (it could be patented), or you want to keep control of parts of it because you do it better yourself or it is a particular sensitive activity for your competences.

1.3 The Future of Business Process Outsourcing

BPO can change the business of the future. Next step in this trend is a generation of businesses that provide only one or a few services. This business is then a part of a patchwork of services, that is used by others or again part of a new service. This new generation of businesses could be characterized as *vertical* businesses, because they build on top of each other and complement each other, as opposed to todays *horizontal* businesses, which offer a wide range of services. Competition will not disappear, because the businesses that have similar core competences will compete even harder, because it is their only competence.

In the ideal environment for BP's, BP's are fine grained entities that float freely around among service providers that can execute the BP at different service levels, which could be parameters as reliability, speed or price. The market will be driven by the market forces on a day-to-day basis (or even finer grained). The market will be a virtual stock market where services are sold and bought at prices influenced by demand. At the same time it will be highly automated as BP's and providers agree on a Service Level Agreements (SLA) so that requirements are exactly matched and prices is automatically negotiated. Also smaller services are automatically composed to match a certain needed service, that way partnerships between service providers can be established fast, automatically and even on short terms. BPO could enable this future.

To reach this future, some challenges must be overcome, e.g. the description and negotiation of SLA's, the description of services and automatic composition of services. I will not look in to these problems. Another problem is mobility. Business processes must be able to move from one system to another, e.g. from the service buyer to the service provider. To use BPO efficiently, moving business processes must happen efficiently and unobstructed.

Unfortunately current business process management systems (BPMS) does not support moving business processes. The reason for this is found in the very foundation of the business process; its workflow.

The workflows are described in a workflow language. These are typically based on either pi-calculus or petri-nets neither of which has any notion of mobile entities or places. They can model this, as they are Turing complete, but they does not intuitively support mobility.

Therefore are the current BPMS's implemented as centralized systems. Although some BPMS'es support running on clusters it does not achieve whats needed; mobile business processes. Some other foundation is needed to intuitively support mobile BP's. Candidates for this are the CCS[18] extension CHOCS[22] or π-calculus derivative "Higher Order π-calculus[21]. The problem with these calculi is that they simulate mobility by sending the process description, not a live process. Mobile Ambients[15] is a different calculi, that moves the process while is alive.

1.4 Mobile Ambients

Mobile ambients[15] is a calculus of mobile processes made by Luca Cardelli and Andrew D. Gordon. An ambient is a place where computations take place. Ambients are bounded and can contain other ambients, this structure forms a tree. They navigate the tree by moving in and out of each other. When ambients move, their subambients move along as they are within the bounds of the moving ambient. Their navigation in the tree can represent mobility.

I want to use ideas from mobile ambients to make business processes mobile, therefore I have chosen to let mobile ambients inspire and influence the foundation of my system for mobility.

1.5 Problem Statement

The main goal of this thesis is to demonstrate that mobile ambients can provide a solid foundation for mobile business processes.

By solid I mean that mobile ambients should support mobility for all kinds of business processes.

My approach is in three steps:

First, I will analyze the requirements to build such a system based on mobile ambients. To make the analysis exhaustive I will use workflow patterns that represents the common workflows business processes use.

Second, based on the requirements I will design a system, where BP's are contained in mobile ambients. The design will make sure that BP's work as expected even when they are mobile, hence that the system is solid.

Third, I will implement a limited prototype of the design to demonstrate that it is realizable.

Although it is important to show that a working system is attainable, there are other factors that influences its success. I have called these factors subgoals and provide a number of them here (in no particular order):

Transparent to BP's. Developing and maintaining good BP's requires a considerable effort, therefore scrapping BP's is hardly acceptable and if changing them is unavoidable, change should be minimal.

Complementing current BPMS's. The system should not require an entire rewrite of the BPMS to run.

Usable. The new features added by mobility should be easy to use and understand for those who develop and maintain BP's.

Understandable. Going from a centralized system to a distributed system requires a change of the perception of the system. It should be possible

to explain how the system works in simple terms.

Deployable. Going from a centralized system to a distributed system is something that requires a careful roll out, therefore the system should support this.

Efficient. The system should gain efficiency and scalability over a centralized system.

Reliable. Information in the system should be accessible anytime.

Secure. It should be possible to keep information confidential.

These subgoals are not necessarily required to be fulfilled for the system to be successful, but they are useful to keep in mind while working toward the main goal.

I have chosen to call the system AmbProMo, freely derived from the sentence: Ambients makes Business Processes Mobile. AmbProMo means that ambients are promoted (as in advertising) by me for this use. It also means that ambients are promoted (as in rank) by me, to be used in a relevant application. And, it also means that ambients are used *pro* (for) mobility.

1.6 Thesis Overview

The following three chapters, *Mobile Ambients*, *Business Process Management* and *Workflow Patterns* provides necessary background information for my work. The *Analysis* chapter contains the requirements for the system, which the *Design* chapter answers. My implementation of the design is in the *Implementation* chapter. The chapter *Conclusion* concludes on my work and presents topics for future works.

Chapter 2

Mobile Ambients

Mobile ambients is a calculi of mobile processes made by Luca Cardelli and Andrew D. Gordon presented in "Mobile Ambients"[15]. The calculi was made to model mobility, both logical, like processes in a machine, and physical, like processes on a network and on mobile devices. Like many process calculi it is reducible[15] to Pi-calculus.

An ambient is a bounded place where computation happens. Bounded means that processes in ambients can not communicate outside the ambient. To make processes communicate, they must be in the same ambient. To achieve this, ambients can enter each other and dissolve so that processes are brought together.

The fact that ambient can be inside each other, makes nested ambients look like a tree, structure wise. This is illustrated in Figure 2.1.

In the ambient calculus names are denoted by small letters and processes by capitals. The following is an ambient by the name "n" that contains a process "P":

$n[P]$

The configuration of the ambients in Figure 2.1 on calculus form would be:

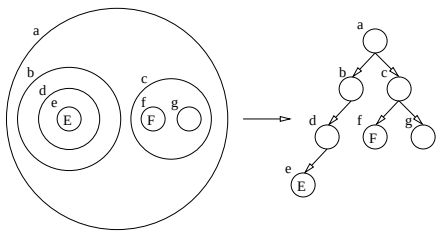

Figure 2.1: A number of ambients represented as being within each other and as a tree.

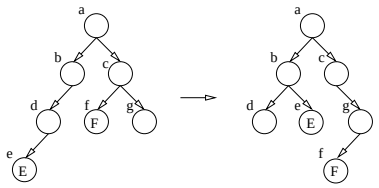

Figure 2.2: Illustration of results of the commands IN and OUT.

$a[b[d[e[E]]]|c[f[F]|g[]]]$

Entering an ambient is done by a command called IN. Ambients can also exit each other by a command OUT and dissolve by a command OPEN. The IN and OUT commands are illustrated in Figure 2.2. In the figure, the process E make the ambient E go out of D and the process F make the ambient F go into G. The commands IN and OUT are good commands for modeling mobility. They simply mean moving down or up the tree, respectively.

The reduction rule for IN are:

$n[in\ m.P|Q]|m[R] \rightarrow m[n[P|Q]|R]$

The reduction above means that in the ambient "n" there is a process "P", that commands the "n" ambient to move into the ambient "m".

The following reduction for OPEN does the opposite; ambient "n" is told to move out of "m", by the process "P".

$m[n[out\ m.P|Q]|R] \rightarrow n[P|Q]|m[R]$

The command OPEN is different, it tells an ambient to dissolve. In the following the process "P" tells the ambient "m" to open:

$open\ m.P|m[Q] \rightarrow P|Q$

As an example the reduction of the configuration in Figure 2.2 is given here:

$a[b[d[e[out\ d.E]]]|c[f[in\ g.F]|g[]]] \rightarrow a[b[d[]|e[E]]|c[g[f[F]]]]$

As ambients are nested inside each other, moving an ambient means moving the entire branch of ambients. Therefore the branches of the tree can be thought of as domains. This feature is good for mobility because it delegates control clearly and movement is 'inclusive', meaning that whatever is included in an ambient, follows that ambient when it moves.

Ambients must dissolve for processes to communicate. Castagna et. al. found this to be counterintuitive as communication between processes in distributed systems had to be modeled by using temporary ambients as carriers for messages and therefore introduced *boxed ambients*.

In Boxed Ambients[13] communication up and down the tree, between parents and children, is allowed. This makes it easier to model communication between

processes. This new way of communicating meant that the "open" command could be dropped. It was found to be a security concern in distributed applications.

I will also not use this command in AmbProMo, for the reasons given above and elaborated on in the analysis. I think that allowing communication over the bounds means that the "boundedness" of ambients is somewhat diluted, but it makes sense that the bounds are open for communication.

