UNIVERSITY OF OSLO
Department of Informatics

Process Management and Orchestration

Wei Gao

Network and System Administration

Oslo University College

Spring 2009

Process Management and Orchestration

Wei Gao

Network and System Administration
Oslo University College

Spring 2009

Abstract

Process automation is a concept used in logistics that has been adopted in some data centre management software suites recently as "Orchestration managers". This project is about mapping out and comparing approaches to process management. Specifically, two popular and very different process modelling methods were compared. One is the BPMN (Business Process Management Notation) that uses the traditional method to model the process in a flow. The other is the Promise Theory that models the process in a network of interacting autonomous agents. Our research question is what's the differences between two methods. We used Promise Theory in two ways: as a framework for discussing modeling of processes, and as a tool for modeling. Results show that Promise Theory can model more features of a process than BPMN. Promise Theory with fewer symbols is easier to learn but requires more thinking when using than BPMN. Under the circumstances where there are many asynchronous activities or many agents/roles involved in the process, Promise Theory has better performance to model the agents' autonomous behavior and the interaction between them.

Acknowledgements

Preface

I surprised myself that night when I finished my first version of this paper draft. It felt so nice to read my own words about describing a scientific problem in a neat way.

I used to doubted myself from time to time during the study of the project whether I had the ability to discuss a complex problem like this. There are already many approaches about process management. The Promise Theory that I am exploring to introduce is quite new and totally different from the way in which most traditional methods hold. Which perspective should I choose and what was the value?

However, there was always a sound around me when I wanted to stop: "Our life needs new ideas. Without innovation a society will stop developing, no matter how prosperous it used to be, such like the four major ancient civilization. The new ideas need brave people to bring it up and test it. In the end no matter you succeed or not, the value is that you have tried..."

This belief drove me to carry on the research, discussion, testing and writing all these months. I don't know why, but I just beleive it. Finally my relying on that belief promises me a rose garden now. Maybe that is the mystery of promises.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

By placing business processes on center stage,
corporations can gain the capabilities they need to innovate,
reenergize performance and deliver the value today's markets demand.

– Howard Smith & Peter Fingar

1.1 What is Process Management?

Process thinking was formally introduced to the companies in the 1990s, which looks horizontally through the company for inducing improvement and measurement, because the traditional "function" and "procedure" thinking failed to measure and support improvement in cross-function activities as the complexity grew.

A process is not a hard concept to understand. A collection of a process definitions can be found in section 2.3. They are in some sense quite similar to each other. Simply a process is a set of activities designed to produce a service or product to the customers.

Then process management is about how to manage these processes effectively and efficiently. Process management is the ensemble of activities of planning and monitoring the performance of a process.[1] It is usually an integrated application of knowledge, skills, tools, techniques and systems to define, visualize, measure, control, report and improve processes. The activities which constitute business process management can be grouped into five categories: design, modeling, execution, monitoring and optimization. They can be easily understood literally. The steps can be viewed as a cycle, see fig 1.1, but the economic or time constraints are usually limit the process to one or more iterations in reality.

Process management is now several decades old. As the fast development of information technology, IT technology has been an indispensable factor in most business processes. More and more business executives and IT managers

Figure 1.1: BPM life-cycle

realized that IT has been the enabler of business in some sense which directly rose the tide of developing process management approaches and tools to automate or semi-automatic the business processes.

Orchestration and choreography have been two popular concepts in the process management/automation field today. Many languages and software name themselves as orchestration language or software such as HP Operations Orchestration software, Orchestration Manager developed by Blade-Logic, ZENworks Orchestrator developed by Novell[2] and WS-BPEL (Web Service-Business Process Execution Language) and WS-CDL (Web Services-Choreography Description Language) also claims themselves as orchestration and choreography langauge separately. What are orchestration and choreography then?

The orchestration and the choreography distinctions are based on analogies: orchestration refers to the central control (by the conductor) of the behavior of a distributed system (the orchestra consisting of many players), while choreography refers to a distributed system (the dancing team) without centralized control. Someone who favors orchestration argues that the primary difference between orchestration and choreography is executability and control. [3] An orchestration specifies an executable process that involves message exchanges with other systems, such that the message exchange sequences are controlled by the orchestration designer. A choreography specifies a protocol for peer-to-peer interactions, defining, e.g., the legal sequences of messages exchanged with the purpose of guaranteeing inter operability. Such a protocol is not directly executable, as it allows many different realizations (processes that comply with it). A choreography can be realized by writing an orchestration (e.g. in the form of a BPEL process) for each peer involved in it. As far as I am concerned, it has exposed some aspects of the differences, but not complete. However, someone who favors choreography believe that choreography has better performance in describing collaborations of participants from a global viewpoint.[4]

Among the ideas, I like the opinion from Jim Alateras which illustrates things in a mild and neutral way.[5] He believes that: ″ The difference between orchestration and choreography boils downs to perspective ″. And he used a classic diagram which depicts three participants collaborating in the business process (i.e. purchase and ship a book) to illustrate it, see fig 1.2. Choreography is concerned with all the message exchanges between all the participants engaged in the process. It's a birds-eye perspective of the process. Orchestration, on the other hand is only interested with message changes form the perspective of a single participant. Therefore in the diagram, he used the yellow circle to represent the birds-eye perspective and the red ovals to represent each participant's perspective.
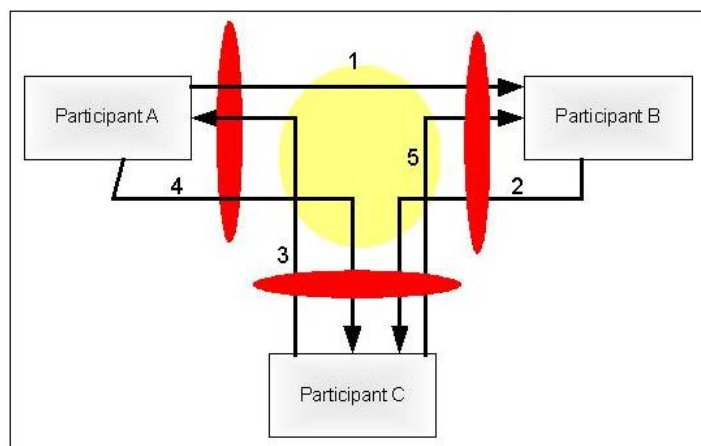
Figure 1.2: Illustrating Orchestration and Choreography

1.2 Motivation

″Don't necessarily do it the way everyone else did. Just find some way of doing it cheap and effectively – so we can learn.″

Process is not an unacquainted concept to all of us. No matter you are aware of it or not, actually you have already been experiencing and managing the processes in daily life in your own way.

To make dinner for a friend, firstly you have to prepare all the necessary materials, e.g. rice, meat, vegetables, sauces, etc. Maybe some of the discreet person would look into ″The Most Popular 100 Menus″ for a proper menu before their purchasing, then the cooking process starts with a series of activities involved: put the rice and proportional water in the rice pot, boil it, wash the meat and vegetables, cut them into right size, heat the pan, saute the meat, stir-fry the vegetables, then mix them with your favorite sauces, then put on plate and serve. Within these activities, some of them should be acted in sequence while some could be performed in parallel. For example, you could cook the

meat and vegetables while boiling the rice, which would save you much time. Still there could be creative activities involved, e.g., you would like to warm the oven while you are doing those things above, so that you could put the sweet flour paste with apple pieces into it and set the timer afterwards. If you are lucky, you could enjoy the delicious apple pie right after your main course.

More seriously, we could easily recognise that making a dinner is a process and there are many activities involved. Usually we proceed these activities in an unconscious state. We choose to do something first and something afterwards just because our mothers used to cook like that, or the books tell us to do so, or we just figure out these processes ourselves according to some common sense.

However, it is obvious if we have a good knowledge of the activities involved and their relationship, we can make a good plan then conduct the activities in an effective and efficient way. Furthermore it would be nice if we have a robot who can understand our language and it will make the dinner after we give it some commands.

The same thing can be applied to business in which the process is to produce a product or service to the customers. However, as the trend of more and more intervention of information technology in business process and the globalization of markets, the complexity of the process increased rapidly. It is reflected in a way of more participants in the process and more interactions among the participants. It has never been such a need to find a better way to manage these processes.

It has not been a flat way to find out the right prescription. Many gurus and theories have contributed and people are still pursuing now. This thesis project work is aiming to map out what has happened and explore to add one stone on this road as well.

1.3   The Frameworks for processes

Since the new millennium, there are two categories of approaches enjoying the most popularity in the process management market.

One category is a kind of process framework or infrastructure which defines measures for each process and describes "best practices" used with each. They are usually written and published by a consortium in a specific business area or government. Famous example of this is the ITIL (IT Infrastructure Library) developed by United Kingdom's Office of Government Commerce (OGC) and the eTOM developed by a consortium of Telecom executives. They are quite like "The Most Popular 100 Menus" in the previous example of making a dinner.

We shall refer to ITIL because it has become a popular set of guidelines for all manners of IT organizations, and because it promotes the idea of IT-business alignment. ITIL has grown in popularity in the past ten years. It is published in a series of books and used by companies and organizations worldwide to establish and improve their capabilities in IT service management. Though there are several versions of ITIL, it is the ITILv2 (version 2) which experiences the biggest distribution and popularity due to the fact that International ISO/IEC 20000 standard has emerged from the basic principles and processes coming from it. As its name clearly tells that it is like a knowledge library containing many good practices for IT service management. ITIL uses the Deming quality circle as a model for continual quality improvement, where quality both relates to the provided IT services as well as the management processes deployed to manage these services.

The current version of ITIL is ITIL v3 which was released in May 2007 and consists of five core publications[6], see fig 1.3. Each provides the guidance necessary for an integrated approach, as required by the ISO/IEC 20000 standard specification:

- Service Strategy

- Service Design

- Service Operation

- Service Transition

- Continual Service Improvement
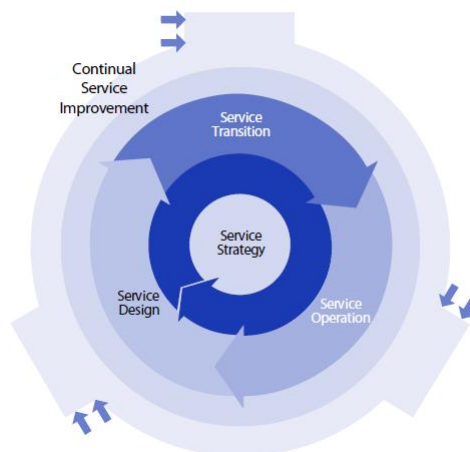
Figure 1.3: ITIL Core

However, there is a lot of confusion about ITIL, stemming from all kinds of misunderstandings about its nature. The problems are:

- It is obvious that ITIL is not a language itself but just a Quality Assurance framework without base on any theoretical model or design criteria.

- ITIL is, as the OGC states, a set of best practices. If we hold a scientific view point then interesting questions come: how can they say that ITIL practices are better than other practices? It would be kind of bureaucratic if the answer is because they are published by the government and authorities.

The second category of the process management approach is the process modeling languages. A model is a simplified view of a complex reality. Joshua M. Epstein has offered sixteen reasons why to build a model.[7] In business sense, it is a means to creating abstraction, allowing you to better understand and analyse the current key mechanisms of an existing business, to seek further improvement in the business structure and operation, also to be the basic action plan for an innovated business, e.g. compliance with new business criteria.

Meanwhile under the great trend of IT-business alignment, only a process model seems not enough. It is also desired that the process modeled with a language could be automated or semi-automated afterwards to elevate efficiency and contain costs. Therefore, an executable process modeling language is really needed such as BPEL (Business Process Execution Language) and Cfengine. See a brief introduction of them in section 2.2.1 and 2.6.

We choose BPMN (Business Process Management Notation) as one of the targets in this project because it meets the requirements above, and also because it is a standard nowadays for business process modeling which enjoys lots of popularity. BPMN is a graphical representation for specifying business processes in a workflow. It is based on a flowcharting technique very similar to activity diagrams from Unified Modeling Language (UML). It is designed to be able to map to any business process modeling language, particularly BPEL. It was developed by BPMI (Business Process Management Initiative), and it currently maintained by the Object Management Group (OMG) since the two organizaiotns merged in 2005. It consists of one diagram - called the Business Process Diagram (BPD) which is more than a flow diagram since it shows the association of data artifacts to activities as well. An example of BPD is shown in fig 1.4.

Other languages are not selected either because they are not process oriented, or they are not executable,or they are not popular standard. See a brief introduction of the other languages such as UML and WPDL in section 2.2

However, it seems that BPMN is also not the perfect answer in every aspect.

1. The scope of BPMN is constrained to support only the concepts of modeling that are applicable to business processes. This means that the other types
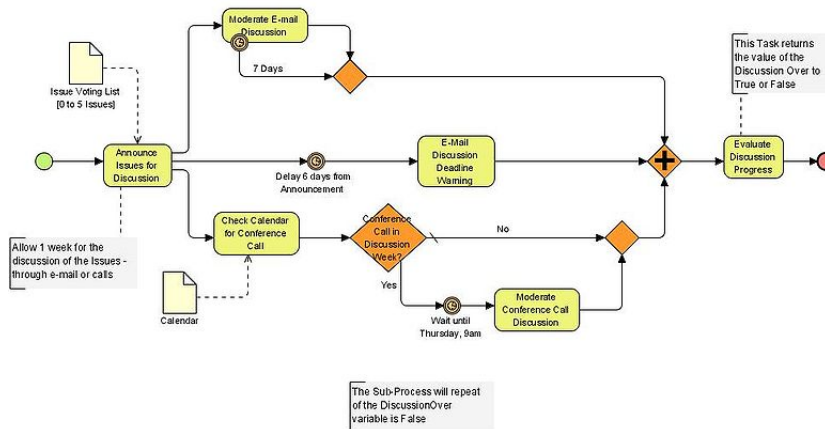
Figure 1.4: An example of BPD

of modeling done by organizations for non-business purposes will be out of scope for BPMN. For example, the modeling of the organizational structures, functional breakdowns and data models will not be a part of BPMN. Also BPMN and BPEL's focus on the web services,which is, on one hand, an advantage as more and more modern business precesses are relying on web services, on the other hand, which could also be a limitation since they put the processes having no relationship with web services aside.

2. Further more, the existing process modeling languages all tend to define the process as a workflow/algorithm starting from a fixed known point and follow some kind of flow/algorithm/pathway, one step at a time from start to finish, whose aim is to drive a system from state to state in a linear fashion. The problem is, if one starts with a blank slate, this is simple and straightforward, but when describing change relative to an unknown or inconvenient starting point, it is a fragile model that mandates a lot of testing and reactive logic to work.[8]

We can demonstrate this point using the famous two theories about how to maintain state/process: **Congruence** and **Convergence**. Congruence theory is the traditional theory, like climbing a mountain, see the right picture in fig 1.5. It is all about sequence, you have to know where you start and where you end, also every step which is dependent on its previous step. If any step goes wrong then everything should come to the initial state and do all over again. While Convergence theory just defines where you end and it works for any initial state, see the left picture in fig 1.5. In short, a workflow language like BPMN has something similar to the picture on the right .

3. Last but not least, since BPMN describes every steps in the process and seems trying to break the whole process as detailed as possible, which to some extent has the suspicion of micra management. Micro management is to man-

Figure 1.5: Change and Persistence

age with great or excessive control, or attention to details. For the person who is responsible for promising a service/product to end users, what they care most is the result that they can successfully deliver that service/product. At the same time, they would not like to be overloaded with all the detailed internal information. Thus, the "blackbox" package way of managing things is quite popular in data centers nowadays. Sometimes it is nice to hide appropriate amount of details in the blackbox or package and only present the users with the necessary information.

Therefore, all the problems above are the reasons why we introduce Promise Theory as our second target in this project. Promise Theory seems to have the properties to solve the problems above. Promise Theory is a graph theoretical framework for understanding complex relationships in networks, where many constraints have to be met, which was developed by Mark Burgess at Oslo University college in 2004. It uses a constructivist approach that builds conventional management structures from graphs of interacting, autonomous agents. Promises can be asserted either from an agent to itself or from one agent to another and each promise implies a constraint on the behavior of the promising agent. The newly released software language cfengine3 is designed basing on Promise Theory, which makes the processes modeled by Promises Theory are also executable afterwards.

The special properties of Promise Theory such like autonomous and recursive makes it might be a useful framework for process management. The whole process can be expressed as an overall big promise and promise can contain promises which make it a black-box approach from top. The autonomous agents and promises idea can help users to map out the dependencies in the process.

Finally, we choose BPMN and Promise Theory as the two candidates for process management in this project because it is possible to use them to com-

pare whether a process is better than another in a scientific way. For example, it is possible to count the number of the matric below in two processes:

- number of agents

- number of links/messages

- number of promises

- number of events

Then to compare whether it is bigger, smaller or equal in two cases. Depend on different situations, analysis of the result can always give you some help in decision making of which process is better than another.

In this project, we are trying to represent processes in terms of promises and do a comparison between BPMN and Promise Theory. The broad research question is:

- What's the difference between BPMN and Promise Theory in modeling processes?

1.4 Problem Statement

To narrow it down, in this project we are concentrate on:

- Is Promise Theory more expressive than BPMN? If so, in what ways? Does the language allow describing situations that are not describable via BPMN?

- Can Promise Theory allow solving problems that BPMN doesn't solve? i.e., is there a way of "interpreting" data so that there are new results?

  - Are there processes in real life whose asynchronous properties keep them from being expressible in BPMN, and what can we learn from them modeling via promise?

1.5 Methodology

In order to answer the research question, a comprehensive learning of process management and two particular methods – Promise Theory and BPMN is needed first. Since process management has already a long history, we don't lack related books, papers and documents. For the two methods, Promise Theory is born in Oslo University College, so it is convenient to find related documentation and also the help from its founder Mark Burgess. On the other hand, since BPMN has been published as a standard, it is not so hard to find related materials on the internet and library.

Then we need some methodology to do the comparison between BPMN and Promise Theory on their performance in process management. In this project,we are going to use the "Promise Approach" to do the analysis.

1. Define all the autonomous parts.

   In our case, the autonomous parts are the comparison criteria that we are going to examine between BPMN and Promise Theory. All the criteria are actually promises. They are in the form of "does the method promise to model a certain property of a process?" For example, does Promise Theory promise to model the order/sequence of activities in a process? Or does BPMN promise to model the cooperation between the roles/agents?

   It is a bit hard for us to find out a standard criteria for process modeling to compare due to the lack of related paper and documents. Our solution is to go for the published specification of BPMN to see which criteria BPMN is proud of in modeling process. Since BPMN has been set as the standard for business process modeling, it must have covered most of the desired properties of process modeling. Then we try our best to find the holes of each method to add the compared criteria.

2. Examine the criteria on two methods.

   In order to make the result more complete,we use two independent ways in the examining. One way is a syntactic mapping between BPMN and Promise Theory: we apply the definitions in BPMN and figure out whether there is a promise-theoretic version of each BPMN construct. And also the other way around, is there a BPMN-version of each Promise Theory special property? The other way is testing the two approaches in scenarios modeling. Each scenario is modeled using Promise Theory and BPMN twice. Then we can have an intuitive feeling of the difference of two methods.

   We are trying to be careful in selecting the scenarios, since we might choose all the scenarios which are incidently only good for one method to show its advantage in modeling or they are so similar that the test span is not enough. Then the result will be unfair in some sense in the former case and and not so valuable result in the latter case. However, there is always dilemma in this situation. Our solution is trying to make the scenarios more general, try to let people, machine and abstract things all involved in the scenarios. Also there are some assumed scenarios to better hit the point we have predicted about certain criteria we think that one method might be better than another.

3. Summarize the result and form a checklist

Finally the result will be in the form of a checklist. And we can count the number of hit features for each method, then we can tell which one is better than another in feature comparison.

1.6   Thesis outline

Chapter 1 presents an overall introduction of the project including the motivation, problem statement and methodology of the project. Project relevant background information and materials are presented in Chapter 2. We introduce Promise Theory and discuss why Promise Theory might be a useful method for process management in Chapter 3. A syntactic mapping between BPMN and Promise Theory is shown in Chapter 4. Then we test to model using BPMN and Promise Theory in six scenarios in Chapter 5. A discussion of the mapping and scenario modeling is in Chapter 6. Finally Chapter 7 shows the conclusion of this project and future work.

Chapter 2

Background and Relevant Materials

2.1 A short history about process management

It is worthwhile to trace a little back to the process management history to have an general idea of how this concept comes from and what people have done about this, see fig 2.1.
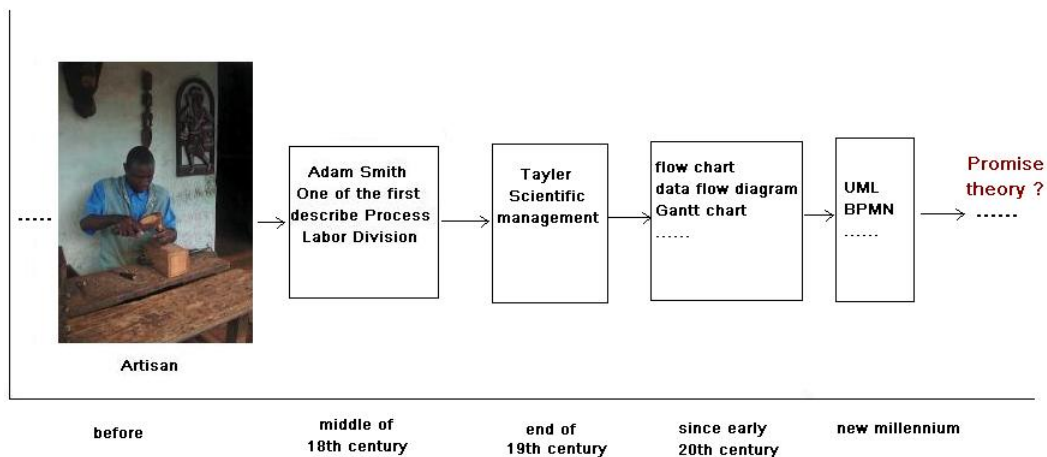
Figure 2.1: A short review about process management

One of the first people to describe processes was Adam Smith in his famous example of a pin factory in 1776. He described the production of a pin in the following way:

"One man draws out the wire, another straights it, a third cuts it, a fourth points it, a fifth grinds it at the top for receiving the head: to make the head requires two or three distinct operations: to put it on is a particular business, to whiten the pins is another... and the important business of making a pin is,

in this manner, divided into about eighteen distinct operations, which in some manufactories are all performed by distinct hands, though in others the same man will sometime perform two or three of them."

Smith also first recognized how the output could be increased through the use of **labor division**. Previously, in a society where production was dominated by handcrafted goods, one man such like an artisan would perform all the activities required during the production process, while Smith described how the work was divided into a set of simple tasks which would be performed by specialized workers. The result of labor division in Smith's example resulted in productivity increasing by 24,000 percent, i.e., that the same number of workers made 240 times as many pins as they has been producing before the introduction of labor division.

However, Smith's view was limited to the same functional domain and comprised activities that are in direct sequence in the manufacturing process. As the complexity of processes grows, an appropriate level of task division was desired. Frederick Winslow Taylor developed the concept of **scientific management** in the end of 19th century. Taylorism is a variation on the theme of efficiency which is a late 19th and early 20th century instance of the larger recurring theme in human life of increasing efficiency, decreasing waste, and using empirical methods to decide what matters, rather than uncritically accepting pre-existing ideas of what matters. Taylor believed that decisions based upon tradition and rules of thumb should be replaced by precise procedures developed after careful study of an individual at work. Taylor has developed a scientific approach for business management and process improvement.

However, criticism exists towards applications of scientific management which sometimes fail to account for two inherent difficulties. One is that it ignores individual differences since the most efficient way of working for one person may be inefficient for another. The other is that it ignores the fact that the economic interests of workers and management are rarely identical, so that both the measurement processes and the retraining required by Taylor's methods would frequently be resented and sometimes sabotaged by the workforce. The character of Charlie Chaplin in the film "Modern Times" gives an vivid reflection of poor situation of the workers in the factory doing repetitive work all the time.

The current management and improvement approach, with formal definitions and technical modeling, has been around since the early 1990s. The classic business process modeling methodologies such as the flow chart, functional flow block diagram, data flow diagram, control flow diagram, Gantt chart, PERT diagram, and IDEF have emerged all over the 20th century. See example of these diagrams from Wikipedia in fig 2.2. We can see the flowchart or diagram shows the flow of the process or data well and the Gantt chart provides you good view about how to schedule your work. Methods from the new millennium here are Unified Modeling Language, Business Process Mod-

eling Notation and etc. They are more complex and try to give users more comprehensive view of the process. Although the initial focus of business process management was on the automation of mechanistic business processes, it has since been extended to integrate human-driven processes in which human interaction takes place in series or parallel with the mechanistic processes.
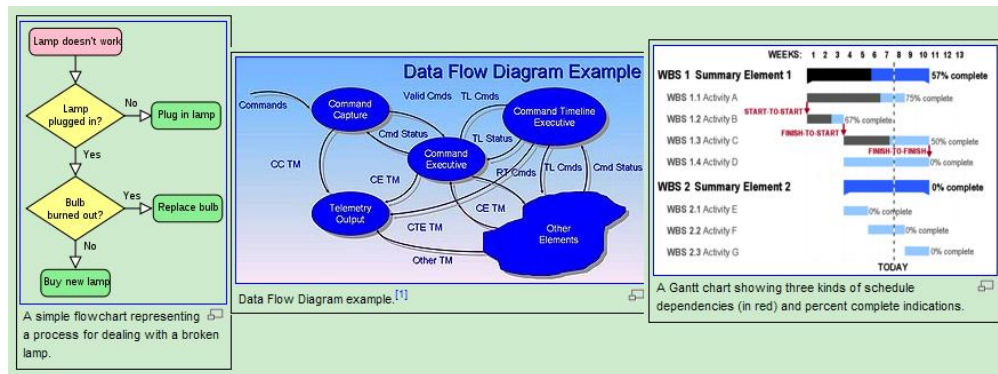
Figure 2.2: Example of Flowchart, Data flow diagram and Gantt chart

2.2 Current approaches to process management

2.2.1 BPEL

Web Service Business Process Execution Language (WS-BPEL 2.0, or in short: BPEL) created in a joint venture by BEA, IBM, Microsoft and others, is one of the vogue process modeling languages currently. It is an executable language for specifying interactions with Web Services. Actually it was renamed from BPEL4WS 1.1 which is known by most people in 2004 to align BPEL with other Web Service standard naming conventions which start with WS-. An XML-based language, BPEL's focus on modern business processes led BPEL to adopt web services as its external communication mechanism. BPEL's massaging facilities depend on the use of the Web Services Description Language (WSDL) which is an XML-based language that provides a model for describing Web services, to describe outgoing and incoming messages. BPEL claims that it is an orchestration language.

Due to the popularity of web-based services, its focus on web services enables its use to be expected to spread further which is also a limitation of itself indicating constrain. What is more, it is true that the centralized orchestration topology provides the orchestration designer (conductor) more control over the whole system. But on the other hand, it may also indicates more risk of the system, what if the orchestration designer down or made a mistake? We all know risk is equivalent to money in some sense in business. Also this centralized control holds a view of management with militarized or bureaucratized

chains of command and control which is rejected by the modern theory on success in business. [9]

2.2.2 BPMN

The Business Process Modeling Notation (BPMN) is a graphical representation for specifying business processes in a workflow. BPMN 1.0 was adopted as an OMG standard in 2006[10]. It is based on a flowcharting technique very similar to activity diagrams from Unified Modeling Language (UML). It was developed by BPMI (Business Process Management Initiative) and it currently maintained by the Object Management Group (OMG) since the two organizaiotns merged in 2005. It consists of one diagram - called the Business Process Diagram (BPD) which is more than a flow diagram since it shows the association of data artifacts to activities as well. An example of BPD is shown in fig 1.4. The BPMN specification provides a mapping between the graphics of the notation to the underlying constructs of execution languages, particularly BPEL.[11]

The four basic categories of elements in BPD are Flow Objects, connecting Objects, Swimlanes and Artifacts. A simple business process diagram (BPD) can be made using these four categories of elements. Self-defined Flow Object or Artifact are also allowed in BPD to make the diagram more understandable.

Flow Objects consist of three core elements: Event, activity and gateway. Event is something that happens which is a trigger or a result. It could be Start, Intermediate or End. Activity shows the kind of work which must be done. It could be task or a sub-process. Gateway is used to determine different decisions or determine fork, merge and join of paths.

The Flow Objects are connected to each other with Connecting Objects. There are three kinds of Connecting Objects: Sequence Flow, Message Flow and Associations. Sequence Flow shows the order in which the activities will be performed. Message Flow tells the message flow between two process participants. Association is used to associate an Artifact, data or text to a Flow Objects.

Swimlanes organize different activities into categories of the same functionality. There are two kinds of swimlanes: Pool and Lane. Pool contains many Flow Objects, Connecting Objects and Artifacts. Lane then organize the Flow Objects, Connecting Objects and Artifacts more precisely.

Artifacts allow developers to bring some more information into the model to make the model/diagram more readable. There are three pre-defined Artifacts: Data Objects, Group and Annotation. Data Objects show the reader which data is required or produced in an activity. Group is used to group different activities but does not affect the flow in the diagram. Annotation is used to give the reader an understandable impression of the model.

The current version of BPMN is 1.1, and a major revision process for BPMN 2.0 is in progress. The new version aims to reconcile the Business Process Modeling Notation (BPMN) and the Business Process Definition Metamodel (BPDM) to specify a single language, entitled Business Process Model and Notation (BPMN 2.0), that defines the notation, meta-model and interchange format, with a modified name that preserves the "BPMN" brand.[12]

2.2.3 WPDL/XPDL and XML

The starting point for XML Process Definition Language(XPDL) was Workflow Process Definition Language(WPDL) which was created by WfMC in 1999 under the need of a standardized language to support the interchange of workflow process definitions. At the same time, XML emerged as a standard for data interchange. Since WPDL was not XML-based, the WfMC started working a new language named XML Process Definition Language (XPDL). In October 2002, the WfMC released a "Final Draft" of XPDL. [13]

XPDL is designed to exchange the process definition, both the graphics and the semantics of a workflow business process. XPDL is stated to be a textual equivalent of the Business Process Modeling Notation(BPMN).[11] The file format can be used for exchange of BPMN diagrams. XPDL contains elements to hold graphical information, such as the X and Y position of the nodes, as well as executable aspects which would be used to run a process. This distinguishes XPDL from BPEL which focuses exclusively on the executable aspects of the process. BPEL does not contain elements to represent the graphical aspects of a process diagram.

The WfMC's reference model identifies five interfaces in which Interface 1 is the link between the so-called "Process Definition Tools" and the "Enactment Service". The Process Definition Tools are used to design workflows while the Enactment Service can execute workflows. The primary goal of Interface 1 is the import and export of process definitions.