Although I will not use boxed ambients the calculus inspired me to change the communication in AmbProMo, instead of using plain mobile ambients.

Another calculus related to ambients is the *Calculus of Mobile Resources*[17] by Godskesen et. al. This calculus was made to model mobile capacity constrained resources. It introduces "slots" as containers of resources and processes. Slots can be nested like ambients. An interesting feature of this calculus is that both communication and movement can be done through multiple levels of the tree. This feature has inspired the way mobility works in AmbProMo.

2.1 Implementations of Mobile Ambients

Cardelli implemented[14] a centralized version of Mobile Ambients. As this implementation locks objects on remote machines it is not efficient. Fournet et. al. implemented and proved the correctness[16] of a protocol for mobile ambients in a distributed environment. The protocol is message based and requires only local synchronization which means that its efficient in a distributed environment.

The protocol is in three steps. In the first step, the "delegate" step, the ambient that wants to move notifies its parent. In the second step, "relocate", the parent ambient gives the address of the moving ambients new parent. The third step, "register", the moving ambient notifies its new parent about itself. I will use this protocol in my implementation.

2.2 Summary

Ambients form a tree structure because they can be nested inside each other. Ambient can move in and out of each other, and when they do, their subambients follow them. In this way processes can move from a place to another modeling mobility.

There exist a protocol for efficient use of ambients in distributed applications which I will use in the implementation of AmbProMo.

The alternative calculi for mobility, boxed ambients and calculus of mobile resources, has inspired the work on AmbProMo.

Chapter 3

Business Process Management

There are two primes uses for BPM, first, it helps understanding business, second, by understanding business, you can manage and change it. In [24] van der Aalst et. al. defines BPM as: *Supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information.* In the following chapter an overview of how BPM helps understanding, managing and changing business will be given.

Often software is made to fit the business as the business looks at the point in time where the software is developed. Therefore when the business environment change, the business is not able to cope with the change, because its software is not made to change with the business needs. Therefore the software becomes a hindrance for change instead of a supporting system for the business.

BPM is an attempt to make software for businesses as dynamic as the business itself. By building a generic software that can support business processes and by handling business processes as yet another type of data, the business processes can be examined and evaluated and evolved. Lets step further into what BPM is.

3.1 Dissecting Business Process Management

Dissecting 'Business Process Management' gives 'business processes' and 'management'.

A business can be viewed and modeled as a number of interacting parallel processes. These processes can be purely internal to the organization, but many also interacts with external processes. These are the business processes. These processes make the business what it is, differentiating it from the competitors

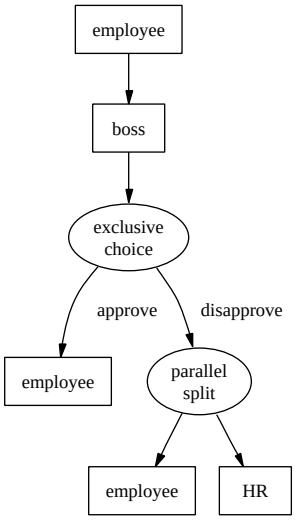

Figure 3.1: A business process to request a holiday.

and providing value to the business customers. An example of a business process could be an order from a costumer, an inquiry to the stock or simply an order from one person to another to do something. A process could also involve persons interacting with software systems or software systems communicating with each other.

A concrete example of a model of a BP is shown in Figure 3.1. This BP models the process of requesting a holiday. It works this way: An employee that wants a holiday, starts this process and enters when he want his holiday. This request is then presented to his boss, whom then makes the decision whether to allow the holiday or not. If he approves, the employee and human resource department are notified, otherwise only the employee is notified. This example will be used throughout the thesis.

Looking up 'Management' in a dictionary gives: "the conducting or supervising of something". Mapping this to management of business processes would give words like enacting and examining. But in BPM, management also means evaluating and evolving business processes. A method called 'Organizational Process Modeling' for doing BPM is described in the book of Warboys et. al. [25]. This book describes the main ideas of BPM:

1. Making the specification of business processes more business specific.
2. Integrating existing software systems.
3. Making specifications of business processes active.
4. Supporting reevaluation of the business processes.

We will look at these ideas in turn in the following.

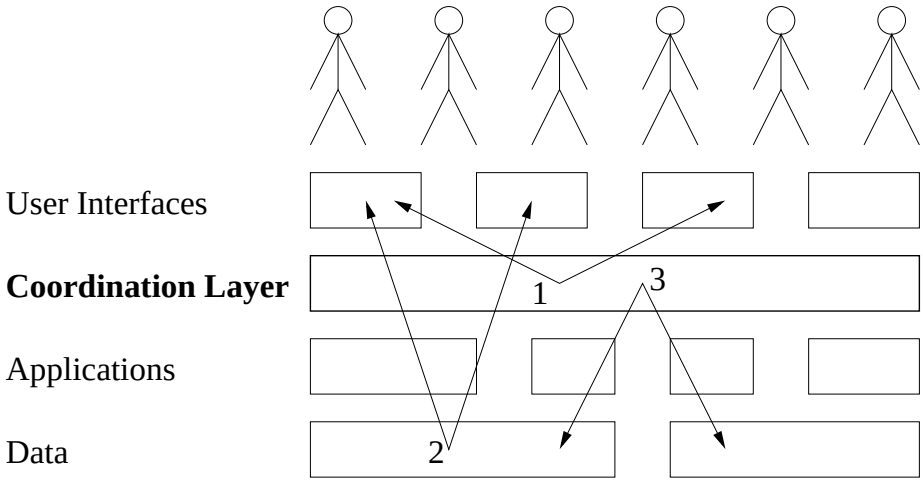

Figure 3.2: The coordination layer is introduced by BPM in the software model [25, p. 77].

3.1.1 Making the Specification more Business Specific

Instead of describing the business from the implementation perspective, it is described from the business perspective. E.g. instead of modeling data, as ER-diagrams, or modeling workflow through the software system, the business processes are modeled as a process of the business. The business is the 'machine' on which the business processes are executed.

Business process management works with terms that are familiar to business language. This is because BPM is supposed to support business persons doing modeling, as opposed to other modeling techniques where the model is a tool for communicating between business individuals and system implementors.

3.1.2 Integrating Existing Software Systems

Business Process Management (BPM) is the introduction of a new layer in the traditional software model. This new layer concerns the coordination of the business' resources and is therefore called the coordination layer. As shown on Figure 3.2 from [25, p. 77], the layer is placed between the user interface level and the application level. Its purpose is to coordinate the flow of business processes throughout the business between people and people (1), between people and tools (2) and between tools and tools (3).

Adding another software system to a business already having lots of systems is not desirable. The idea is not to introduce another new application in the business. Rather it is to make the existing applications work together and integrate the already existing resources with each other. Since the coordination layer spans all applications in the business, it has access to all data. It can bridge the

gap between incompatible applications and synchronize data between different databases. It can provide a uniform user interface for the people interacting with the system. Of course there has to be some software on the coordination layer. But by design, the coordination layer does not manipulate data as applications would.

3.1.3 Making Specifications Active

Modeling is often used in software development. When modeling a business many modeling techniques exist such as ER-modeling, Dataflow-modeling, Workflow-modeling etc. They all model a part of the business in some defined way. Because they describe something, they are called descriptive models. The models that are the outcome of the modeling by these techniques, are used in the development of the actual software. They serve as a basis for the structure of the software. If the software needs changing later, ideally one would take a look at the model, change the model, then change the software to reflect the new model.

BPM does more than modeling. When a model is made, the model is also the specification for the software, that is - the model is executable. The model is executed, when the business process that the model reflect is started. There is no need to keep the model and the business process synchronized, they are the same. These models are called active models, because they actually control the process that they describe.

3.1.4 Supporting Reevaluation

To support reevaluation of business processes, a special process can be made. This process uses the other business processes as its data and does evaluation of these processes. This process becomes a meta process for the business processes. In this way the design of a business process becomes a continuous process as the development of the business as a whole is, instead of being a one time event.

3.2 Workflows

Workflows are a method to design a process and its interactions. The following definitions are based on the definitions in [23]. A *workflow* is a specification of activities and their order. An *activity* is an atomic piece of work. Activities are connected through *transitions*. The activities are undertaken by *roles*. Roles are to be understood broadly. It could be the function of a person or a group of persons. It could also be the specific activity of a software system serving the business. A *case* is a specific instance of a workflow. A case can be executed concurrently which means that is has *threads*.

Many workflow languages exists, some based on formal foundations. Also many implementations of workflow engines exist. Aalst et. al. investigates languages

and implementations of workflow engines. They have collected a set of common workflows that can model the usual situations in workflows, these situations are called workflow patterns.