XPDL uses an XML-based syntax, specified by an XML schema. The main elements of the language are: Package, Application, Workflow- Process, Activity, Transition, Participant, DataField, and DataType. The Package element is the container holding the other elements. The Application element is used to specify the applications/tools invoked by the workflow processes defined in a package. The element WorkflowProcess is used to define workflow processes or parts of workflow processes. A WorkflowProcess is composed of elements of type Activity and Transition. The Activity element is the basic building block of a workflow process definition. Elements of type Activity are connected through elements of type Transition.The Participant element is used to specify the participants in the workflow, i.e., the entities that can execute work. Elements of type DataField and DataType are used to specify workflow relevant data. Data is used to make decisions or to refer to data outside of the

workflow, and is passed between activities and subflows.[13]

2.2.4 UML

Unified Modeling Language (UML) is a standardized object-oriented general-purpose modeling language originally in the field of software engineering,but it can also be employed to model business processes. [14]

UML has succeeded the concepts of Grady Booch's Booch method, which was better for object-oriented design(OOD), Rumbaugh's Object-modeling technique (OMT), which was better for object-oriented analysis(OOA) and Ivar Jacobson's Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modelling language. UML aims to be used with all processes, throughout the software development life cycle, and across different implementation technologies. [15].
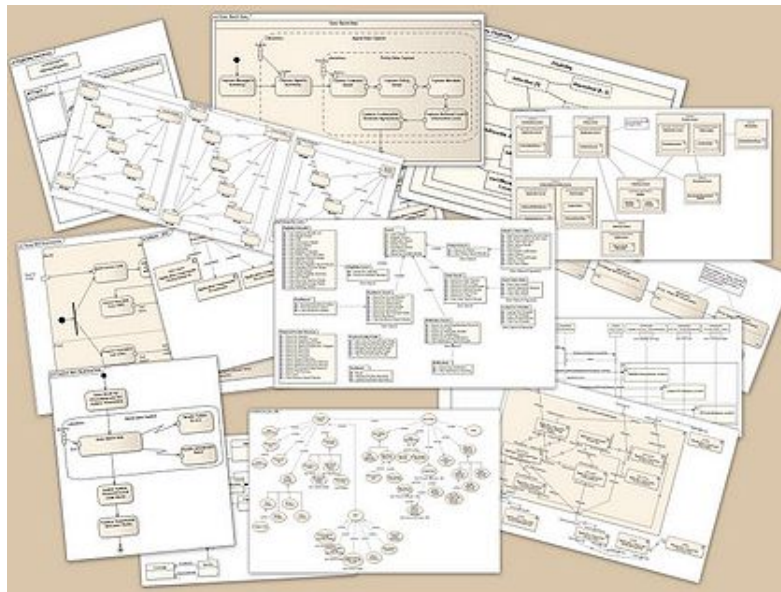
Figure 2.3: A collection of UML diagrams

UML includes a set of graphical notation techniques to create abstract models of specific systems, see fig 2.3. It contains many types of diagrams as well as documentation such as written use case that drive the model elements and diagrams. UML 2.0, the current version, has 13 types of diagrams divided into three categories[16]. These diagrams can be categorized hierarchically as shown in fig 2.4. Stereotypes were introduced into the UML in order to offer extensibility to the basic metamodel structure by the user and without actually modifying the meta model.
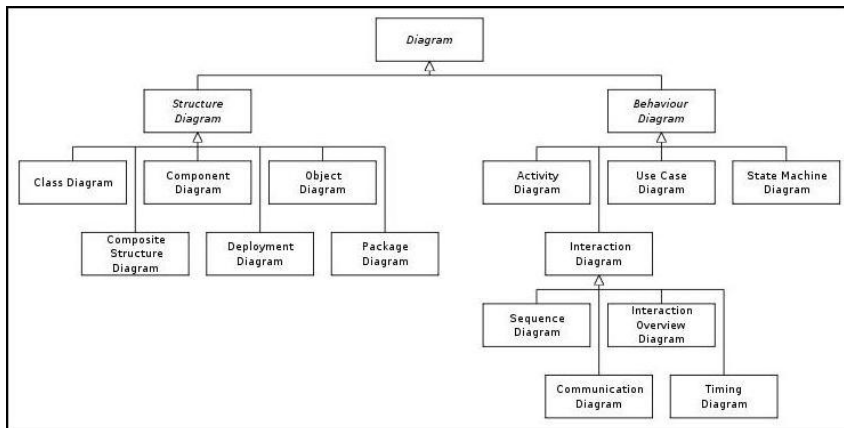
Figure 2.4: UML Class diagram

Nowadays, many new methods have been created based on UML. The best known is IBM Rational Unified Process(RUP). There are many other UML-based methods like Abstraction Method, Dynamic Systems Development Method, and others, designed to provide more specific solutions, or achieve different objectives.

The strength of UML includes that: it is a widely recognized and used modeling standard; it is a general purpose modeling language that tries to be compatible with every possible implementation language; what is more, with various types of diagram it can also give the reader different views of a system model.

However, comparing to BPMN: firstly, BPMN contains only one diagram comparing to UML with a huge set of diagrams; secondly, BPMN offers a process flow modeling technique that is more conductive to the way business analysts model; finally, BPMN is designed to map to business execution languages which means that model can be automated afterwards, whereas UML is not.

2.2.5 ITIL

The Information Technology Infrastructure Library (ITIL) is a series of documents which currently version ITILv3 comprises five volumes (Service Strategy; Service Design; Service Transition; Service Operation; and Continual Service Improvement) that are used to aid the implementation of a framework for IT Service Management, or quoting by the OGC which owns ITIL : "a consistent and comprehensive documentation of best practice for IT Service Management".

ITIL was originally created by United Kingdom's Office of Government Commerce (OGC), but is now used throughout the world. Though the cur-

rent ITIL version 3 has been published in May 2007, it is still its previous version ITILv2 gains the most distribution and popularity since the International ISO/IEC 20000 standard has emerged from the basic principles and processes coming from it.

The core modules of ITILv2 are the books entitled Service Support and Service Delivery. The Service Support processes (e.g. Incident Management, Change Management) aim at supporting day-to-day IT service operation, the Service Delivery processes (e.g. Service Level Management, Capacity Management, Financial Management) are supposed to cover IT service planning like resource and quality planning, as well as strategies for customer relationships or dealing with unpredictable situations.

ITIL uses the Deming quality circle as a model for continual quality improvement, where quality both relates to the provided IT services as well as the management processes deployed to manage these services. Continual improvement as to ITIL means to follow the method of Plan-Do-Check-Act:

- Plan: Plan the provision of high-quality IT services, set up the required management processes for the delivery and support of these services, define measurable goals and the course of action in order to fulfill them.

- Do: Put the plans into action.

- Check: Measure all relevant performance indicators, and quantify the achieved quality compared to the quality objectives. Check for potentials of improvement.

- Act: In response to the measured quality, start activities for future improvements. This step leads into the Plan phase again. [9]

The key concepts of ITIL include service and process orientation, and *service orientation* is an important model for system organization because it can encompass everything from the monolithic hierarchical systems of yesteryear to modern day peer to peer architectures which better mirror a free-market economic business interaction. It can be applied to computer-provided services (e.g. web services, or even configuration operations like cfengine) or it can be applied to human services and operations such as help desks and support. [9]

2.3 A collection of Process definitions

The term "process" became a new productivity paradigm in the 1990s. [17] A collective of process definitions are shown in table 2.1. [18][19][20][21][22]

Davenport	1993	A process is a structured, measured set of activities designed to produce a specific output for a particular customer or market. It implies a strong emphasis on how work is done within an organization, in contrast to a product focus's emphasis on what. A process is thus a specific ordering of work activities across time and space, with a beginning and an end, and clearly defined inputs and outputs: a structure for action. Taking a process approach implies adopting the customer's point of view. Processes are the structure by which an organization does what is necessary to produce value for its customers.
Hammer Champy	1993	A process is a collection of activities takes input and creates a valuable output to the customer.
Rummler Brache	1995	A business process is a series of steps designed to produce a product or service. Most processes are cross-functional, spanning the "white space" between the boxes on the organization chart. Some processes result in a product or service that is received by an organization's external customer. We call these primary processes. Other processes produce products that are invisible to the external customer but essential to the effective management of the business. We call these support processes.
Johansson	1993	A process is a set of linked activities that take an input and transform it to create an output. Ideally, the transformation that occurs in the process should add value to the input and create an output that is more useful and effective to the recipient either upstream or downstream.
Workflow Management Coalition (WfMC)	1993	The representation of a business process in a form which supports automated manipulation, such as modelling, or enactment by a workflow management system. The process definition consists of a network of activities and their relationships, criteria to indicate the start and termination of the process, and information about the individual activities, such as participants, associated IT applications and data, etc.

Table 2.1: A collection of Process Definitions

2.4 Causal Analysis

As discussed in [23] : "All human and computer systems satisfy basic laws of physics that tell us how the world works. We can not escape such laws; they bind us to basic truth about processes, even when the physics of processes seems completely buried from view." Causality is the term used to express a basic truth about the world:

For every effect, there must be one or more causes.

In other words, the effect is dependent on the cause. All of the immediate cause of a phenomenon or an event are called dependencies, that is, the event depends on them for its existence.

We often would like to be able to establish a causal connection between the result and the reason. However, determining the causes of an effect becomes increasingly difficult with the increasing complexity of the process.

Charting cause tree is a systematic method used in fault diagnosis.The idea is to begin by building lists of possible causes, then causes of those causes, and so on, until one has converted an appropriate level of detail. An example of cause tree from [23] is shown in fig 2.5.
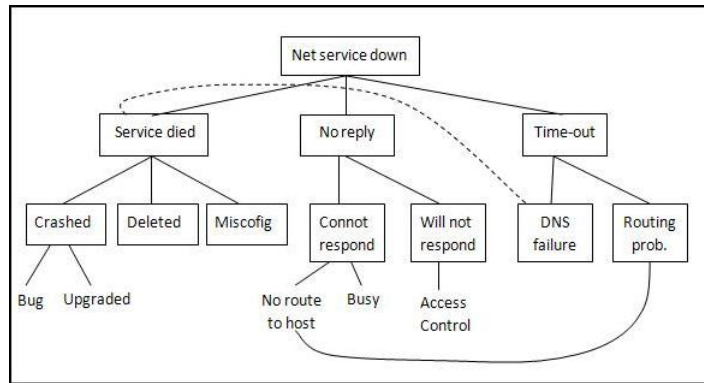
Figure 2.5: Attempt at cause tree for unavailable network service

Based on causal analysis, it can help us to find out the likely dependencies/causes of a high level goal/promise. In Promise Theory, we can make a conditional promise for each of these dependencies. A complex high level promise might have many dependencies. One of them X might be very important for the result, while Y might be not that important for the result. So we say this promise has "strong dependency" on X and "weak dependency" on Y. However, the relationship between n parts of a system are not always causal or mandatory. Promise Theory emphasizes the role of voluntary cooperation between the parts. For BPMN, the connection between the effect and cause of an activity is implicit, but always in a linear way.

2.5 Folk theorem

The only strategy certain against component failure in system is redundancy, or parallelism. Systems are a combination of serial dependencies and parallel flows. Folk theorem is:

> Low level component redundancy is never a worse strategy for reliability than high level system redundancy.

In other words, a single computer with redundant components is almost always better, and never worse than several redundant computers, see fig 2.6. [23]

This could be seen as an argument against cooperation at promise level.
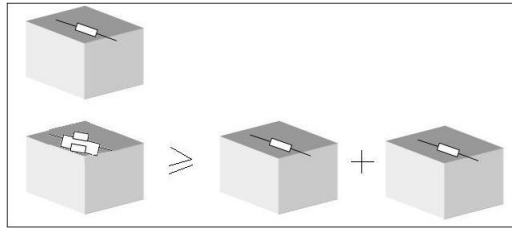
Figure 2.6: The upper architecture is parallelized in two ways below. In the left hand case redundancy is applied at the low level component layer. Disabling any single component results in the minimum disruption to the architecture. In the right hand solution, disabling the serial component from either the left or right arm results in the entire arm not working.

2.6 Cfengine

As introduced in the newest cfengine manual [24] : Cfengine is a suite of programs for integrated autonomic management of either individual or networked computers. Existed as software suite, it is published under the GNU Public License. It has been used on computing arrays of between 1 and 20,000 computers since 1993 by a wide range of organizations.

It was created by Mark Burgess in Oslo University College. The newly released Cfengine 3 in Jan 2009 has been changed to be both a more powerful tool and a much simpler tool. The main goal in changing the language is to simplify and improve the robustness and functionality without sacrificing the basic freedoms and self-repairing concepts.

Cfengine 3's new language is a direct implementation of "Promise Theory". Everything in cfengine 3 can be interpreted as a promise. Promises can be made about all kinds of different subjects, from file attributes, to the execution of commands, to access control decisions and knowledge relationships.

This simple but powerful idea allows a very practical uniformity in cfengine syntax. There is only one grammatical form for statements in the language that you need to know and it looks generically like this:

```
type:

  classes::

    "promiser" -> { "promisee1", "promisee2", ... }

        attribute_1 => value_1,
        attribute_2 => value_2,
        ...
        attribute_n => value_n;
```

It speaks of a promiser (the abstract object making the promise), the promisee

is the abstract object to whom the promise is made, and them there is a list of associations that we call the 'body' of the promise, which together with the promiser-type tells us what it is all about.

2.7 Process Management and Automation

People are often confused with the two important concepts: process management and process automation. Actually the initial focus of business process management was to automate the mechanistic business processes. Afterwards, it has been extended to integrate human-driven processes in which human interaction takes place in series or parallel with the mechanistic processes.

So when we talk about process management it is mainly referred to process modeling to to provide an architecture for all processes to be mapped; and process automation is mainly referred to software automation of the processes which are found from process modeling where there is most opportunity with the aid of computers. And the purpose of process automation system is not purely "automation to replace manual tasks", but rather "to enhance manual tasks with computer assisted automation". But process management is in itself also delays the automation of individual process and so benefit may be lost in the meantime.

In one word, process management and process automation are complementary to each other.

Chapter 3

Promise Theory

3.1 What is Promise Theory?

According to the Promise Theory webpage[25], Promise Theory is a model of voluntary cooperation between individual, autonomous actors or agents who publish their intentions to one another in the form of promises. It is a graph theoretical framework for understanding complex relationships in networks, where many constraints have to be met. It uses a constructivist approach that builds conventional management structures from graphs of interacting, autonomous agents. Promises can be asserted either from an agent to itself or from one agent to another and each promise implies a constraint on the behavior of the promising agent.

A promise is a declaration of intent whose purpose is to increase the recipient's certainty about a claim of past, present or future behaviour [26]. Uncertainty is bad since we will often stop doing something when we feel uncertain about it, e.g., in doing the final thesis, some of us just stop doing anything because they feel so uncertain about the result of the thesis and don't want to proceed any more. In the company, if the employees feel too much uncertainty then it will hinder their action and add human cost/overheads to the work. For a promise to increase certainty, the recipient needs to trust the promiser, but trust can also be built on the verification that previous promises have been kept, thus trust plays a symbiotic relationship with promises [27].

So the value of a promise are:

- It can help increasing people's belief/trust in outcome.

- An avenue to recompense.

- Allow people to plan resources and procedures.

- Provide information to the opponent, e.g. in game theory.

And the cost of a promise is the cost of keeping and the cost of auditing/verification.

Promise Theory was proposed by Mark Burgess in 2004 in order to solve insurmountable problems present in obligation based computer management schemes for Policy Based Management [28]. However its usefulness was quickly seen to go far beyond computing. The simple model of a promise used in Promise Theory can easily address matters of Economics [29] and Organization [30].

In this project we think Promise Theory might be a useful framework for process management as well, because:

1. The Promise Theory itself is very easy to understand and with fewer kinds of notations to learn which make it possible to use for a system designer, a system administrator, a business manager or even to a student.