3.3 Summary

The BPM method makes the modeling of business processes approachable by business people by using business terms to describe business processes. Therefore is it easier for business people to understand their business processes. By using existing software systems and integrating them instead of introducing another, redundant work and data is avoided and the software systems become more manageable. The direct approach of executing the business models in a software system makes it possible to find flaws in the models and makes changes fast to deploy. This makes the business manageable.

Chapter 4

Workflow Patterns

In an article called "Workflow Patterns"[23], Aalst et. al. have collected, a set of common workflows that can model the usual situations in workflows, these situations are called workflow patterns (WfP's). The work by Aalst etc. is highly regarded and spoken off as the 'design patterns' of workflows, referring to the famous design patterns identified in object oriented programming.

I have chosen to use these workflow patterns as a kind of requirement specification for what my system must support to be complete, workflow wise. These descriptions will be useful to analyze and understand whats required to build a BPMS with mobile BP's.

In this chapter I will describe the WfP's and explain how they work. The description, examples and figures are my own, but are inspired by the "Workflow Patterns" article and the workflow patterns homepage[3].

4.1 Description of the Workflow Patterns

Here follows the descriptions of the individual workflow patterns.

4.1.1 Sequence

When an activity is complete, the next is started. Activities are chained together by transitions. This is shown in Figure 4.1 where activity B follows activity A and activity C follows activity B.

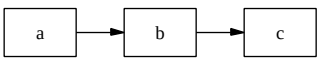

Figure 4.1: Workflow Pattern: Sequence

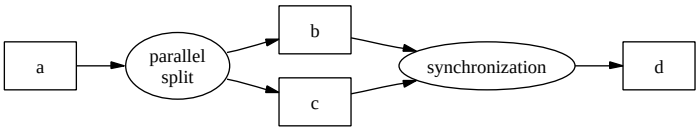

Figure 4.2: Workflow Patterns: Parallel Split and Synchronization.

4.1.2 Parallel Split

When an activity is complete, two or more activities are started and execute in parallel. This is shown in Figure 4.2 where activity B and C are started after A has completed.

Example of a use of this pattern: When an order has arrived, simultaneously package the goods and print a delivery note.

4.1.3 Synchronization

After a split, this activity waits for both (all) activities to complete, before continuing. This is shown in Figure 4.2 where activity C is not started before both A and B has completed.

Example use: When goods has been packaged and delivery note has been printed, ship the package.

4.1.4 Exclusive Choice

This pattern gives exclusive choice functionality like a switch-statement of an imperative programming language. Figure 4.3 shows two routes the BP can take, either B or C.

Example use: When a request for holiday comes in, either approve or disapprove it.

4.1.5 Simple Merge

After an exclusive choice the paths of execution are merged and execution continues. There is no need for synchronization as only one of the paths will actually be used. Figure 4.3 shows a simple merge where activity C is executed when only one of A or B has completed.

Example use: Regardless of the response to the request for holiday, notify the requester.

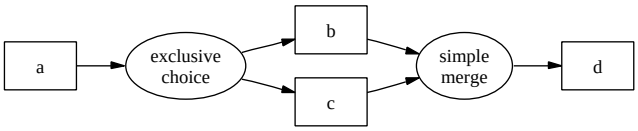

Figure 4.3: Workflow Patterns: Exclusive Choice and Simple Merge.

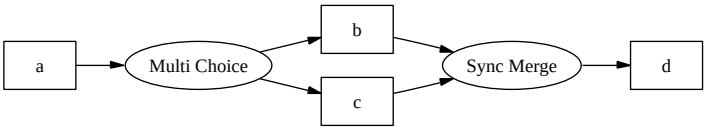

Figure 4.4: Workflow Pattern: Synchronizing Merge

4.1.6 Multiple Choice

Similar to parallel split this pattern can split into one or more parts which execute in parallel. Unlike parallel split this pattern does not necessarily have to take all of the paths, it can choose one or more or all of them. Figure 4.4 shows two paths, B and C, where either B, C or B and C can be taken.

4.1.7 Synchronizing Merge

As parallel split has synchronization, multiple choice has synchronizing merge. This pattern merges the threads the multiple choice makes. Implicitly it knows how many threads it must merge. Figure 4.4 shows synchronizing merge after a multiple choice.

4.1.8 Multiple Merge

After a parallel split a multiple merge can be convenient, if both threads have to finish with the same activities. There is no synchronization as the threads just run the same activities, they do not communicate. Figure 4.5 shows a case where the two threads, that execute B and C respectively, both will run D thereafter, i.e. D is executed twice.

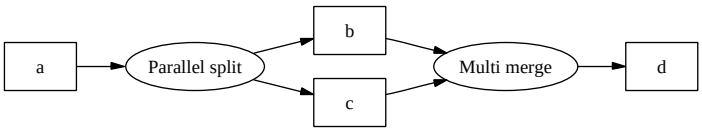

Figure 4.5: Workflow Pattern: Multiple Merge

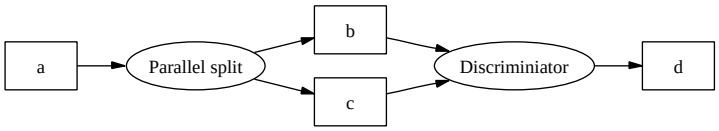

Figure 4.6: Workflow Pattern: Discriminator

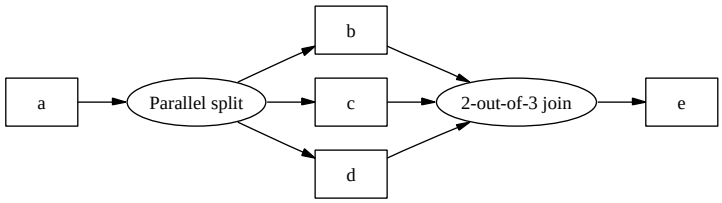

Figure 4.7: Workflow Pattern: N-out-of-M Join

4.1.9 Discriminator

A different kind of merge can be done with the discriminator. In this merge the following activity is activated only once and that is when the first thread reaches the merge. The remaining threads are ignored. Figure 4.6 shows a case where activity D is activated once, when either B or C is done.

4.1.10 N-out-of-M Join

The pattern n-out-of-m join is a variation of the discriminator pattern, as this pattern requires n activations before the following activity is activated. Figure 4.7 shows a 2-out-of-3 join where two of the activities B, C and D must be done before E is activated.

4.1.11 Arbitrary Cycles

This pattern supports the case where one or more activities are repeated arbitrarily. Figure 4.8 shows a workflow where activities B and C and activities C and D can be repeated arbitrarily.

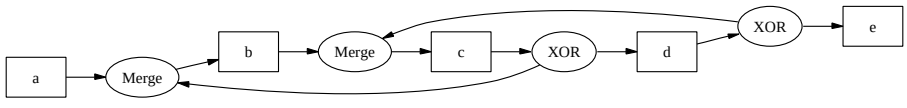

Figure 4.8: Workflow Pattern: Arbitrary Cycles

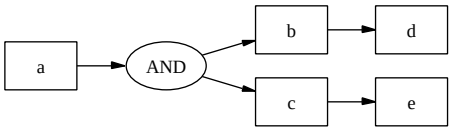

Figure 4.9: Workflow Pattern: Implicit Termination

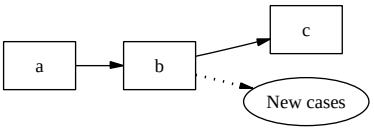

Figure 4.10: Workflow Pattern: MI Without Synchronization

4.1.12 Implicit Termination

This pattern simply states that when there are no more activities, the workflow is terminated. Figure 4.9 shows a workflow where there is a parallel split. This workflow should not terminate before both threads have terminated, i.e. both activity D and E are complete. This does not mean that the individual threads can not terminate, just that the BP exists as long as there are threads in it.

4.1.13 Multiple Instances Without Synchronization

Multiple instances are when a case can start new cases, and therefore a case consist of multiple instances of cases. This pattern allows the creation of a number of cases independent of the current case. Figure 4.10 shows a workflow wherein activity B creates a number of cases.

4.1.14 Multiple Instances With Synchronization

This patterns describes a BP wherein a BP is repeated a number of times and the parent BP must wait for it to terminate.

4.1.15 Deferred Choice

This pattern is a mix of exclusive choice and parallel split. It models the situation where two (or more) roles can make a choice but only one has to make it.

Example use: Only one of two persons has to approve a travel expense, both are offered the opportunity to approve it, until one of them actually approves it, then the other does not have to bothered and the opportunity is withdrawn for him.

Figure 4.11: Workflow Pattern: Deferred Choice

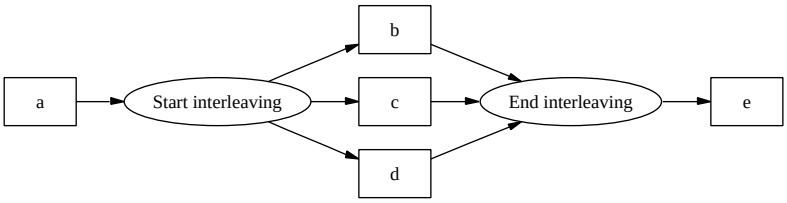

Figure 4.12: Workflow Pattern: Interleaved Parallel Routing

4.1.16 Interleaved Parallel Routing

This pattern makes it possible to have a set of activities, which all must be executed once, but not at the same time, executed in any order. Figure 4.12 shows a case where B, C and D can be executed in any order.

4.1.17 Milestone

The milestone pattern makes it possible to have an activity check on the state of the case. Figure 4.13 shows a case where activity D checks if the case has reached its milestone M, between activity B and C.

The thread that checks if the milestone has been reached is called the measur*ing* thread, the other is called the measur*ed* thread.

4.1.18 Cancel Activity

In this pattern a single activity can be canceled, which means that the thread running the case is terminated. Figure 4.14 shows a case where activity B can

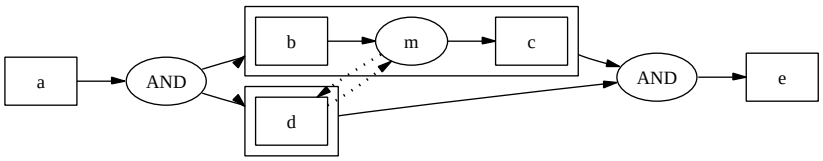

Figure 4.13: Workflow Pattern: Milestone

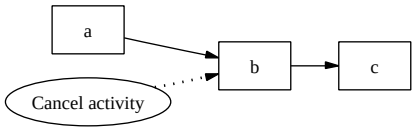

Figure 4.14: Workflow Pattern: Cancel Activity

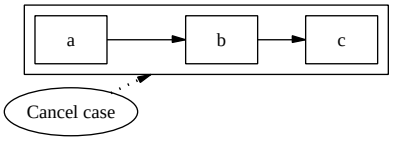

Figure 4.15: Workflow Pattern: Cancel Case

be canceled.

4.1.19 Cancel Case

This pattern enables gives the possibility of canceling an entire BP. This means that child BP of the BP are canceled. Figure 4.15 shows a case that can be canceled.

4.2 Summary

The workflow patterns are fundamental building blocks of workflows that spans the most usual situations that workflows has to model. These descriptions will be used to analyze the requirements to build a BPMS with mobile BP's.

Chapter 5

Analysis

In this chapter I will analyze the requirements to build AmbProMo. The requirement basically comes from two sources, namely the foundation on which I build the system, mobile ambients, and the application that will run on the system, represented by WfP's.

First I will take a look into the features and limitations of mobile ambients. Then I will consider each WfP and whats required to make it work when in an environment where BP's are mobile.

The requirements found in the analysis will be considered and answered in the design.

5.1 Mobile Ambients

As any model, the foundation which I have chosen, brings some features and some limitations to AmbProMo. I will look into those in the following.

5.1.1 Features of Ambients

Mobile ambients gives some features that are nice in a distributed system. First of all, mobile ambients gives a platform for mobile processes. Processes can, encapsulated in ambients, move in and out of other ambients, thereby modeling mobility. This was the reason for choosing ambients as the foundation for AmbProMo.

Second, the fact that the ambients encapsulates the process(es) is convenient to secure and package the process. Access to the processes are under some control and it is well defined what is inside the ambient and what is not.

5.1.2 Limitations of Ambients

Mobile ambients also have some limitations that are a by design. I provide workarounds for some of the limitations that I find too restricting.

The fundamental structure of ambients, the tree, comes from the fact that ambients are nested inside each other. Having such a structure has its advantages and its disadvantages. The advantage is that a structure gives some kind of order that can be relied on and the disadvantage is that this requires that some rules are obeyed. E.g. in a tree you can rely on the facts that there are only one root, each node has max one parent, there are no cycles etc. Some of the rules that must be obeyed are that siblings must not be connected, a node must be connected to the tree etc. It is a limiting structure because it requires more moves to navigate the tree, than it would have, if the structure had been a fully connected graph, e.g. like the Internet is. This structure is accepted in my system, when it comes to movement of the ambients.

Communication is also limited. Processes can only communicate when they are in the same ambient. To my use, this is too limiting, therefore is this limit relaxed as it will be apparent in the design chapter.

5.1.3 Undesired Features

One thing that mobile ambients have that is not desired in my system is the `open` capability. The `open` capability dissolves an ambient into its subambients. This means that when a process in an ambient wants to talk to another process in another ambient, one of the ambients must enter the other and `open` thereby dissolving itself. Since it is not rare that processes need to talk to each other, this becomes an issue, because dissolving ambients, and thereby business processes, while communicating, is counterintuitive in a business process management system. Business processes does not dissolve themselves, although they terminate at some point and are removed.

5.2 Supporting the Workflow Patterns

Workflow languages were made to describe the order in which some activities are executed. Therefore it is crucial to examine how the workflows can be made mobile and which workarounds there are needed to make sure that the workflows work as in a centralized environment. To do this I will take van der Aalst's "Workflow Patterns" as a starting point. These workflow patterns are regarded the basic building blocks of workflows and therefore serve as a testing ground for the design of workflow languages and thus they must be supported in some way by the underlying system that executes these workflows.

For each workflow pattern I will explain what the problems are of using it in a mobile environment (if any). I will name each of these requirements to be able to refer to them during the rest of the report. How to overcome these problems

will be discussed in the design chapter.

This section will therefore be used to determine the requirements that my system must meet to be successful workflow wise.

5.2.1 Sequence

Since each activity of a sequence can happen on different machines there must be some way to move a BP from one computer to another. This means that the BP must be packaged and shipped to another machine. This requirement will be called *the movement requirement*.

The BP must be moved in its entirety, that means the state of the workflow and the data in the BP. This requirement will be called *the complete state requirement*.

Since the BP can be executed on the machines it visits, there must be an environment on each machine that can execute the BP. This requirement will be called *the execution environment requirement*.

These are major requirements for the system because they require much functionality to work, but they are also a kind of baseline requirements because they are quite obvious. In a distributed system something must be able to move between the machines for it to be called a distributed system, also the machines in the system must be able to do work, otherwise it would just be a distributed storage. So the above requirements where not unexpected and must off course be met by AmbProMo. The above requirements are together called *the base requirements*.

5.2.2 Parallel Split

To support this WfP the BP must be able to make copies of a BP. The BP is copied for each branch in the split. This requirement will be called *the copyable requirement*. Copying the BP is not a problem, but synchronizing is. This situation is considered in the next WfP, synchronization.

After the copy, there are a number of instances of BP with each their workflow. In this workflow the next transition must be enabled, so that each one of the BP goes to one of the next activities, the one corresponding to the enabled transition. This requirement will be called *the manipulatable requirement*.

5.2.3 Synchronization

After a split of a BP, like after a parallel split, there are two or more copies of the BP, which may contain different data. When these copies of the BP meet again and synchronize their states must also be synchronized. The state of the workflow is already the same, as they meet in the same activity, thus this

does not need any treatment. But the data that they carry also needs to be synchronized.

In a centralized system all data and all processes are in one place and all communication happen internally and cannot fail. So there is no concern whether something is available or accessible, either everything runs, or it does not. In a distributed system, single machines can fail, multiple machines can fail, machines can be transient unreachable, networks can split and it is difficult (if not impossible) to determine the state of the entire system. Also clocks in a distributed system can not be synchronized.

Several strategies can be used to synchronize the data:

- Updated data can be flagged, so that flagged data overwrites non-flagged data. But in case the same data is flagged more than once, this scheme does not always work.
- Last updated data can overwrite the older data. Since clocks may not be fully synchronized, this scheme fails in some cases.
- Data can be marked as read-only and writable for one of the threads. This scheme only works, if the sharing of the data can be agreed on, before the BP is copied.
- Each thread can have a priority, that determines which data is overwritten. The priorities must then be assigned somehow.
- The synchronization of the data can be done manually.
- The synchronization could be done by communicating between the copies of the BP.

Which of the above schemes are chosen depends on a factors as how likely is it that the data is updated? what are the different threads doing? Therefore, the choice should be made on a case by case basis and therefore it is wise to support as many of these schemes as possible. This requirement will be called *the data synchronizing requirement*.