2. It has the properties that tend to lead to a reduction of complexity. As we discussed in introduction, we can often easily view a process, subprocess or activities as a promise with the final result/output of that process, subprocess or activity as its body, and avoid mining the details inside them. This kind of blackbox approach reduce the complexity of a process model greatly.

3. It makes it possible to discuss issues like trust between parts of a system which is not representable in other languages.

4. The newly released software language cfengine3 is designed basing on promise theory, which makes the processes modeled by Promises Theory are also executable afterwards.

3.2 Promise notation

Promises are made by a promiser 'agent' to a promisee 'agent', i.e. they are directed relationships each labelled with a promise *body* which describes the substance of the promise.

A promise with body $+b$ is understood to be a declaration to "give" behaviour from one agent to another (possibly in the manner of a service), while a promise with body $-b$ is a specification of what behaviour will be received, accepted or "used" by one agent from another (see table 3.1).

A promise *valuation* $v_i \left(a_j \xrightarrow{b} a_k \right)$ is a subjective interpretation by agent a_i (in a currency of its choice) of the value of the promise in the parentheses; this can be used for ranking of importance, for example. The value can be negative if it is pure cost. Usually an agent can only evaluate promises in which it is involved.

A promise body b has a *type* which describes the nature or subject of the promise, and a *constraint* which explains what restricted subset of the total possible degrees of freedom are being promised. Since any dynamical, systematic

37

Symbol	Interpretation
$a \xrightarrow{+b} a'$	Promise with body b
$a' \xrightarrow{-b} a$	Promise to accept b
$v_a(a \xrightarrow{b} a')$	The value of promise to a
$v_{a'}(a \xrightarrow{b} a')$	The value of promise to a'

Table 3.1: Summary or promise notation

behaviour is a balance between degrees of freedom (avenues for change) and constraints, this is sufficient to describe a wide variety of phenomena.

3.3 Rules of making promise

Actually making promise is a property of all the roles/agents. One role/agent could make lots of promises, e.g., promise to feed your cat while you are away or promise to dry the sea with a bucket within one day. But not all the promises are relevant or going to be kept. These promises are like a queue buffered in the agent itself. Certain promises are triggered only when they are relevant to the process. But triggered is not equal to say that it will be put into action. So the content of a promise and whether a promise will be done is separate. Thus *promise is a combination of specification of a promise and the preparation to act a promise.* It is the policy that defines/constrains how the actions themselves should be acted.

3.3.1 Autonomy

In promise theory, promiser and promisee are considered to be autonomous agents. They can be concrete things, e.g., system administrator, servers, clients, travel agency, etc; however, they can also be abstract things, e.g., SLA (Service Level Agreement). Agents are autonomous means that one agent is only responsible for making promises about its own behaviour and he can not be forced to change behaviour by another other agents. All promises are made voluntarily. For autonomous agents it is meaningless to make promises about another's behaviour.

When A makes a promise to B, since agents are autonomous, B has no obligation to accept this promise. Thus B must make explicitly promise to receive/accept this promise which A gives to it.

Although this autonomy assumption could be interpreted morally or ethically,in promise theory this is simply a pragmatic engineering principle, which

leads to a more complete documentation of the intended roles of the actors or agents within the whole. The reason for this is that, when one is not allowed to make assumptions about other's behavior, one is forced to document every promise more completely in order to make prediction; thus it leads to a more complete documentation which in turn points out the possible failure by which cooperative behavior could fail.

The idea of autonomy is a kind of decentralization approach. A research has been done in [31] cited that the only reason why people choose centralization is to observe and distinguish system components on a single calibrated scale of measurement: the comparison of capabilities. However, a centralized hierarchy seems have heavy color of bureaucracy. While on the other hand, the autonomous structure is more than just a friendly idea but also provide a strong and reliable structure. As the paper said the promise graph is not a map of the network but an abstract set of relationships whose message passing medium is not necessarily known. Structural or organizational relationships do not have to occur through regular interaction as long as the agents can remember their promises. Once established, promises persist like intrinsic properties.

3.3.2 Conditional promise

In Promise Theory, a promise can be a conditional promise, which means that a promise is only made if certain condition is met.

For example, A1 makes a promise with body X to A2 dependent on an intermediary third agent A3. A1 would therefore make a promise to A2 saying that "I will promise you X if I get that it has been promised Y by A3 to me". To fulfil the promise, A1 also has to promise A2 that it will receive/accept/use the promise Y given by A3 to it, see fig 3.1.
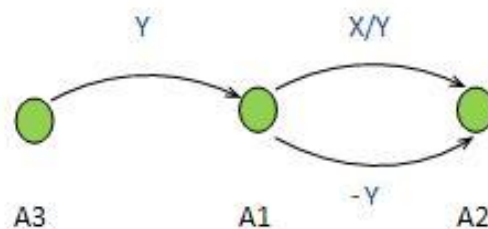

Figure 3.1: An example of conditional promise

In a conditional promise, a promise from the promiser has dependency on some certain promises to be true first. In order to keep the promise, the depended promises in back-stream should be kept first. So as we see in fig 3.2, A0 can be the manager or CEO something, A1 could be the section leader,

A2 could be the group leader and so forth. The promise *d* could be that the section leaser promises to the manager that the service they are delivered is quality approved on the condition that the promises made by group leader and the administer to him is true. Then actually we can see that in order to verify the promise,we have pushed the burden onto the end source nodes.

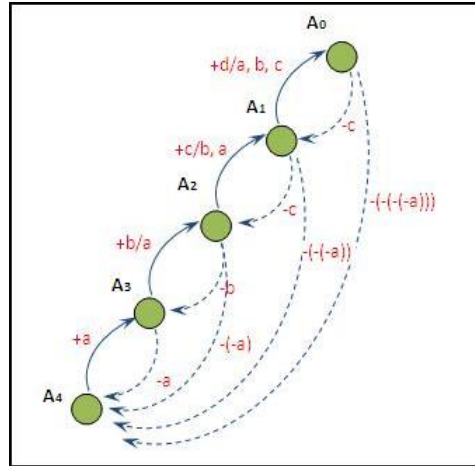Figure 3.2: Verification of a promise

3.4 How a promise is kept?

The idea of promise is to increase the trust between two agents. However, the trust is depend on the verification of the promise. We can not say surely a promise will be kept or not, but it is possible to calculate the possibility of how much percentage likely the promise will be kept. Then in promise theory the answer of how a promise is kept is not just Yes or No, but it is a probabilistic percentage.

We can view the whole process as a promise to the customer of a product or service. In order to fulfill the promises, there are a lot of smaller promises involved inside that promise.

First of all, we need to figure out the dependencies in the process. We will ask how many of the sub-promises are needed in order to trigger the higher level promise – All of them? Any of them? A certain number? Promises thus combine in ways that can be represented by simple combinatoric set notation – with 'AND' and 'OR' or other conditions. These are best known to computer scientists in the form of *logic gates*. Fig 3.3 shows the standard symbols for gates types. Although there are many gate types, for a richness of expressions, in practice 'AND' and 'OR' suffice for most cases.
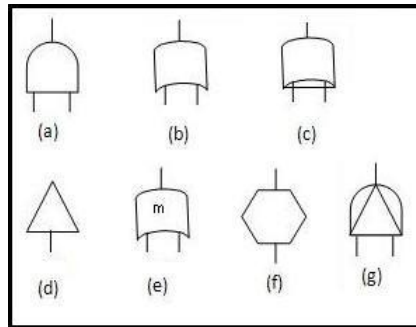
Figure 3.3: Basic gate types: (a)AND, (b)OR, (c)XOR, (d)Transfer partial result to separate sub-tree, (e)Voting gate (m of n), (f)Inhibit conditional of 'if' gate, (g)Priority AND (inputs ordered from left to right)

Secondly we can draw the process as a tree architecture similar to the cause tree in fig 2.5.

Now we are ready to calculate. We assume that every promise has a probability about how much it could be kept. The properties of the gates in combining the probabilities are noted below. Note that it makes a difference whether or not the promises are independent, in the probabilistic sense: that is, the occurrence of one promise does not alter the probability of the occurrence of another.

- In OR gates, probabilities combine so as to get larger.

$$P(A\textbf{OR}B) = P(A) + P(B) - P(A\textbf{AND}B) \tag{3.1}$$

In general,

$$P(A_1\textbf{OR}A_2\textbf{OR}\ldots A_n) = \sum_{i=1}^{n} P(A_i) - \sum_{i=1}^{n-1} n \sum_{j=i+1}^{n} P(A_i)P(A_j) + \ldots \\ + (-1)^{n+1}P(A_1)P(A_2)\ldots P(A_n) \tag{3.2}$$

- In AND gates, probabilities combines so as to get smaller:

$$P(A\textbf{AND}B) = P(A)P(B|A) \tag{3.3}$$

or in general:

$$P(A_1\textbf{AND}A_2\textbf{AND}...A_n) = \prod_{i=1}^{n} P(Ai) \tag{3.4}$$

If A and B are independent, then

$$P(A)P(B|A) = P(A)P(B) \tag{3.5}$$

41

which is smaller than P(A) or P(B); but if the promises are not independent, the result can be much greater than this.

Therefore, if we know the likely dependencies of a high level promise and the probabilities of each branch, then we can use the formula above to calculate the approximate probability of how the promise would be kept. Then the answer of how a promise is kept is not just Yes or No, but it is a probabilistic percentage.

Here is an example: assume that we are going to work out the probability of a process to successfully fulfil a Service Level Agreement (SLA) (e.g. the provided web service is running 24/7/12). We assume that this SLA depends on the web server is promised to be up all the time and the service is promised to be properly configured, see fig 3.4.
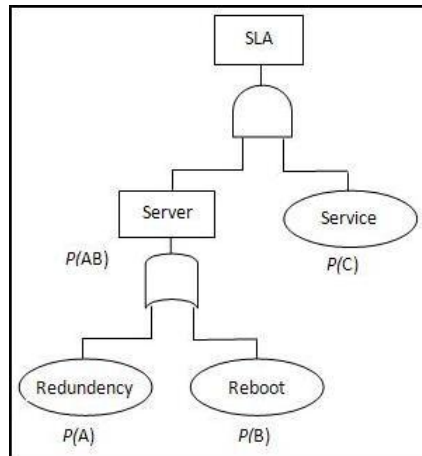

Figure 3.4: Attempt at possibility tree for keeping a SLA

We split the tree into two main branches: the web server is promised to be up all the time, 'AND' the system administrator promise to properly configure the web service.

- The two main branches are 'independent' in the probability sense, because the state of the web server does not change the configuration of a service and vice versa.

- On the server arm, we split (for convenience) this probability into two parts and say that server is keeping up if we have more than one web server as redundancy when one server is down 'OR' there is a master server in the network sends ping to the web server periodically, if it has not received the response longer than a certain time interval then it will send reboot signal to reboot the web server.

Since all the events are independent, we get:

$$P(SLAismet) = (P(A)\mathbf{OR}P(B))\mathbf{AND}P(C)$$
$$= (P(A) + P(B) - P(A\mathbf{AND}B))\mathbf{AND}P(C) \quad (3.6)$$
$$= (P(A) + P(B) - P(A)P(B))P(C)$$

Suppose we have, from experience, that

- Chance of have redundancy web server P(A) = 99/100 = 0.99

- Chance of have master machine in the network to reboot web server when server is down P(B) = 10/100 = 0.1

- Chance of the system configure the web service properly P(C) = 90/100 = 0.9

$$P(SLAismet) = (0.99 + 0.1 - 0.99 * 0.1) * 0.9$$
$$= 0.991 * 0.9$$
$$= 0.8919 \quad (3.7)$$
$$= 89.19\%$$

Important Notes:

When we assign probability to how much likely a lower-level promise is going to be kept, mainly it can come from these ways:

- Estimate experience: give a probability according to estimate experience. For instance, the probability that passengers come to the check-in desk a certain time in advance before the boarding time (40 minutes in advance for a domestic flight and two hours in advance for an international flight) is uncertain. Some passenger may come late because of the traffic jam, some may just be used to come late and etc.. The probability that the passenger would get on board on time after airport security check is also uncertain. The passenger may receive a business emergency phone call before getting on board and he must cancel this trip, some may do too much shopping and miss the boarding time. From experience, we know these two probability is high in real life but still there some exceptions. We can estimate the first probability is 97% and the latter is 99%, since in general people would more likely to successfully get on board after security check than before their check in. Of course, these are just estimated values, but they can give you an overall understanding of the probability of how a promise would be kept.

- A guess(belief): just simply guess or believe a promise would be kept with a certain probability. When we buy an laptop in a big electronic center, we would simply believe that the quality of the laptop is good (the probability that the electronic center has keep its promise to deliver a quality approved product to its customer is 100% for instance). We give this probability could be based on the reputation, credit history, first sight impression of a shop, a brand, or a person.

- Testing similar tools: test the similar tools we are going to use for many times, record how many times the promise is kept as n and how many times the promise is not kept as m, then we give the approximate probability that the promise would be kept is T, and

$$T = \frac{n}{(m+n)} \times 100\%. \qquad (3.8)$$

But what if a SLA might not depend on anything else? Can we know for certain that we can meet this SLA? Since the promise is about our own behavior, only we can made this promise and can not be changed by others. As long as we made this promise and keep it, it is possible to meet this SLA.

3.5 Model a process using Promise Theory

When we model process using promise theory, it is quite simple:

1. find out the key agents in the process;

2. figure out the promises they have to make in order to complete the process;

3. draw the promise network and promise list.

In the promise network/net, the key agents in the process are represented in the form of nodes in a promise net. The promises are represented in the form of an arc with one arrow at one end starting from a promiser to a promisee. The promise body are represented in the form of a letter together with a few words near the promise line. Details of the promise body is maintained in the promise list. Necessary annotation can be noted in the empty area in the promise net.

The output of promise modeling are two parts: a promise network/net + a list of promises. We define it in this way because it gives us a neat view of the relationship between the agents in the promise net with fewer words, then you can go into promise list to see details about what exactly needs to be promised if needed.

Some hints:

Firstly, to find out the key agents in a process is not a hard work. Mainly the people, machine or the things we care about involved in the process are the agents. **Secondly,** it requires some thinking to figure out the promises they will make. Basically we have to remember the rule for making a promise is that we assume all the agents in the process are autonomous which means that they are only responsible for their own behaviour and not allowed to force others to do anything, then it is meaningless to promise about other agents' behaviour. Also one role can make lots of promises, e.g. someone would promise to dry

the sea using his super bucket within one day, but not all the promises are relevant or going to be kept. Therefore, actually a promise is a specification plus preparation to act. **Finally,** we are ready to put our result in the form of promise net and promise list. While, sometimes you will find that the promises you thought would be made between agents are not so accurate when you are drawing the promise net or filling the promise list. Then it is time to correct it. Usually we have to modify our promise model several times to get the final version. And it just needs some exercise, when you are used to this way of thinking you will surprising find that it is not so hard to figure out the promise relationship and you can be expert in promise modeling as well.

When the promise model is done, we can easily view the agents(represented as nodes) and the activities(represented as promise arcs) involved in that process. Also you can view the sequence of the activities in terms of conditional promises, since the conditional activity of a promise should always be done first, see fig 4.3. More properties of a process can be modeled using promise theory as well. See a detailed comparison between BPMN and Promise theory in modeling the properties of a process in next chapter.

Chapter 4

Feature comparison: BPMN vs. Promise Theory

The information about BPMN we are using in this paper mainly comes from the published papers and articles about BPMN[10, 32] and the specification of the current BPMN version 1.2 which was released in January 2009[33]. Also there is a major revision process for BPMN 2.0 in progress[12], but not released yet. Here we mainly focus on the information from its former published version.

BPMN and Promise Theory are totally different animals when modeling processes. To model a process using BPMN, you model the events that occur to start a process, the processes that get performed, and the results of the process flow. On the other hand, to model a process using Promise Theory, you model the key roles in the process and the promises they will make in order to fulfill the result/goal of the process. The rule for making a promise is autonomy which means that the agents can only make promises about its own behaviour and can not force other to do anything, only if the other agent promises to accept its suggestion.

However, by applying the definitions in BPMN and Promise Theory, it is possible to find out whether there is a promise-theoretic version of each BPMN construct, and the other way around as well.

4.1 Modelling the Activities in a process

Activities are the basic construct elements of a process.

In BPMN, the activities inside a process are graphically depicted by a rounded rectangular symbol. But it can either be a sub-process or a task, since sub-process can contain task as well which can be graphically shown by another BPD connected via a hyper-link to an activity symbol. A task is the lowest level same to an activity. To tell whether it is a sub-process or a task, you just

see whether there is a '+' mark inside the rectangular which marks it as a sub-process, see table 4.1.

While on the other camp, in Promise Theory the activities inside a process are represented by promise arcs between two roles/agents. A promise is about some declared constraints to the promiser's behaviour which means what activity the promiser is going to do, e.g., I promise to feed your cat while you are away, the database server promises to do the backup every midnight or the software development team promises to work out the customer requirement analysis in one month, see table 4.1. Because of the special property of promise (a promise can also contain promise), users are also allowed to make different level abstraction of the activities.
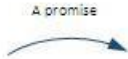
BPMN		Promise Theory	
Sub-process and task		Activities	
Arrive at Airport	A rectangular has no '+' maker in its body is considered a activity	A promise	A promise starts from a promiser to a promisee. The direction of the arrowhead is the direction of the promise. It expresses the tasks the promiser intents to do.
Check-in	A rectangular has a '+' maker in its body means it has sub-process inside and another diagram describing the sub-process is linked		

Table 4.1: Modeling Activities in a process using BPMN and Promise Theory

4.2 Modeling the Roles in a process

Since activities in the process are done by roles/agents, it is also desired to show the roles involved in the process via modeling.

When using the traditional flow-chart, it is impossible to see who does which activities clearly, see fig 2.2. Though BPMN is a flow-chart method, it is an advanced flow-chart since it introduces the concept of "pools and lanes". A pool is drawn as a rectangular region which is drawn horizontally across the diagram or vertically down it. A lane is a sub-partition within a pool and extends the entire length of the pool. Activities are put in a pool or lane, then it is possible to show who does what, see table 4.2.

In Promise Theory, it can intuitively show this process property. The roles are simply represented as nodes in the network. And the promise arcs started from it are the activities that role is going to do; the promise arcs ended to it are the activities has relationship with it. It is possible to see the relationship

between roles and activities in two ways: what activities a roles will do and in an cooperation activity which roles are involved.

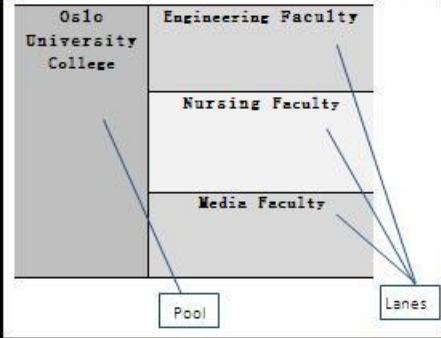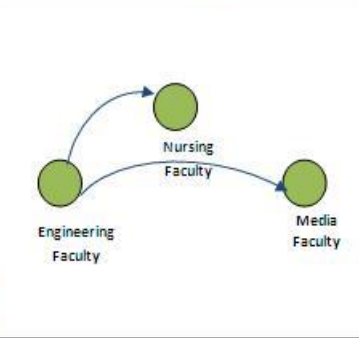

BPMN	Promise Theory
Pools and Lanes	**Promiser and promisee**

Table 4.2: Modeling Roles in a process using BPMN and Promise Theory

4.3 Modeling the Sequence of activities execution

Another important property of a process is that the activities inside it having some order/sequence to be executed, which is also the idea of most people traditionally thinking that a process is executing activities one by one in certain order.

In BPMN, it can intuitively show the sequence/order of activities by connecting two activities with a Sequence Flow which is represented as a line with a filled-in arrowhead at one end indicating the sequence of execution. This line connects the activities from beginning of the process to the end in a linear way, see table 4.3.

In Promise Theory, the sequence of activities execution is not so intuitive as BPMN. However, because there is a rule for dependency, a simple syntactic transformation can reveal the explicit sequence, assuming trust that promises will be kept. We are able to view the sequence of execution activities in terms of conditional promises. A conditional promise means the promise is made depending on something else is true. For example, role A promises X if Y is true, role B promises Z if X is true. Then we know in order to fulfill the whole promise involved by A and B, the activities reflected as promises made by A and B should be executed in this order: first Y, second X, then Z, see table 4.3.

BPMN	Promise Theory
Sequence Flow	**Conditional Promises**
	Z/X X/Y Role B — Role C Role A Y → X → Z
A Sequence Flow is a line with a filled-in arrowhead. It shows the order of the execution of steps, here the sequence is Y,X,Z.	A conditonal promise means the promise is depending on some condition to be true first. The condition is show after the slash "/" symbol. Here, role A promises X to B if Y is true, and role B promises Z to C if X is true. Then we can view the sequence of the steps in terms of confitional process, here is also Y,X,Z.

Table 4.3: Modeling the Sequence of activities execution in a processes using BPMN and Promise Theory

4.4 Modeling Decision making of the roles

In execution the activities, the roles sometimes have to ask some questions like what should I do next if there are more than one alternatives activities? Or to get one point there are a set of alternative ways which way should I pick?

BPMN and Promise Theory use different ways to model this property of the process:

In BPMN, a gateway symbol which is similar to a decision symbol in a flowchart is used to merge or fork the process flow, see table 4.4. There are eight kinds of gateway defined in BPMN[10]. It seems to be intuitive since it seems all the alternatives are listed there. However,the premise of this is that the user knows all the alternative answers in advance. Is it possible in real life? The answer is absolutely No, since there is always uncertainty in the process. In fact, the alternatives listed in the gateway is just a subset of all the alternatives. We have to say that BPMN has a good wish to model the decision making of roles, but it is unavoidable constrained by the modeling method itself which is trying to list everything in detail.

While in Promise Theory, instead of trying to list every alternative in detail, it uses promises to indicate the decision of a role, since promises are the written intents of the role. What makes it different is that, a promise defines the

final result/goal an activities will get after finishing. For example, A promises B to take care of his cat while B is away. But maybe A just gives breakfast to the cat, or maybe three meals, or maybe just water, but no matter how he executes it, as long as he can reach the result that the cat is feed is fine. It is vividly illustrated in fig 1.5.
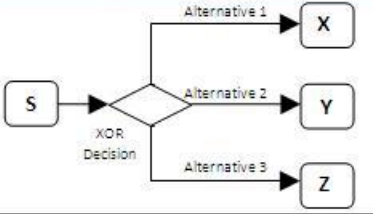
BPMN	Promise Theory
Decision Gateways	**Promise is a decision**
Alternative 1 → X S → XOR Decision → Alternative 2 → Y Alternative 3 → Z	I will feed your cat while you are away. Role A → Role B
There are many types of gateways defined in BPMN. Above is the most common used **Data-based XOR decisions**. A data token traverses the Process Flow and arrives at the XOR gateway. The path that it flows out on is chosen based on condition expression for each gate of the gateway. It can only go out on one flow.	A promise is the written intents of a role in the process, then it is a **decision itself**. The promsie defines the final result/goal the role will get after finishing the activity. Then maybe he will only give breakfast to the cat or give three meals or just water, no matter what he did as long as he can reach the result that's fine.

Table 4.4: Modeling Decisions in BPMN and Promise Theory

4.5 Modeling Cooperation between roles in a process

In a process when there are more than one role involved, in order to effectively and efficiently working, we emphasize that the roles need to cooperate with each other. The cooperation can be in many forms.

4.5.1 Modeling the Message between roles in a process

One form can be transiting message between each other. They send message to each other to communicate.

In BPMN, it augments one annotation called Message Flow to the traditional flow-chart which is impossible to model the message between roles originally. The Message Flow is represented with a dotted line with a filled-in arrow at one end. It starts from the activities,events or gateways in one pool/lane to another, see table 4.5. The concept of Event is BPMN is also a kind of message representing what's the state when a process starts, in the middle and finishes. Each of them has distinct notation, see fig 4.1.

On the other hand, in Promise Theory, the promises made between roles is a neat and nice form of message transiting between each other. No much
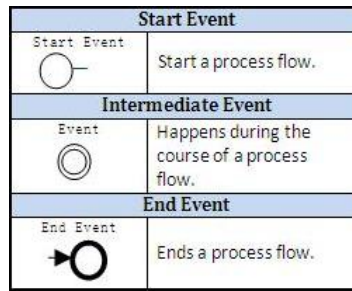
Figure 4.1: Events in BPMN

explanation is needed.
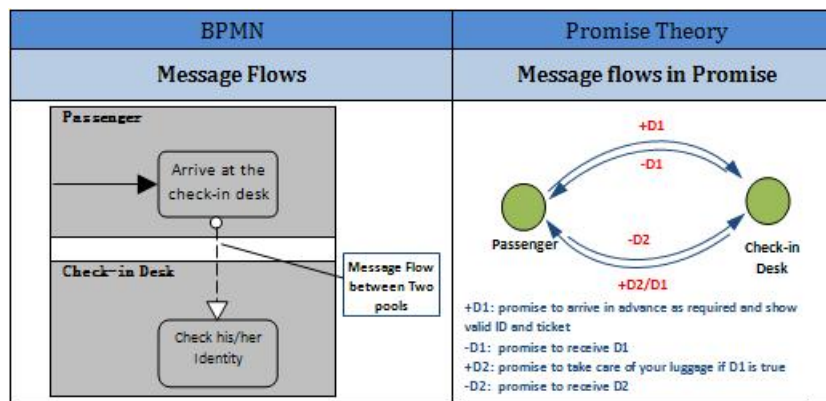

Table 4.5: Modeling Message between roles in a process using BPMN and Promise Theory

4.5.2 Modeling the Trust between roles in a process

Another form can be trusting each other. Trust has its economic value since a person maybe doesn't trust another person who sends the message from. Then because of the uncertainty, that person may tend to delay his action to wait until he is certain about the result. The wait causes delay of time and time is equal to money in a company if the company have to pay their employee according to working hours.

Promise Theory can efficiently model the trust between each other since roles are sending promises to each other in the process. And the purpose of promise is to increase role's certainty/trust of the result. On the other hand, BPMN seems to have no chance to model this delicate cooperation between roles.

4.6 Modeling Black-box of a process

There are times when you are modeling that you do not care how a process is performed in a company. It may be another company or a customer that is outside your scope; you have no control over it. When we buy a television, we only want a television instead of knowing how every piece such like CRT and transistors inside the television look like. You do not care how the company creates a message; you only care that the message has been delivered to you and contains information that you can use. Or you don't care what a company does with message that you deliver to it —- you trust that it does right thing with it, maybe because the company or the shop has a good credit and reputation before, or it is simply just a guess (belief).

Therefore, nowadays the "black-box" package opinion is very popular in datacenters. It would be nice to hide appropriate amount of details in the black-box or package and only present the user with necessary information so that he can easily understand and operate with. In contrast, Micro management is to manage with great or excessive control, or attention to details, which can always be abused and overload the user with too much detailed information.

When modeling using BPMN, you can treat the company (or application, function, and so forth) as a "black-box" – only draw Message Flow to or from the pool representing it, and not show any details inside the pool, see table 4.6.

On the other hand, Promise Theory is in itself a kind of black-box approach. The whole process can be expressed as a high level promise. This promise might contain many lower level promises which also might contain promises. What presented to the users is usually the high level promise. For example, in the check-in procedure, in a promise net you can make the abstract to only model the promises between passenger, check-in desk, security control and the boarding gate, which is a nice scope for a passenger. But actually there are many much more lower level promises have to be made in order to make it work, e.g., the computer should promise to the people working in the check-in desk that it contains the necessary correct information about the flight and the customer. Therefore, the person doing the modeling can control to which level he wants to make the abstract to give proper information to the end users about the process.

4.7 Annotation

There is a proverb saying that "a picture is worth a thousand words". However, conversely, sometimes a picture is not enough – we need words to describe the nuances of something that a picture can not do justice to.

Table 4.6: Black box and White box in BPMN and Promise Theory

In BPMN, Text Annotations is used to provide a textual annotation that can be affixed to any model element. It is displayed within an open rectangle, attached to the symbol by a straight line, see table 4.7.

In Promise Theory, there are two parts of the modeling: one part is a promise net which gives you a neat brief view of the process; the other part is a list of detailed promises which are made by the roles in the promise net, see table 4.7.

Table 4.7: Annotation in BPMN and Promise Theory

From above sections, we have seen most of the symbols used in BPMN and Promise Theory. We can see that BPMN do have a complex of symbols system which includes around 20 different symbols. While the symbol system

in Promise Theory is quite simple, just a node representing agents/roles and an arc representing promises. The advantage of a complex symbol system is that it gives user more intuitive and rich view of a process than fewer kinds of symbols. However, the disadvantage is its usability and learnability reduce at the same time.

4.8   Mapping to Execution Language

As we introduced in section 2.2.2, BPMN was developed with a solid mathematical foundation – the PI-Calculus branch of Process Calculi has been used. This is a formal method of computation that forms the foundation for dynamic and mobile processes. It means that business processes designed using the BPMN standard can be directly mapped a process modeling executable languages for immediate execution, particularly BPEL. See an introduction of BPEL in section 2.2.1. This is analogous to the functionality of relational data models and the generation of SQL/DDL statements.

On the other hand, the newly released datacenter configuration and automation language cfengine3 this January was developed based on Promise Theory. Cfengine 3 is both a more powerful and much simplified version of cfengine2 which has been widely used in a lot of middle to large size datacenters. When you make a promise, you express the confidence to deliver. Cfengine 3 has been redesigned to place promises rather than procedures in the driving seat of IT management, so that everyone from technical engineers to management can understand the real intentions behind datacenter choices. Cfengine 3 is not just a face-lift but a commitment to the next generation intelligent revolution for datacenters, a future where mobility and diversity mesh seamlessly into reliable infrastructure.[34]

Chapter 5

Scenarios Modeling

To start with, we can review a little about the method BPMN and promise theory used to model a process. BPMN models a process using a Business Process Diagram(BPD). In BPD, it models the process in the form of a flow. It starts by modeling the events that occur to start a process, the processes that get performed, and the end results of the process flow. Decisions and branching of the flows is modeled using gateways.

When modeling using Promise Theory, it is totally different from the traditional approach which models the process in form of a flow, but in the form of a promise network/net together with a list of the promises the agents has to make in the process. The process is quite simple. It first finds out the key agents in a process, then figure out the promises they have to make in order to complete the process. It can also be a conditional promise which means someone promises something if something is true.

5.1 Scenario 1: Provide remote access to the system for venders

5.1.1 Two situations

In this scenario, the vendor of ABC company's service platform needs remote access (e.g. ssh) to the system for testing purpose during a project period. Usually it needs three phases: application, authentication and implementation. We assume two different situations in this process.

In the first situation, it is a fairly small company and for this remote access provision all the work is going to be done by the only system admin in this company.