5.2.4 Exclusive Choice

This WfP does not add any requirements, but falls back on *the base requirements*, as the BP only needs to moved in the correct direction determined by the BP.

5.2.5 Simple Merge

This WfP does not add any requirements, but falls back on *the base requirements*.

5.2.6 Multiple Choice

This WfP is similar to the parallel split, only not all of the possible transitions are necessarily taken. Therefore does this WfP not add any requirements, but falls back on *the base requirements* and those of the parallel split, *the copyable requirement* and *the manipulatable requirement*.

5.2.7 Synchronizing Merge

This WfP is similar to the synchronization pattern and the counterpart to the multiple choice pattern. It adds no new requirements, but falls back on *the base requirements* and that of the synchronization pattern, *the data synchronizing requirement*.

5.2.8 Multiple Merge

This WfP is similar to the sequence pattern, since the threads that merge, does not actually merge data or synchronize in any way. This add no new requirements, but falls back on *the base requirements*.

5.2.9 Discriminator

To make this WfP work there must be something that makes sure that the following activity is activated only when the first thread reaches the discriminator. Therefore some message must be left behind at the discriminator to stop the threads that arrive after the first thread. This message is deleted when all the threads have been stopped, because if the discriminator is in a loop, the next iteration must also be able to launch the following activity once. This requirement will be called *the discriminator requirement*.

5.2.10 Arbitrary Cycles

Since the BP's moves in a tree, they can not move arbitrarily like they should in this pattern. It must somehow be possible to make arbitrarily moves to make this pattern work. This requirement will be called *the arbitrary move requirement*.

As the workflow languages are Turing complete and there can be more than one thread of execution, live- and deadlocks are possible. But as AmbProMois being based on a formal foundation, chances are that workflows can be analyzed and verified on correctness as opposed to the ad-hoc languages and implementations.

5.2.11 Implicit Termination

This pattern does not give any new requirements.

5.2.12 MI without synchronization

In this WfP a case, the parent, creates new cases, child cases. The new child cases can be of any BP, it does not have to be the same as its parent. This requires that the there must be some mechanism to take care of these new cases. This requirement will be called *the child case requirement*.

As the new cases can be of any BP it is also necessary to move these new cases to the machine where their first activity should be executed. This is taken care off by the *the arbitrary move requirement*.

5.2.13 MI with synchronization

I have called the WfP's: MI with a priori known design time knowledge, MI with a priori known runtime knowledge and MI with no a priori runtime knowledge collectively for MI with synchronization.

I have gathered these WfP because their only difference is in when they know how many child cases there shall be created. The number does not matter to AmbProMo, and the child BP's are taken care of by *the child case requirement* it only has to make sure that the cases can be synchronized with the parent case. The requirement here is to make sure that the workflow is not continued before all the child cases have returned to the parent case. When all have returned, the parent case can continue its workflow. This requirement will be called *the synchronizing child case requirement*.

This WfP is similar to the above except that the child cases must be brought back to the parent when they are done, so that the parent can synchronize with them before it continues its workflow. This is taken care off by the *the arbitrary move requirement*.

5.2.14 Deferred Choice

This pattern first copies the case like in a parallel split. The copies are moved to the next activity. When one of the copies are executed, the others must be withdrawn. This can be done by communicating between the copies. The problem is that if the activity executed is on two (or more) different machines at the same time, there must be some algorithm to decide which copy came first. This requirement will be called *the deferred choice requirement*.

5.2.15 Interleaved Parallel Routing

To make this pattern work, there must be some mechanism on both sides of the parallel routes that communicates. When a route is taken, the other routes are disabled until it is complete, i.e. the thread arrives at the other side of the routes. This requirement will be called *the interleaved parallel routing requirement.*

5.2.16 Milestone

To make this WfP work there must be some way for the measuring thread, to find out where the measured thread is in the workflow. This requirement will be called *the milestone requirement.*

This could be done by leaving a message at the entry and the exit of the milestone area. These messages must then be removed when the case terminates. This requirement will be called *the garbage collection requirement.*

5.2.17 Cancel Activity

This WfP would be trivial to support if there never was more than one thread in a case. To support this, it must be carefully considered when a canceled thread was 'expected' to arrive at a synchronizing activity. In this case, some action must be taken to make sure that the thread is not expected after its been canceled. This requirement will be called *the thread cleanup requirement.*

5.2.18 Cancel Case

To cancel a case, all threads belonging to the case and all child cases it has must be stopped. Therefore it must be possible to locate and stop all child cases and threads. This requirement will be called *the case cleanup requirement.*

5.3 Summary

The analysis revealed a number of requirements, some of which where expected and general in nature like, the basic requirements, and some more specific to the individual workflow pattern, like the milestone requirement.

The requirements are influenced by the foundation the underlies them, ambients, this influence has been taken into account.

To realize a system that supports all the WfP's all the requirements has to be fulfilled. For each of the requirements there may exist different solutions, I will look into these in the next chapter, the design.

Chapter 6

Design

The analysis brings forth a collection of requirements that must be met, to make a system based on mobile ambients that will support the WfP's.

These requirements are given both by the foundation on which the system is build and by the application that must run on it, represented by the WfP.

In this chapter I will show how the basic system is designed and how the requirements are met.

6.1 The Basic System

As stated in the introduction, the system I build is based on ambients. In this section I will describe how I have designed the relation between ambients and BP's. This design will be referred to as *the basic system*.

For each role in a business process, there must be somewhere for the 'keeper' of the role, to access the business processes that belongs to that role. E.g. if the boss must approve each request for holiday, the boss must be able to access these requests. Naturally the boss wants to make this decision in his office, at his desk, through his desktop computer. Therefore there is a mapping between the role (boss) and a machine (his desktop). I have chosen to make this mapping explicit, so that for each role there must be a mapping to a machine. As an example, the holiday request BP introduced in chapter 3, reproduced in Figure 6.1, the mapping could look like this:

machine	role
peter.sales.abc.com	employee
boss.sales.abc.com	boss
hr.abc.com	hr

Depending on the actual execution of a single case of the BP, one or more of these machines are visited. Therefore must the machines be available to visit

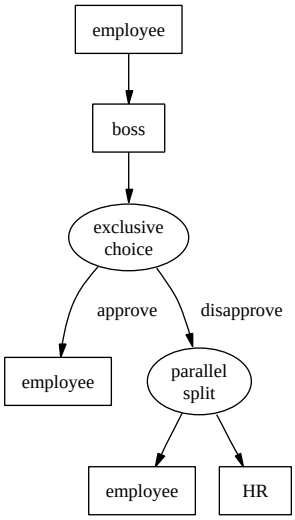

Figure 6.1: A business process to request a holiday.

for the BP. For the purpose of having machines available I have made a special kind of ambient: the *ambientnode*.

6.1.1 Ambientnodes

An *ambientnode* is an ambient with some special properties. First, it cannot move. The ambientnodes are meant to represent infrastructure, not processes, therefore I found that it would be counterintuitive if a process could include e.g. a machine and move it around. Also the ambientnodes can represent resources such as databases and special software or hardware. These may also be unmovable or undesirable to move.

Second, as the ambientnodes represent infrastructure, it would be intuitive if they were connected, therefore can ambientnodes contain other ambientnodes.

And third, as it is an ambient, it can contain other ambients. The ambientnodes has the special property of being a place where ambients can execute. Therefore it is also a place where BP's are executed.

The Figure 6.2 illustrates a couple of ambientnodes and a few ambients on these nodes. In all of the following figures the ambientnodes are represented by a big circle and ambients by a small circle, relative to each other.

As stated in the analysis the communication in the ambient tree is restricted to happen only between processes in the same ambient. I found this too restricting and counterintuitive as the BP's would have to dissolve to communicate. Inspired by boxed ambients, that allows communication between parents and children, I decided to allow communication between sibling ambients. This

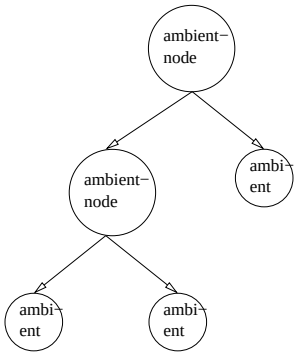

Figure 6.2: Ambientnodes and ambients.

means that ambients on the same ambientnode can communicate freely.

Every BP has a workflow that decides which roles it has to visit to make progress in its execution. The roles are mapped to a machine where the 'keeper' of the role can access the BP. This does not mean that the system forces the need for a computer for each person in a business or that it forces a one-to-one mapping of computers and roles. What it does mean is that any given role can be mapped to any computer and that whoever must interact with a business process know where it is and can access it via that computer. This does not mean the person or process that interact with the BP must be located at/in the computer, interaction could happen remotely, e.g. via a web interface.