In the second situation, it is a big company and has sound policy for how to provide remote access to the system. It is specified in this way:

1. firstly the vender should deliver a standard fill-in application form and
   sign the "promise of secrecy" to the system administrator in that service
   group;

2. when the application is delivered,the system administrator in that ser-
   vice group should deliver it to the section leader to ask for approval;

3. the section leader should make a decision to approve it or not according
   to certain policy and write the decision back to that system admin;

4. if not approved, that service system administrator should write back to
   the vender to tell the result and explain the reason;

5. if approved, the service system administrator should send an order to the
   system administrator in network group to ask for network information
   for the vender to access to the company's network, such as username,
   password, access server name, ip addresses, port forwarding informa-
   tion and etc.;

6. when receive the order, the network administrator should create user on
   the related servers and send necessary logon info directly to the vender,
   after that write confirmation to that service system admin;

7. when receive confirmation from the network administrator, the service
   system administrator should create a user on the service platform and
   send logon information back to the vender when complete.
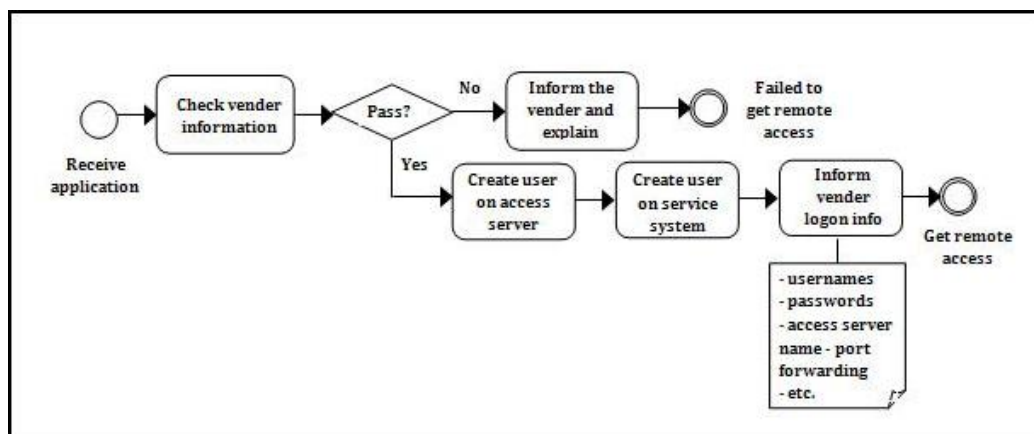
5.1.2 Results

Figure 5.1: BPD of providing remote access to the system for venders in a small
company

In fig 5.1, the process is modeled using BPMN. Since all the work is going
to be done by the only system admin, we didn't use pool and lane in the BPD.

The process starts from the event that the system admin receives the remote access application from the vender, then he processes the application step by step as the graph clearly shows: three steps to get to the result of fail and five steps to get to the result of providing the vender remote access successfully. What makes it different from a data diagram it that we use a data object to show which data is produced in the activity of informing vender about the server logon information.
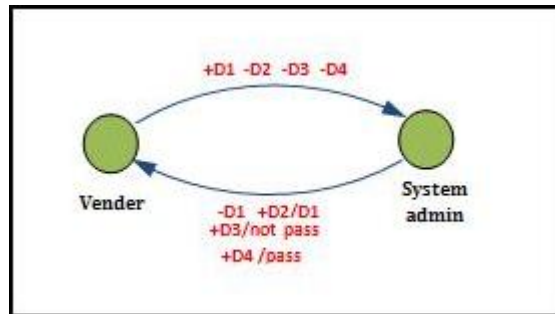

Figure 5.2: View the process of providing remote access to the system for venders in a small company using promise theory

Table 5.1: Promise list of providing remote access in a small company

In fig 5.2, the same process is modeled using promise theory. We see in this process both the vender and the system admin have to make four promises about their own behavior. A list of what promises they should make in listed in table 5.1. It is mainly the system administrator who has to make most of the promises since he is going to provide the service and the vender is mainly to accept the promises that the system admin makes to him. For the three promises that the system admin has to make, we can see they are all conditional promises which means that the conditional things should be met first.

In fig 5.3, the second situation of the process is modeled using BPMN. The process also starts from the event that the system admin receives the remote access application from the vender and ends with the results that the vender gets the remote access successfully or does not. But in this process, more peo-
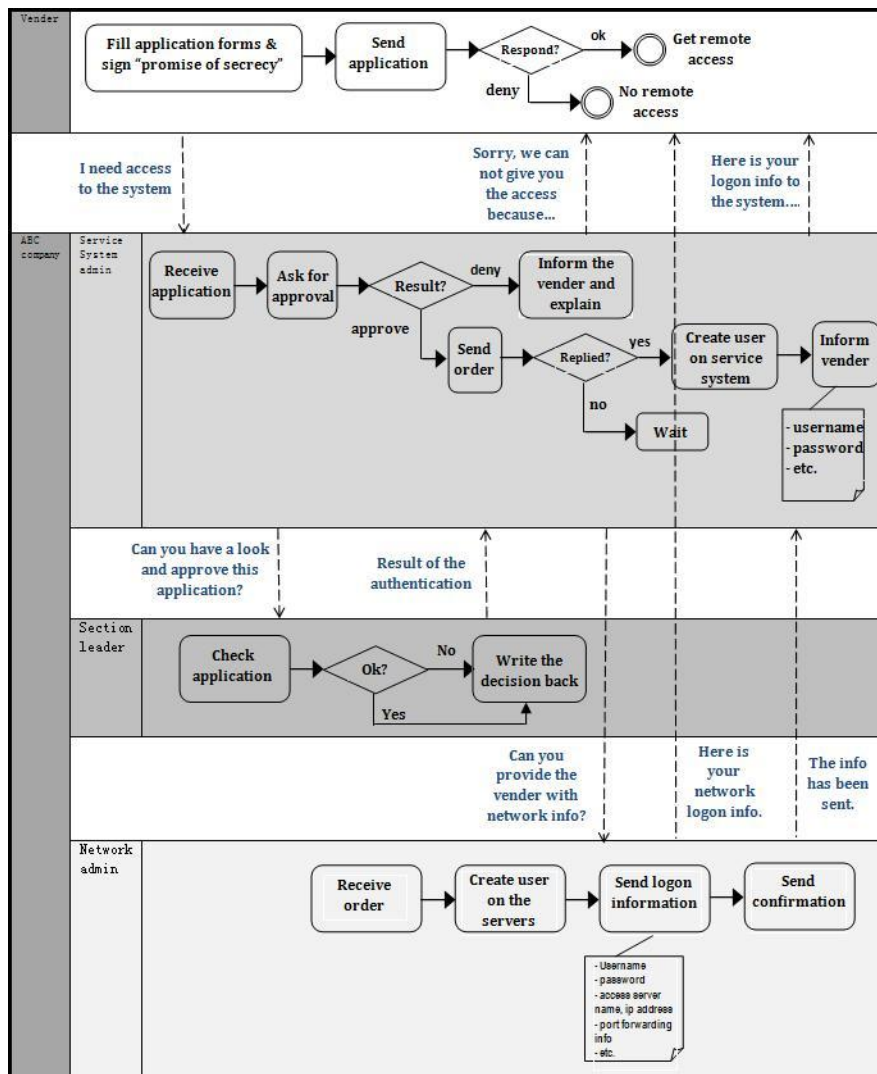
Figure 5.3: BPD of providing remote access in a big company

ple are involved, apart from describing the order of the activities, it is also interesting to know who does what, further the responsibility of each participants. We used pool and lanes in this model. The ABC company is a pool and has three different lanes resided with three different key roles in this process. The activities they are going to do is located inside each lane. What's more, we can see there are more people involved in this process which means that there are a lot of interaction/cooperation between them. For example, the service system admin will not send order to the network admin if the section leader doesn't approve the application. We used the message flow in this model to show the messages sent between the agents which is in some sense the interaction between the agents. We can see several dotted lines flying above the lanes in the model. They are the message flows.
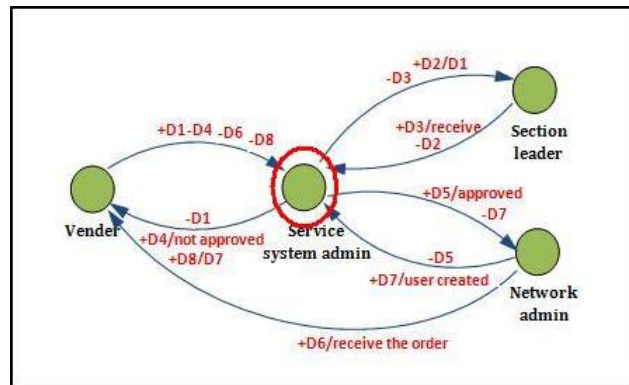
Figure 5.4: View the process of providing remote access in a big company using Promise Theory

Table 5.2: Promise list of providing remote access in a big company

In fig 5.4, the process in the second more complicated situation is modeled using Promise Theory. This gives a more correct picture of the topology of the system and allows us to identify a critical (important) agent more easily. A list of what promises the key agents should make are shown in table 5.2.

5.1.3 Analysis

Comparing the each two pictures for the each situation, though they look totally different. However, both pictures can provide similar information such like the activities and the roles in the process. What's more, both can also show the responsibility of each role – labor division. For BPMN, a role is responsible for the activities inside its lane or pool. For Promise Theory, a role is responsible for the activities in the promises which started from it.

For the sequence of the activities, BPD gives intuitive view. In contrast,

it needs some thinking to figure out in terms of conditional promises in a Promise network.

However, when we model the cooperation in terms of messages in the second situation in which more roles involved, we found that the Promise network still keeps the neat way as its first model. In contrast, the message flow lines are vertically crossing above the sequence flow lines which looks a bit messy.

Trust

We can see the trust between the agents in Promise network via the promises made between two roles. For example, the network system admin promise to create a user on the vender server and send the logon information to the vender when he receives order from the service system admin. we can see that he trust the service admin when he ask him to provide sensitive network information to the vender. But we can not see the similar information in a BPD.

Expose Uncertainty

Another problem we found in a BPD is that: a model like a flow takes it for granted that the activities will be executed exactly in this way as shown in the flow from one step to another step. But in reality some steps could fail because of unpredictable reasons. We should admit that uncertainty exists all the time in the processes. We don't like uncertainty,but being aware of the uncertainty is not a bad thing. It can greatly help us in analyzing the failure mode and doing risk assessment which means money in economic related processes.

The predictable failure modes can be modeled using BPMN via gateway. For example, the vender can not get the remote access because the application is not approved by the section leader. But when the reason is unknown, BPD just omit them since it doesn't know.

On the other hand, the model of Promise Theory is not event based. The autonomous roles in the process make cooperative promises to each other about the final state they intent to achieve. We assume that the roles know what to do in order to keep the promise and we don't need bothering to worry about it. Of course, we have to be clearly aware that it has the chances that the promises are not to be kept. Then actually the uncertainty is fully exposed in this way.

Convergency

What's more, because a promise is about the final result/goal the role is going to get when finishing the activities, it makes the activity/process convergent. See an illustration of convergence in fig 1.5. While in BPMN it defines where a process start and each steps, it makes the process like the left picture of in fig 1.5. From fig 5.3 we can see that if any link of the entire process is

broken then the whole process is stopped and broken. The model is very fragile. But if the steps inside the black-box are convergent, no matter which link is broken, we can always get to the desired state when after finishing the last step in the black-box. The entire process is "fault tolerant" in this way.
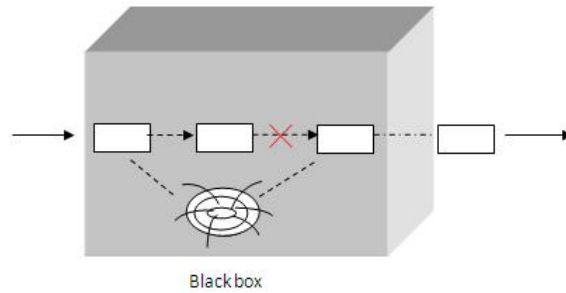
Table 5.3: A fault tolerant process

5.2   Scenario 2: Make an orchestra performance

An orchestra is an instrumental ensemble, usually with string, brass, woodwind sections and possibly a percussion section as well. Parts of a beautiful orchestra music score is shown in fig 5.5. Different instrument or section plays different rows in it. In an orchestra, the conductor see all parts of the score, but the players only see their own parts.
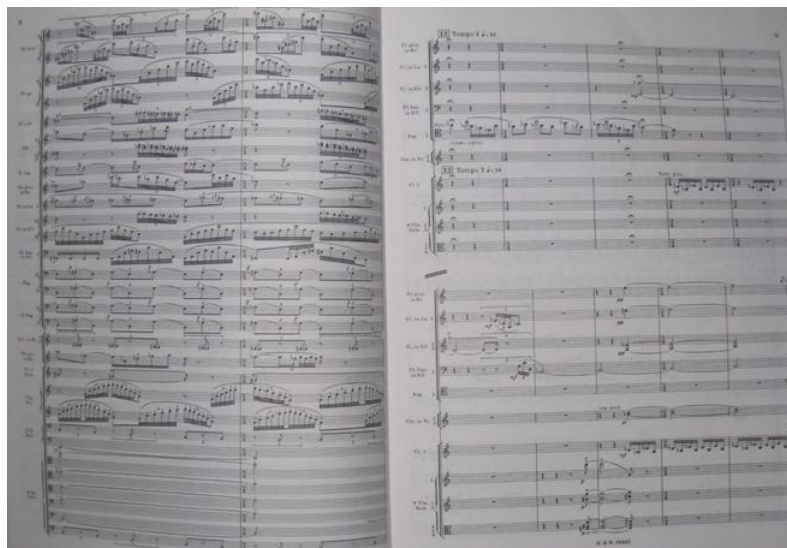
Figure 5.5: Part of an orchestra music score

In this scenario, to make it easy we assume that there are three instruments performing in the orchestra. They are violin, piano and flute.

5.2.1 Result

A BPD of this process is shown in fig 5.6.
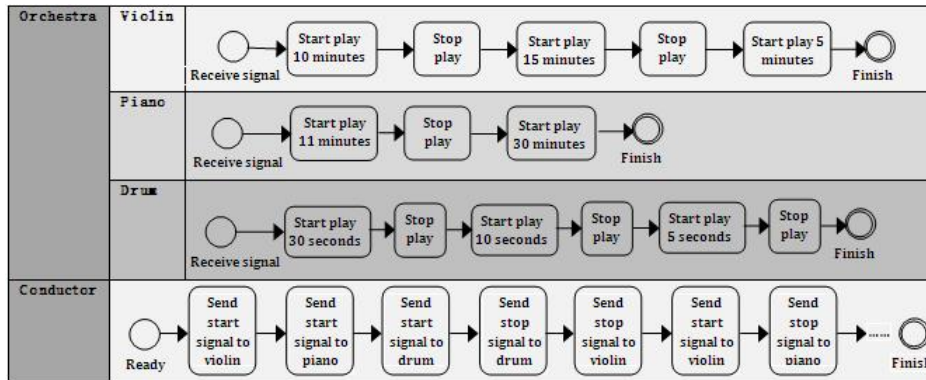
Figure 5.6: BPD of making an orchestra performance
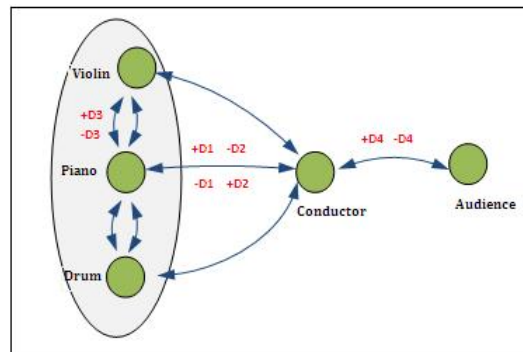
Figure 5.7: View the process of making an orchestra performance in Promise Theory

The Promise Theory model result is shown in fig 5.7 and a list of the promises made by the roles in table 5.4.

5.2.2 Analysis

From fig 5.6 we can see that different instrument has their own activities in their own lanes – labor division. And the lanes look quite like the rows in the music scores. However, we can not see the peer-2-peer cooperation from it, not via a conductor. On the other hand, in fig 5.7, we can intuitively see the resposibility of each player and the communication between the player and

> **+D1:** promise to play in time and in tune and follow the signal from inductor.
> **+D2:** promise to give the correct signal according to the score to the players.
> **+D3:** promise to hear the others' playing.
> **+D4:** promise a good orchestra.
> **−D1∞−D4:** promise to receive/accept corresponding promises from D1 to D4

Table 5.4: Promise list of making an orchestra performance

the conductor, also the communication between the players because they can all hear the music.

Orchestra mitigates the central service needed from conductor by giving each player a script – so that they can be almost autonomous. The players just needs a few signals such like start and stop from the conductor in performing. Many technologies have misunderstood orchestration, (e.g. SNMP) by making their conductor paly every instrument itself.

5.3 Scenario 4: Numerical weather prediction in a weather center

As the fast development of computers, practical use of numerical weather prediction began in 1955 which made the weather forecasting come into a new age. Numerical weather prediction models are computer simulations of the atmosphere. It uses the meteorological data, such as temperature, pressure, humidity, rain, wind and etc. collected from tens of meteorological observation stations, which are distributed in an area, to analysis and evolve the state of the atmosphere forward in time using complicated equations in physics and fluid dynamics.

In this scenario, the weather center in city ABC has set up 20 meteorological observation stations in this area. These stations collect the meteorological data of its place and send the data back to center all at certain same time everyday. There are hundreds of computers in the weather center processing and calculating the collected weather information. It is a "cluster". In the "cluster architecture", there is one master control all the other slave computers. The master just sends some control signals to the slaves, e.g. update, restart, etc. All the slaves play identical roles in processing and calculating the collected weather data. The output from the weather computing provides the basis of the weather forecast to the users.

5.3.1 Result

Fig 5.8 gives the BPD of this process.

Figure 5.8: BPD of the Weather forecasting in a date center
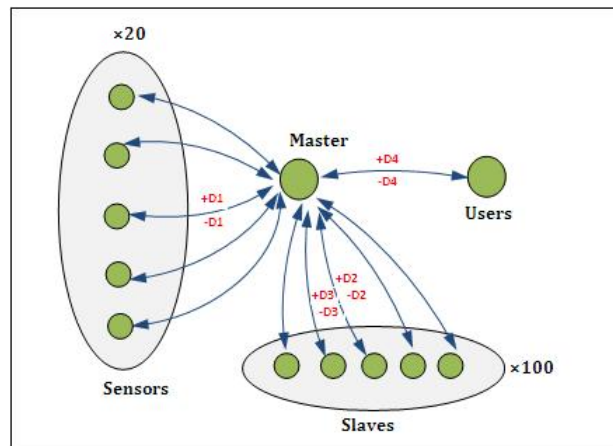
Figure 5.9: View the process of Weather forecasting in Promise Theory

The modeling result of Promise Theory is shown in fig 5.9 and a list of the promises is in table 5.5.

5.4 Scenario 5: System admin team work

In this scenario, there are more than one system administration teams inside a big company which provides services to thousands of clients. There are teams in charge of database, network, application and etc. The process is to deliver a good service to the clients to fulfil the SLA (Service Level Agreement), a contract between the client and the service provider.

5.4.1 Result

Fig 5.10 is the BPMN modeling result.

See the promise theory modeling result in fig 5.11 and table 5.6.

+**D1:** *promise to collect the local meteorological data such as temperature, pressure, humidity, rain and wind and transmit those data to Weather Information Center the same time every day.*
+**D2:** *promise to process and calculate the collected weather data and receive signal from master.*
+**D3:** *promise control all the slaves and send signals such as update and restart to slaves when needed.*
+**D4:** *promise to provide the users with future weather forecasting*
−**D1**∞−**D4:** *promise to receive/accept corresponding promises from D1 to D4*

Table 5.5: Promise list of Weather forecasting
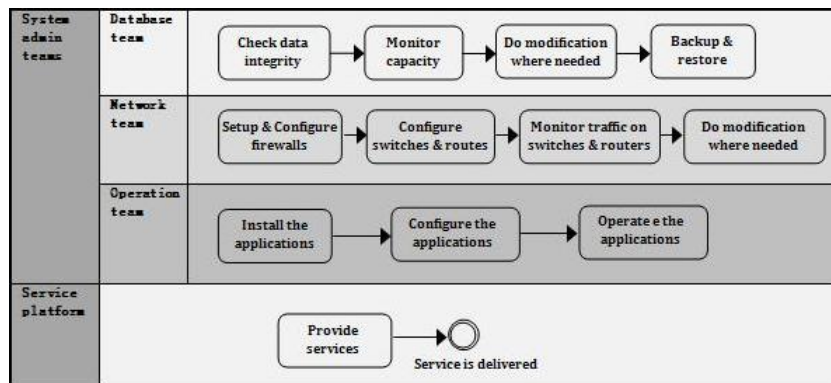
Figure 5.10: A BPD of System admin team work

5.4.2 Analysis

The process in scenario 2,3,4 are different processes. In the modeled BPD diagram, we can see the different activities and orders. Basically there is no similar structure between these three BPDs.

However, we can find a similar architecture of these processes from the Promise Theory modeling, see fig 5.12. The value of this is that when we there are two processes having similar architecture. One of them we are familiar with, while the other not. Then you can apply the experience from your familiar process to your unfamiliar process. It helps to increase learnability of some unfamiliar processes.

5.5 Scenario 5: Check-in procedures for a flight

In this scenario, Oslo Gardermoen Airport is the main international airport serving Norway, with flights to a large number of European airports and other continents. More than 19 million passengers travelled through this airport in 2007. Mikal is working in one college in oslo and plans to take a holiday to
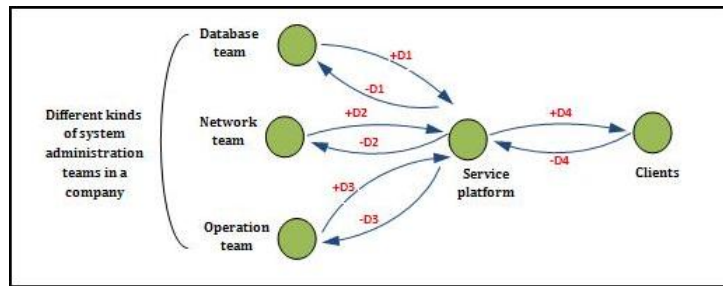
Figure 5.11: View the System admin team work in Promise Theory

Table 5.6: Promise list of System admin team work

San Diego next week departure from this airport. He has already bought the air ticket on-line.

5.5.1 Result

Fig 5.13 gives a BPD of this process. In this process,the start event is that Mikal has bought the air ticket. Then he comes to the airport and go through the check-in desk, security control and boarding gate in order. The end result of the process is either he gets on board successfully or not. The main agents in the process are Mikal, check-in desk, security control and boarding gate.

The result of the Promise Theory modeling is shown in fig 5.14 and a list of the promises in table 5.7.

5.5.2 Analysis

Find the key point

In this scenario, the process represented in BPD is in a leaner way, so it's difficult to tell which activity or agents is the most important in the process. While in promise net we can find the key agent in the process by counting the promise number that he has to make. We can easily see from fig 5.18 that the passenger has to make 6 promises, the check-in desk, security control and

Figure 5.12: Similar architecture of scenario 2,3 and 4
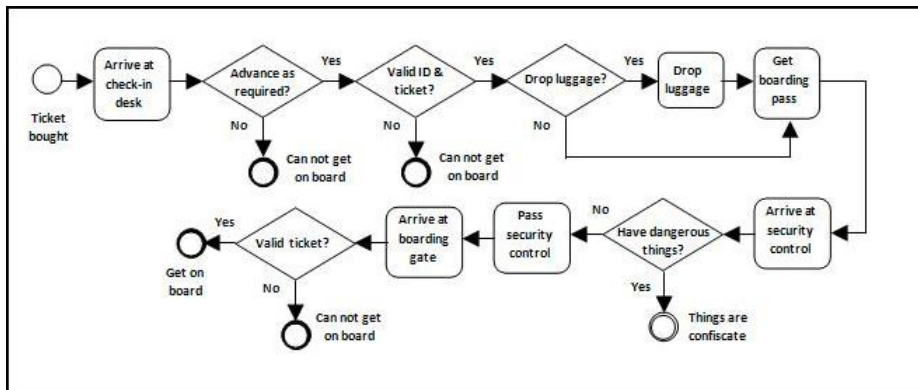
Figure 5.13: BPD for check-in procedures for a flight

boarding gate all have to make two promises. Then we know that the passenger is the key agent in this process. Whether this process can be completed successfully is largely depend on the passenger's behaviour.

Information compression

Comparing this scenario with the weather center scenario,we can find that: in weather center scenario there are many identical parts in the system (the slaves play identical roles in a cluster); while in this scenario there is a large degree of variation between the parts of the system (passenger, check-in desk, security control and boarding gate all play different roles) which can be compared to the workstation in multiple departments playing different roles.

According to the Information Theory – many same symbols can be compressed. [23] The promise list in the weather center scenario can be compressed, because they are many identical symbols/promises. But the process in this scenario can not be compressed because there are all different sym-

Figure 5.14: Promise net for check-in procedures for a flight

Table 5.7: Promise list of check-in procedures for a flight

bols/promises. For the two BPDs in these two scenario, neither can be compressed, because they are all different symbols.

5.6 Scenario 6: How Google processes requests

The recent report on Nielsen Online shows that Google has processed about 6,100,000,000 search requests in March this year and has occupied 64.2% of the total U.S. search market, an increase of 27.6% comparing to last year. It is interesting to see how Google processes the search request. As Professor Alva Couch pointed out that the request processing in Google is different from RPC (Remote Procedure Call). The process of RPC is synchronous:

1. X calls Y.

2. X waits.

3. Y responds.

4. X continues.

Instead, each Google request is broadcast to 500 servers that "might" have an answer. Then the servers act autonomously, and the first one that responds cancels all of the other requests. The process is asynchronous:

1. The search request in the form of list of words a client types in are sent to the Google frontend server in CGI format.

2. The frontend server UDP broadcasts the search request to 500 backend servers that "might have a answer" in an unreliable way.

3. One backend server that has the answer responds and sends the raw response using focused UDP to the frontend server.

4. The frontend server cancels the search request via UDP broadcast to the rest 499 backend servers.

5. The backend server that has the answer sends the formatted raw data to the frontend.

Note that the request entities are queued in the servers and the servers do not respond in FIFO (first in first out) mode; they randomize their choices of what to choose next to respond to, and if more than a second of real time elapses, they discard the queue entries and don't respond at all!

5.6.1 Result

When we model this process using BPMN and promise theory, we get two different results as shown in fig 5.15, fig 5.16 and table 5.8.

5.6.2 Analysis

Autonomous behavior/asynchronous processes

The goal of this scenario is not to verify the technique details of how Google processes the search request. Since lack of formal reference, detail of the description of the the process maybe wrong. But the scenario itself is a classic case of application of asynchrony and autonomy.

There could be two kinds of asynchronization: one is that not waiting for the response before continuing, the other one is not asking for something from someone specific. A phone call to another person is a synchronous process - it can't go forward if the person you want to talk to doesn't answer the phone. Leaving a message on an answering machine turns it into an asynchronous
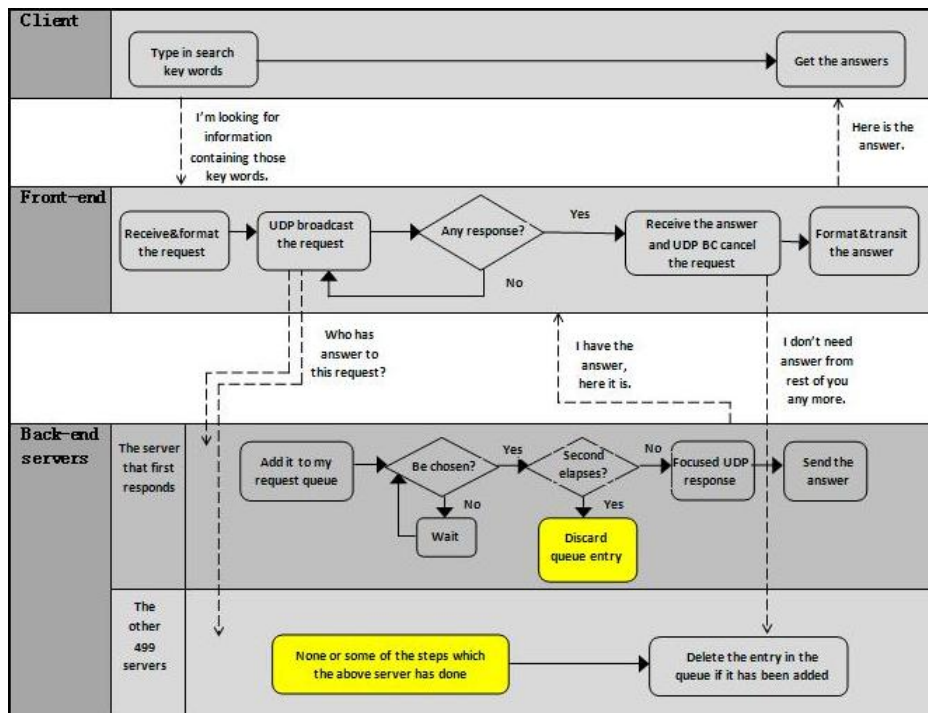
Figure 5.15: BPD of how Google processes requests

process. You leave your message and go on with your business, figuring the person will respond when they get the message.

We can find these two kinds of asynchronization in the last scenario about how Google processes search request. When the frontend server send the request to the backend servers, it doesn't know which specific one will have the answer but still it just broadcasts the request to all the backend servers. It doesn't wait this response before it goes on with sending other request. Also the backend servers' behaviour is quite autonomous. It is not interfered by anyone. When see the request, it adds the request in its request queue and randomize the choice of which request to choose to process next. When the time expired, it just drop the whole queue and doesn't say anything to the frontend server.

It is hard to read this kind of information in a BPD. As fig 5.19 shows, there are two blind yellow areas when we are trying to model this process using BPMN. First for the rest of the 499 servers who didn't respond, they have to delete the entry which they may have already added in their request queue and maybe have already deleted it. But we can not model what the servers have done before it deleted the entry since we don't know. What's more, when the request is chosen and time expired, the server just drop the entry and doesn't respond at all. But what if at that time it has just responded the frontend server that it has the answer, then the frontend server has to wait
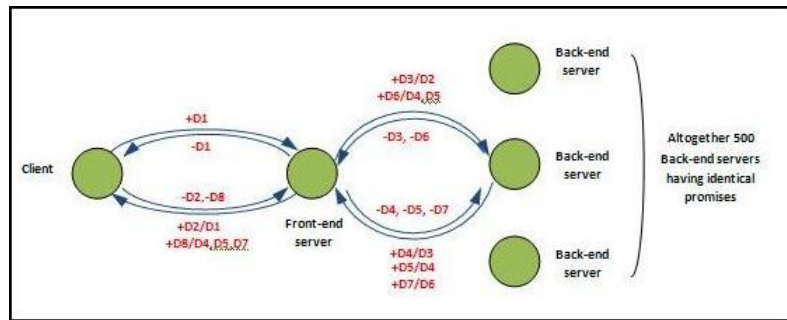
Figure 5.16: View how Google processes requests in Promise Theory

+D1: promise to type in the search key words and send the request.
+D2: promise to receive and format the request if the client sends it.
+D3: promise to UDP broadcast the request to all the 500 back-end servers if I have received and formatted it.
+D4: promise to put the request entry in my request queue and randomize my choice of what to choose next to respond to if I receive any request.
+D5: promise to send the raw answer to front-end via focused UDP if the entry is chosen and before a second of real time elapses, otherwise discard the queue entries and don't respond at all.
+D6: promise to UDP broadcast to cancel the request to the rest of the back-end servers if I receive the first backend response, otherwise broadcast the request again if no response within a second.
+D7: promise to delete the entry in my queue if I receive the cancel signal from front-end.
+D8: promise to format the raw answer and transit it to the client if I receive the answer
−D1∼−D8: promise to receive/accept corresponding promises from D1 to D8

Table 5.8: Promise list of how Google processes requests

and has no clue about what has happened at all.