To make progress the business processes must follow their workflow. To do this they must make themselves available to the roles that need to interact with them. Therefore they must navigate around the tree to visit the ambientnodes where the roles are that must interact with them. The actual structure of the tree of ambientnodes can be laid out in different ways.

6.1.2 Layout of Ambientnodes

The tree structure is useful because it complements many already existing structures in a business. People are organized in trees when responsibility and work are delegated, the organizational tree. Parts of the business are organized as trees into subparts like divisions and departments. There are many ways this structure can be put to good use, here I present two ideas:

First Idea - One Tree

Let the tree follow the already made structures of the business; divisions, departments, individuals. Let each of them be a node in the tree. So when a business process executes it travels up and down the organizational hierarchy. Pros and cons are:

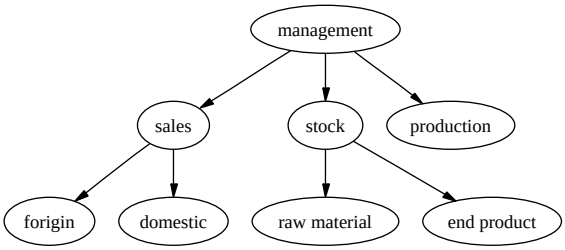

Figure 6.3: Example of a tree structured like the business.

Simple, as this only gives a tree as complex as the business organizational structure itself.

Maintainable, as this structure seldom change.

Counterintuitive, as business processes not necessarily follow this structure.

An example of this structure is given in Figure 6.3.

Second Idea - Many Trees

Let each business process have its own structure dictated by the requirements of it and including only the nodes it potentially could use. Pros and cons are:

Demanding, because when a business process is made, a tree must be designed for that business process.

Efficient, because the business process does not have to traverse unnecessary nodes.

Intuitive, as the tree is described to suit the business process instead of business process is "fitted" to suit some other structure.

An example is given in Figure 6.4.

I have chosen the last idea as basis for my implementation. Since I am focusing on mobility of business processes, this was chosen because of its efficiency and intuitive approach. The first idea make the business processes travel over computers/nodes that it will never interact with, although this can happen in the last idea, it is less likely to happen, as only nodes which are interacted with are included in the tree.

For the holiday request business process the tree could like in Figure 6.5. This figure shows three ambientnodes where the "boss" and "HR" are inside "em" (employee). These three ambientnodes represent the three roles and thereby the three machines as the previous table showed.

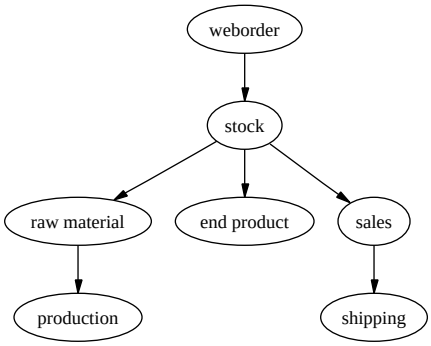

Figure 6.4: Example of a tree structured by the workflow of the business process.

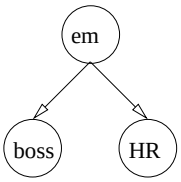

Figure 6.5: The holiday request process laid out as ambientnodes.

Any layout of the tree is allowed, but not all layouts would not be efficient. By efficient is meant that the number of moves should be minimal. I will not go into details on making an efficient layout of the tree.

With the tree in Figure 6.5 and the workflow from Figure 6.1 we can make a new version of the workflow with the movement commands of the BP inserted.

To make the movement of the BP happen, a part of its workflow is translated to a simple ambient-tree navigation language. Every time the BP changes activity it is considered whether it must move to another ambientnode, because when it changes activity, it could also change role. The ambient-tree navigation language consists of only to commands, in X and out X where X is the name of an ambient. The semantic meaning of these command corresponds to the semantics of IN and OUT of ambient calculus.

The first role of the holiday request workflow is the employee. Therefore I have made the employee root of the tree of ambientnodes. The next activity belongs to the role of the "boss", therefore the BP has to move from the ambientnode "em" to the "boss", this is simply done with an IN BOSS see (1) on Figure 6.6. Figure 6.7 shows the corresponding moves on the ambient tree. When the boss has made his decision two things can happen. If the boss disapproves the request the BP either moves back to "em" (2). If the boss approves the BP splits and one part moves back to "em" (3) and the other part moves to "HR" passing "em" on the way (4).

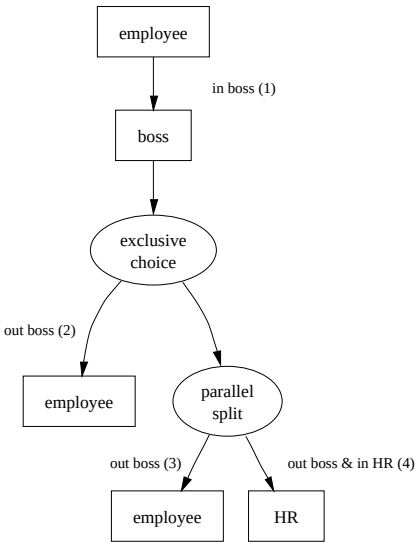

Figure 6.6: The holiday request workflow with movement commands.

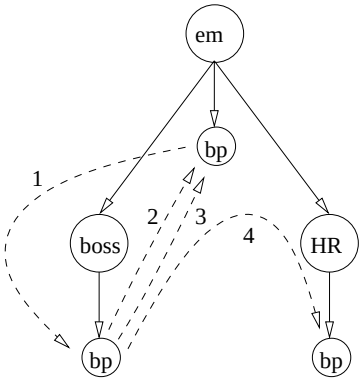

Figure 6.7: Movement of the holiday request BP on the ambientnodes.

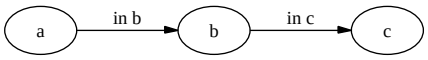

Figure 6.8: Workflow Pattern: Sequence. Translated to ambient navigation.

Ambientnodes can serve as more than a simple container for ambients. Some special ambientnodes can be made, e.g. one that contains a database which can not be moved. These uses are outside the scope of this project, but the idea will be elaborated on in the "Future Works" section of the conclusion.

6.2 Meeting the Requirements

In this section I will show how the requirements, found on basis of the WfP's in the analysis, are met.

In the following, the activities of the workflow patterns are named alphabetically. The figures with trees of ambients are also named alphabetically, this is not to create confusion but to exemplify the worst case of a workflow where each activity is located on a different computer than the activity before. Therefore when a workflow has activities A, B and C, the activities are executed on the computers A, B and C, unless otherwise noted.

6.2.1 The Movement Requirement

As shown in the previous section, ambients can model mobility and the basic structure outlined allows ambients to represent machines as ambientnodes. To make cases of BP's move, the cases are simply put in an ambient. This ambient is a child of the ambientnode where it was created and is ready to navigate the tree of ambients.

An example of this functionality is illustrated in Figure 6.8 where the sequence WfP is implemented. When the BP is done with activity A, the ambient executes a `in b` which transfers it to computer B as, when activity B is done, the ambient executes a `in c` which transfers it to computer C.

This functionality is a basic foundation of the design. Most of the following requirements will use this functionality in some way.

6.2.2 The Execution Environment Requirement

To meet this requirement a full BPMS must be running on each of the ambientnodes in the tree. This is required so that the case can be executed and sent forward to the next ambientnode.

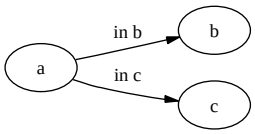

Figure 6.9: Workflow Pattern: Parallel Split. Translated to ambient navigation.

6.2.3 The Complete State Requirement

To reliably move a case, its entire state must be moved, to maintain its integrity. Therefore must the entire case be removed from the BPMS and the ambientnode it leaves and put in respectively in the destination ambientnode.

6.2.4 The Copyable Requirement

Copying a BP is not a problem on a computer. A case consists of a workflow and some data, which can be copied exactly, so that any number of similar BP's can be made. Each copy of the case is put in a ambient for themselves. These ambients can now take their own path in the ambient tree, determined by the workflow of the case they are carrying.

An example of this feature is given in the next section.

6.2.5 The Manipulatable Requirement

When a copy is made of a case, it must be possible to manipulate the workflow to make the case take the right transition next. Therefore the layer that distributes the cases must be able to understand and manipulate the workflow.

An example of the manipulatable requirement and the copyable requirement is given in Figure 6.9 where the WfP parallel split is implemented. Each ambient takes it own path through the ambient tree as shown in the figure where one executes in b and the other executes in c.

6.2.6 The Data Synchronizing Requirement

To synchronize the data one or more of the schemes suggested in the analysis must be applied. This requires, unfortunately that the workflow of the BP is extended with information about what scheme is chosen. Furthermore it requires that the BPMS can handle these schemes or that it supports manual synchronization, or both.

6.2.7 The Discriminator Requirement

To make sure that the activity after the discriminator is not activated twice, a 'blocking ambient' is left behind after the discriminator by the first thread the reaches it. The thread continues to activate the following activity. Threads that arrive after are stopped by the 'blocking ambient'. When the last thread arrives, it is stopped and the 'blocking ambient' is removed.

6.2.8 The Arbitrary Move Requirement

Since moving in a tree is restricted to moving up and down, arbitrary are not possible. Instead it is simulated, by making several moves look like one move, an atomic move, where the BP is unaware that it moves through a number of other nodes on its way. This was inspired by the Calculus of Mobile Resources mentioned in chapter 2. Also the ambient should be able to do an atomic move before and after it has executed its BP. In that way it can be 'programmed' to start at an arbitrary ambientnode and return when it is done.

6.2.9 The Child Case Requirement

To fulfill this requirement a new ambient is simply made for the child case. The ambient containing the parent case must keep track of the names of the ambients with child cases, to enable location of these in the event of the canceling of the case.

6.2.10 The Synchronizing Child Case Requirement

After a child case terminates, the parent must somehow be notified so that it can continues its execution. Therefore there must be some event mechanism that wakes up the waiting ambient when a child case has returned.

6.2.11 The Deferred Choice Requirement

To fulfill this requirement, an ambient is sent to each of the ambientnodes that can make the deferred choice. There is unfortunately no method in AmbProMo to synchronize ambients on different nodes. Therefore is this requirement left unfulfilled.

6.2.12 The Interleaved Parallel Routing Requirement

The first thread that reaches 'the other side' of the 'structure' waits there and sends back a message to signal that a new thread can begin. When the last route has been taken, the thread continues.

6.2.13 The Milestone Requirement

To fulfill this requirement, an ambient is left at the activity where the milestone area begins and one is left where the milestone area ends, when they are reached. The thread that checks whether the milestone has been reached, can check if the milestone area has been entered, and if it has, can check if it has been left again, thereby determining if the measured thread has reached the milestone.

6.2.14 The Garbage Collection Requirement

If ambients are left behind, as in the milestone requirement, they must be cleaned up, when the case terminates. Therefore when an ambient is left behind it must be recorded so that it can be removed when it is no longer needed.

6.2.15 The Thread Cleanup Requirement

If the thread is part of a split, a message must be forwarded to synchronization point, so that the thread is not waited for in vain.

6.2.16 The Case Cleanup Requirement

To cancel an entire case the ambients must keep track of which ambients are created. These must be located and canceled.

6.3 Results of the Design

All except one of the requirements of the analysis has been answered by a design. The deferred choice requirement is not fulfilled, as there is no way to synchronize ambients on different ambientnodes.

The rest of the requirements are fulfilled by using ambients as synchronizers, messages or as medium for communication.

I belive that the basic structure I have designed, is adequate to support most WfP's and it is therefore fairly solid, in my definition.

6.4 Summary

This chapter has answered all but one of the requirements found in the analysis, by a number of designs. The designs are based on the basic structure described in the first section of the chapter.

The design is solid on paper, but to demonstrate its realizability, an implementation is needed. An implementation would reveal flaws, if any, in the design. Therefore I have chosen to implement a prototype of the design, which is described in the next chapter.

Chapter 7

Implementation

To demonstrate the realizability of AmbProMo I have chosen to implement a part of the system. The design in the previous chapter answers the list of requirements found in the analysis, with a list of designs to the requirements. I have chosen to implement the designs that answer the base requirements; the movement requirement, the execution environment requirement and the complete state requirement. These requirements have been chosen because they represent the most interesting problem of realizing AmbProMo, mobility.

7.1 Software to Build Upon

To ease the effort of implementing the prototype and to avoid writing code unrelated to this thesis I have used some existing software packages to build upon. I needed a BPMS that supports various WfP's to execute the BP's. I also needed a framework to move objects between machines.

For the BPMS package I needed the source code so I could add support for mobility, therefore I only considered open source packages.

The packages I looked at will be presented in the following two sections along with the reasons for my choices.

7.1.1 Choosing a Mobility Framework

I searched for frameworks that could provide mobility of state and processes. I found a number of frameworks of which I looked further into: Pathwalker[11] by Fujitsu, FlexiNet[5] by ANSA and μCode[2] by Gian Pietro Picco.

FlexiNet looked promising at first, because of its extensive API. Unfortunately it did not work well with the current versions of Java and since it apparently no longer is being maintained, I abandoned it.

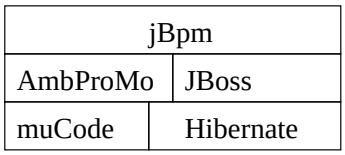

Figure 7.1: The layers of the implementation.

Pathwalker was more lightweight than FlexiNet and was based on simple message passing. Unfortunately it too gave a mysterious runtime error. The source was not available, so I could not fix it.

μCode turned out to be usable. Despite a problem with serialization in Java 1.4 it worked flawlessly with Java 1.3.

μCode is a small framework for mobility of code and state. It is made by Gian Pietro Picco and it is meant to be used in research and teaching.

It provides primitives for mobility of objects and classes. In its core it provides a basic unit of mobility, called a *group* and a place where groups execute, called a *μServer*.

7.1.2 Choosing a BPMS

To build the prototype I needed a BPMS to execute the BP's. I looked at two BPMS's: XFlow[12] and jBpm[10].

XFlow does not have a strong support of WfP's, it only supports exclusive choice, parallel split, synchronization, discriminator, multiple instances without synchronization and multiple instances with a priori design time knowledge.

jBpm has more focus on the WfP's, but does not support many directly either. I chose jBpm for the prototype because of the authors commitment to support the WfP and because it has the least worst developers documentation and the most professional 'feel' of the two.

jBpm is a workflow management system that can execute BP's described in a language called jBpm Process definition language (jPdl). It is a complete BPMS that have interfaces to both external applications and users.

It is build to run on JBoss[9], a java application server and uses an Object/Relational Persistence system called Hibernate[8] for its data storage needs. The relation between jBpm, JBoss and Hibernate are illustrated in Figure 7.1.

7.2 The Movement Requirement

In the following sections I describe the implementation of the design that answers the base requirements.

The movement requirement, is the requirement of the ability to move a BP from a machine to another; mobility of BP's.

To support mobility I used the μCode framework which provides basic mobility primitives. On top of this framework I have implemented the ambients that provide mobility of BP's in AmbProMo.

7.2.1 Mobile Ambients in μCode

My implementation of Mobile Ambients use the group primitive of μCode to move between machines. This is illustrated in Figure 7.1 where AmbProMo is located on top of μCode which means that AmbProMo depends on features of μCode.

An ambient class is implemented from which ambients can be instantiated. The ambients can have a parent, which is an ambient, and children, which also are ambients. They can move in and out of ambients and can carry a BP.

Ambientnodes inherits from ambients. They can not move, but ambients can move into and out of ambientnodes.

This part of the implementation gives ambients that lives on ambientnodes and carry a BP. They can move in and out of other ambients thereby bringing mobility to BP's.

The most important classes of the implementation of ambients are Ambient, AmbientHost and ShadowAmbient. I will describe these classes in the following.

The Ambient class is the base class of ambients. It contains methods to move in and out of other ambients. These methods rely on the distributed implementation of mobile ambients described in chapter 2. To use the class and the ambient capabilities a new class must be made that inherits from Ambient.

The AmbientHost (corresponds to an ambientnode) class inherits from the Ambient class. This class is used to make environments for ambients to execute in. It has a "ship" method that can be used to send ambients to another ambientnode. The class overloads some of the methods of the Ambient class to make ambients move from one ambientnode to another, by using its "ship" method.

The ShadowAmbient is a placeholder that represents an ambientnode. It is used to make the ambient tree look like one tree. All it does is that it holds the address of another ambientnode, so that when an ambient moves into another ambientnode, the address of that node is fetched from the ShadowAmbient.

The following is a walk-through of how an ambient is moved from an ambientnode to another. Specifically I will show how an IN is performed. The first code

listing is from the Ambient class. When an ambient calls the IN command, it sends a "delegate" message to its parent asking for permission to move into one of the parents children. The message is implemented as a method call (line 242) because an ambient will always be on the same machine as its parent.

```
239     public void in (String inA) {
240         Ambient newparent = null;
241         try {
242             newparent = parent.delegateIn(this, inA);
243             if (newparent != null) {
244                 newparent.register(this);
245             }
246         } catch (Exception e) {
247             e.printStackTrace();
248         }
249     }
```

The delegateIn method is in the class HostAmbient. It checks, whether the destination ambient exists (line 63) and fetches the address of destination ambientnode from the ShadowAmbient (line 65). Before shipping the ambient (line 67) it makes sure that the ambient does not have references to the local machine that will ruin serialization. The "ship" call wraps up the ambient in a wrapper that can unpack the ambient at the destination.

```
59      public Ambient delegateIn(Ambient child, String into)
60          throws MuAException {
61          Ambient intoA = getChildAmbient(into);
62          if (children.contains(child)) {
63              if (children.contains(intoA)) {
64                  if (intoA.getClass().equals(ShadowAmbient.class)) {
65                      ShadowAmbient dest = (ShadowAmbient) intoA;
66                      child.prepareSerialize();
67                      ship (child, dest);
68                  }
69              } else {
70                  throw new MuAException ("Destination ambient"
71                                          + "does not exist");
72              }
73          } else {
74              throw new MuAException ("Moving ambient not"
75              + "registered as child of this ambient");
76          }
77          children.removeElement(child);
78          return null;
79      }
```

When the ambient wrapper arrives at the destination, the ambient is unwrapped (line 18), registered at the ambientnode (line 32) and pointed toward its new parent (line 33).

```
17      public Thread unpack (Group g) {
18          Ambient a = (Ambient) g.getObject("_AMBIENT_");
19          String position = (String) g.getObject("_POSITION_");
20          MuAmbientServer m = (MuAmbientServer) g.getServer();
```

```
21          HostAmbient ha = m.getAmbient();
22
23          if (position != null
24              && position.equalsIgnoreCase("parent")) {
25              ha.setParent(a);
26              a.register(ha);
27              a.setHost(ha);
28          } else if (position != null
29                      && position.equalsIgnoreCase("host")) {
30              m.setAmbient((HostAmbient) a);
31          } else {
32              ha.register(a);
33              a.setParent(ha);
34          }
35          return new Thread(a);
36
37      }
```

The ambient is now in place at the new ambientnode.

7.3 The Complete State Requirement

The complete state requirement is the requirement of moving the entire BP from one machine to another.

Figure 7.1 show the relation between AmbProMo and jBpm. jBpm is placed on top of AmbProMo which means that jBpm depends on features in AmbProMo.

Since jBpm was build on Hibernate, I could, by looking at the 'database schema' deduce that the entire case was stored in Hibernate and therefore take the object and move it to another instance of the BPMS without complications, as there were no object references from the BPMS to the case left behind.

7.4 The Execution Environment Requirement

The execution environment requirement is the requirement of having a BPMS that can execute the BP's on the machine.

As jBpm is a complete BPMS it also provides an environment for execution of BP's.

To hook into this environment I found the code in jBpm that stores the cases in Hibernate. Recall that when an activity in a workflow is done, the next activity may "belong" to somebody else to work on. Therefore when an activity is done, it may be time for the case to move to another machine. Therefore when a case is saved, it is a good time to check whether it should be moved to another machine (also I am sure that the case is in a consistent state). Therefore the case is inspected at this point and if it should move, it is packaged in an ambient

and sent to its destination, then removed from the execution environment on the machine.

This part of the implementation provides a safe execution environment for the BP's.

7.5 Results of the Implementation

The implementation is the realization of the base requirements. The base requirements are those requirements that collectively enables business processes to be mobile.

The implementation uses two software packages to reduce the need for writing code unrelated to the problem of making business processes mobile. The experiences with these packages have been mixed.

The implementation of ambients was made on top of the μCode framework. This framework has minimal primitives for mobility but I found these simple primitives hard to use, but as an overall the package save time on the implementation.

The jBpm package provided a good framework for implementation, although it was complicated to straighten out its relations to JBoss and Hibernate.

7.6 Summary

With the implementation described above I have realized the selected part of the design, which in turn was an answer to the requirements found in the analysis.

This implementation shows that mobility of BP's is feasible with the foundation I use and based on the design I have made.

Although this implementation only realizes a part of the design, it demonstrates that the base requirements for mobility are fulfilled.

Chapter 8

Conclusion

Business Process Outsourcing gives the possibility of reducing costs and regaining focus on core competences of a business. To use BPO, business processes must be defined by a workflow language. Unfortunately current workflow languages does not support mobility of business processes, which is needed to outsource business processes efficiently. Therefore I propose a new foundation for the workflow languages that support mobility; mobile ambients. My problem statement was:

The main goal of this thesis is to demonstrate that mobile ambients can provide a solid foundation for mobile business processes.

In the introduction I gave my definition of solid: *By solid I mean that mobile ambients should support mobility for all kinds of business processes.*

To demonstrate this I have analyzed mobile ambients and WfP's to find the requirements to build such system. The analysis finds a set of requirements that must be fulfilled to support all the WfP in a system based on ambients.

In the design chapter I take the set of requirements found in the analysis and make a design that takes the requirements into account. All except one of the requirements are answered by the design and I believe that the system, is fairly solid. The design takes into consideration the common workflow patterns that are used in workflows and the possibilities and limitations that mobile ambients inflict on the system.

To demonstrate that the design is realizable I have implemented a working prototype that supports the most important requirement, that of mobility. The prototype revealed no flaws in the design, but as the prototype does not cover the entire design, it can only answer for the implemented part.

A more exhaustive implementation would give a better basis for demonstrating that the design is realizable when it comes to supporting all workflow patterns.

8.1 Future Works

As the foundation, Mobile Ambients, have been sliced and diced to fit the application of mobile business processes, it would be nice to collect the pieces and determine what would make a calculus of mobile business processes. Or, to take another approach, determine how to extend a calculus or a workflow language that supports relevant concepts for mobile business processes, like the concept of location or roles.

To further demonstrate that the design is realizable a complete implementation would be desirable. This would also give further experiences with mobile BP's in general.

The ambientnodes leaves room for flexibility. One thing they would be suited for is to control the domain they naturally have by being the parent of a branch. They could enforce security, e.g. deciding which BP's are allow to enter and which information is allowed to leave.

Another use of the ambientnodes are for special unmovable ambients like ambients that control physical things, ambients that control special software that runs on special hardware or simply ambients that control processes that undesirable to move, like a large databases or data warehouses.

Bibliography

[1] See: http://cyberatlas.internet.com/markets/b2b/article/0,,10091_2220371,00.%html, 2004-02-10.

[2] See: http://mucode.sourceforge.net.

[3] See http://tmitwww.tm.tue.nl/research/patterns/.

[4] See: http://www3.gartner.com/teleconferences/attributes/attr_46768_115.pdf.

[5] See: http://www.ansa.co.uk/.

[6] See: http://www.bpmi.org.

[7] See: http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm.

[8] See: http://www.hibernate.org.

[9] See: http://www.jboss.org.

[10] See: http://www.jbpm.org.

[11] See: http://www.labs.fujitsu.com/en/freesoft/paw/.

[12] See: http://xflow.sourceforge.net/.

[13] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *In Proceedings of the 4th International Conference on Theoretical Aspects of Computer Science (TACS'01)*, volume 2215 of *LNCS*. Springer-Verlag, 2001.

[14] Luca Cardelli. http://www.luca.demon.co.uk/Ambit/AmbitPkgRelease\%201.1/AmbitIntro.htm%l.

[15] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structures: First International Conference, FOSSACS '98*. Springer-Verlag, Berlin Germany, 1998.

[16] Cedric Fournet, Jean-Jacques Levy, and Alan Schmitt. An asynchronous, distributed implementation of mobile ambients. In *IFIP TCS*, pages 348–364, 2000.

[17] Jens Christian Godskesen, Thomas Hildebrandt, and Vladimiro Sassone. A calculus of mobile resources. Technical Report TR-2002-16, The IT University of Copenhagen, 2002.

[18] Robin Milner. "a calculus of communicating systems". In *"Lecture Notes in Computer Science"*, volume "92". "Springer-Verlag", "1980".

[19] Robin Milner. The polyadic pi-calculus: a tutorial. 1991.

[20] C.A. Petri. Kommunikation mit automaten. Doctoral thesis, University of Bonn, 1962.

[21] Davide Sangiorgi. From pi-calculus to higher-order pi-calculus - and back. In *TAPSOFT*, pages 151–166, 1993.

[22] Bent Thomsen. *A Calculus of Higher Order Communcating Systems*. PhD thesis, Imperial College of Science and Technology, 1989.

[23] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(3):5–51, July 2003.

[24] W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske. Business process management: A survey.

[25] B.C. Warboys, P. Kawalek, I. Robertson, and R.M. Greenwood. *Business Information Systems: a Process Approach*. McGraw-Hill, first edition, 1999.