On the other camp, Promise Theory can model this process without diffi-
cultly. We can describe the autonomous behaviour of each agent in the form
of promises which made by the agents themselves, no matter synchronous or
asynchronous.

```
In one word,
-Promise Theory can not only represent all the entities, it can also
show non-layered architecture and the non-linear behavior.
-BPMN tends to make everything linear, e.g., straight line flow or
layer upon layer.
-The representation of asynchrony is automatic in a promise. Synchrony
is always represented by a conditional promise.
```

Chapter 6

Discussion

6.1 A check-list of the feature comparison

Table 6.1 shows the result of the comparison of process feature modeling using BPMN and Promise Theory.

Criterion	BPMN	Promise Theory	References
Can it model the activities in a process?	√	√	Section 4.1, Section 5.1.3
Can it model the roles in a process?	√	√	Section 4.2, Section 5.1.3
Can it model the execution sequence of activities?	√	(√)	Section 4.3, Section 5.1.3
Can it model the roles' decision making?	(√)	√	Section 4.4, Section 5.1.3
Can it model the messages between roles?	(√)	√	Section 4.5.1, Section 5.1.3, Section 5.2.2
Can it model the trust between roles?	×	√	Section 4.5.2, Section 5.1.3
Can it model the black-box of a process?	√	√	Section 4.6
Can it add annotation to the model?	√	√	Section 4.7
Is it mapped to an executable language?	√	√	Section 4.8
Can it expose the uncertainty in a process?	×	√	Section 5.1.3
Is it fault tolerant/convergent?	×	√	Section 5.1.3
Can it find the similarity among processes?	×	√	Section 5.4.2
Can it find the key point in a process?	×	√	Section 5.1.3, Section 5.5.2
Can the information of the model be compressed?	×	√	Section 5.5.2
Can it show the labor division in the process?	(√)	√	Section 5.1.3, Section 5.2.2
Can it model autonomous behavior/asynchronous processes?	×	√	Section 5.6.2
Is there process management software or tools in the market using this method?	√	×	Section 6.1

Table 6.1: ASSESSMENT CRITERIA AND EVALUATION RESULTS

In essence, the analysis is performed using Promise Analysis approach as we discussed in the section 1.5. The list is a summary of chapter 4 and chapter 5 capturing the most important criteria to check for and the result. The leftmost column of table 6.1 show the examination criteria at a glance. The columns captioned with a method name contain the grade of compliance of that method to a criterion. A "✓" denotes the criterion to be satisfactory fulfilled, a "✗" marks a failed criterion and a "(✓)" means partially fulfilled.

From the checklist, we can easily see that BPMN and Promise Theory both can model the activities, roles, black-box, annotations in a process and both are mapped to an executable process modeling langauge. BPMN partially fulfills the criteria of modeling the decision making in a process because it is impossible to list all the alternative decisions in the BPD since there is always uncertainty exists. And it also partially fulfills the criteria of modeling the message between agents since when there are more than two agents in the process and there are much interactions, then lots of Message Flows fly across the diagram which looks a bit messy.

Promise Theory partially fulfills the criterion of modeling the sequence of execution activities because it is only possible to figure out the sequens in terms of conditional promise, and it is not so easy to figure when there are many activities or non-conditional promises involved in a process. However, there are some other things we should be aware of as well:

- Actually, in our real life, sometimes, we don't care exact sequence of all the activities A, B, C, D, E in a process. For example, we just need to make sure that A is always executed in front of E is fine. B, C, D can be executed parallel (some starts a little earlier, some starts a little later) and the order is not so important.

- What's more, when an activity is modeled with a non-conditional promise, it means that this activity is independent. It further means that there is no requirement about the execution sequence of this activity. However, BPMN tries to put this kind of activities in a linear way in a flow as well.

Further more, when we use cfengine to implement the ordering of the activities in a process. The output tells us more. An simple example of how cfengine manages the ordering is shown in appendix. The output is:

```
atlas$ /usr/local/sbin/cf-agent -f ./unit_ordering.cf
Q "../bin/echo one": one
-> Last 1 QUOTEed lines were generated by "/bin/echo one"
Q "../bin/echo two": two
-> Last 1 QUOTEed lines were generated by "/bin/echo two"
Q "../bin/echo three": three
-> Last 1 QUOTEed lines were generated by "/bin/echo three"
Q "../bin/echo four": four
-> Last 1 QUOTEed lines were generated by "/bin/echo four"
Q "../bin/echo five": five
-> Last 1 QUOTEed lines were generated by "/bin/echo five"
```

We can see from the output clearly that each line indicates which promises are executed and what is the result of that promise. In this example, we first got "one", then in order "two", "three", "four", then we come to a question – whether the promises are correct. If yes, then we got "five", otherwise we got "six". To verify it in the error case, we made an error in the scripts. We finally got "six" instead of "five" this time. The output is:

```
atlas$ /usr/local/sbin/cf−agent −f ./unit_ordering.cf
Proposed executable /bin/xecho doesn't exist
/bin/xecho one promises to be executable but isn't
Q "..../bin/echo two": two
−> Last 1 QUOTEed lines were generated by "/bin/echo two"
Q "..../bin/echo three": three
−> Last 1 QUOTEed lines were generated by "/bin/echo three"
Q "..../bin/echo four": four
−> Last 1 QUOTEed lines were generated by "/bin/echo four"
Q "..../bin/echo six": six
−> Last 1 QUOTEed lines were generated by "/bin/echo six"
Proposed executable /bin/xecho doesn't exist
/bin/xecho one promises to be executable but isn't
```

We can also see from the last two lines of the output, it indicates which promises are not met (where the error occurs), it is very helpful for error detection. This is the result of we first time run the script. Actually cfengine is running this script periodically according to your settings. When the same script is running more times later, it only checks if there are promises which haven't been executed and only execute those. In this way the process can always come convergent to the final destination we want to achieve.

Apart from that, the autonomous property makes Promise theory being able to model the cooperation between agents/roles in terms of messages or trust very well, and also the autonomous behavior/asynchronous process without difficulty. Because of its convergence property, it can also expose the uncertainty in a process and make the process fault tolerant. What's more, the information can be compressed when there are many identical promises in a process. By counting the number of promises a roles made, it can also find the key point in a process.

BPMN tends to make everything linear, e.g., straight line flow or layer upon layer. The modeling method itself makes it hard to have the above the features which Promise Theory additionally has. However, since BPMN has been set as the standard business process management notation for many years, there are already many software or tools in the market which can automate this process modeling. When using them, the users can just select, drag and put the symbols they want to use from the tool bar in the software.

6.2 Two ways of management

BPMN focuses on events and the model is based on transaction. BPMN tends to make everything linear, e.g., straight line flow or layer upon layer. When

we model process using BPMN, it is a kind of bottom up management. Basically it would assume that it knows all the steps involved in the process, then put them in a sequence or loop somewhere. In order to fulfill that, it would be based on experience. One problem with this is that if the process is too complicated, then the person doing the modeling would easily be overloaded or even lost while collecting information. What's more, in this way BPMN seems to give end users a recipe of how to work the process, but it is impossible to model all the changes/uncertainty during the process. Then in a BPD, some limitation is defined, e.g., the start event, then give the steps in that particular situation. Therefore, we say that BPMN just gives you an example about how the process will work.

On the other hand, when Promise Theory describes a process, it focuses on cooperation between the agents/roles. A simple promise concept can show a lot of information of the process, such like message and trust between agents/roles. Also its autonomy idea makes the model of asynchronous activity automatically. One could just as easily claims that it is bottom-up management, since it starts with the component agents and joining them with promises. However, we can also interpret it as a kind of top down management in another perspective. In the modeling, it views the whole process as a high-level promise. For example, when a product produced by many departments finally coming to the market manager labeled with "quality approved", it is actually a high level promise of the production process. The manager should trust that the activities delivered from its previous role has kept its promise of the product quality. The same its previous role should trust that its previous role has kept its promise as well, the same forth. Of course many lower level promises might need to be made first in order to fulfill the high level promise. In this way we break down a high level promise into many lower level practical promises. But when you decide which roles are related and what promises should be made, it must be based on trust.The promise theory tells you what you need minimum to make it work.

Chapter 7

Conclusion

Today in order to take the challenges of creating and preserving value in a highly competitive environment and also meeting their clients' various changing requirements, business people are encouraged to improve their way of thinking about and managing their business. Process Management is about how to manage the processes involved in the business effectively and efficiently in order to fulfill the requirements from the clients.

As the trend of IT-business alignment in recent years, a proper process management method is desired to not only model but also be able to automate the complex processes afterwards. In this way, the managers or system administrators can plan their work from an abstract model of the process and automate the precesses later to save a lot of time and energy.

In this project, two popular and very different process modeling methods are compared. One is BPMN which uses the traditional method to model the process in a flow. The other method is Promise Theory which models the process in a network of interacting autonomous agents. Promises can be asserted either from an agent to itself or from one agent to another and each promise implies a constraint on the behavior of the promising agent.

For feature judgement, a Promise Analysis approach was used in this project to compare the two methods. From the check-list result in chapter 6, we can see that Promise Theory can model more features of a process than BPMN.

For usability and learnability, there are much more symbols in BPMN than Promise Theory, which is on one hand provides more detailed information about the process than fewer symbols; on the other hand, it would be more difficult than fewer symbols for users to learn and to remember in using. However, when using Promise Theory you have to use your brain a bit to figure out what promises an agent/role has to make in order to fulfill the process according to the autonomous rule. In contrast, when using BPMN you don't need so much thinking. Do you prefer a method easy to learn but need some thinking when use it or you like a method hard to remember lots of symbols but easy

when using.

For value judgement,it is not easy to conclud which method is better or worse. It is always context dependent.

- Because of the average performance of Promise Theory in modeling the execution sequence of the activities in a process, under the circumstances where there are more synchronous activities involved in the process or fewer interaction between the agents, e.g.,there is only one agent in the process, it is better to use BPMN, since sequence of execution of activities became more important and there is no interaction or cooperation exist in this situation.

- Under the circumstances where there are a lots of agents/roles involved in the process, to model the interaction or cooperation between the agents/ roles are unavoidable, Promise Theory is better, since a lots of Message Flow flying across a BPD looks a bit messy.

- When there are many asynchronous activities in a process, Promise Theory is best in this circumstance, since the representation of asynchrony is automatic in a promise.

7.1 Future Work

In this project, the solution to select the comparison criteria is not sufficiently scientific, which may directly affects the conclusion on which method is better in feature, because we obtain the conclusion only by counting the number of fulfilled criteria by each methods. The rule for selecting scenario in this project needs to be improved as well. If there is more time, it would be nice to do a deep research of the desired criteria of a process management method or simply do a questionnaire towards the system administrators or business managers about what criteria they prefer for a good process management method. Then it has value to redo this project again with deleting the undesired criteria or adding the additional criteria according to the research or questionnaire result.

Also, it would be of use to write some scripts to show the automation of the scenarios using cfengine3. however, since the scenarios we have chosen in this project include not only machines but also humans. It may increase the difficulty of scripting using cfengine3 to model the human interaction at the same time in a process.

What's more, Matt Disney has done a research on exploring patterns for scalability of network adminstration with topology constraints in his master thesis project in 2007.[35] In his project, he has tried to find the right pattern for scalability via calculating the time it costs for a change in the network to be convergent which means fully distributed among all the agents in the network.

It inspires me that it would also be a useful project in the future to explore the timing for an established process model to be convergent in a network. It would give valuable parameter for the development of process management tools.

Appendix A

Example of cfengine ordering

An simple example of cfengine ordering (unit_ordering.cf)

```
bundle agent order
{
  vars:
        "list" slist => { "three", "four" };

  commands:

        ok_later::
                "/bin/echo five";

        otherthing::
                "/bin/echo six";

        any::
                "/bin/echo one" classes => d("ok_later","otherthing");
                "/bin/echo two";
                "/bin/echo $(list)";

        preserved_class::
                "/bin/echo seven";
}

###############################################

body classes d(if,else)

{
        promise_repaired => { "$(if)" };
        repair_failed => { "$(else)" };
        persist_time => "0";
}
```

Now make an error in the script

```
bundle agent order
{
  vars:
        "list" slist => { "three", "four" };

  commands:

        ok_later::
                "/bin/echo five";

        otherthing::
                "/bin/echo six";

        any::
                "/bin/echox one" classes => d("ok_later","otherthing");
                "/bin/echo two";
                "/bin/echo $(list)";

        preserved_class::
                "/bin/echo seven";
}

#############################################

body classes d(if,else)

{
        promise_repaired => { "$(if)" };
        repair_failed => { "$(else)" };
        persist_time => "0";
}
```

Bibliography

[1] Michael Rosemann Jorg Becker, Martin Kugeler. *Process Management:a guide for the design of business processes*. ISBN 3-540434-99-2. 2003.

[2] Evelyn Hubbert. The forrester wave:data center automaiton,q2 2008. Technical report, 2008.

[3] Business process execution language. http://en.wikipedia.org/wiki/Business Process Execution Language.

[4] Zhao Xiangpeng Yang Hongli Qiu Zongyan, Cai Chao. Exploring into the essence of choreography. Master's thesis, Department of Informatics,Peking University.

[5] Jim Alateras. Illustrating orchestration and choreography. In *OŔeilly Search*, 2006.

[6] Colin Rudd etc Alison Cartlidge, Ashley Hanna. *An Introductory Overview of ITIL v3*. The UK Chapter of the itSMF, 2007.

[7] Joshua M. Epstein. Why model? In *Journal of Artificial Societies and Social Simulation*, page 12, 2008.

[8] Mark Burgess. Unifying configuration mangement and knowledge management (cfengine on topical islands). 2009.

[9] THOMAS SCHAAF MARK BURGESS. *INTEGRATING CFENGINE, ITIL AND ENTERPRISE PROCESSES*. July 2008. page 6-7,23-25.

[10] IBM Stephen A.White. Introduction to bpmn. 2006.

[11] S.A. White. Business process modeling notation (bpmn). *Business Process Management Initiative (BPMI)*, 2004.

[12] Object Management Group. Business process model and notation (bpmn) 2.0 request for proposal. 2007.

[13] Wil M.P. van der Aalst. Patterns and xpdl: A critical evaluation of the xml process definition language. pages 2–3.

[14] Magnus Penker Hans-Erik Eriksson. *Business Modeling With UML: Business Patterns at Work*. 0471295515. John Wiley Sons, Inc., 1998.

[15] Satish Mishra. Visual modeling unified modeling language (uml) : Intro-
duction to uml. Rational Software Corporation, 1997.

[16] Armin Zimmermann. Stochastic discrete event systems: Modeling, eval-
uation, applications. ISBN 3540741720, page 52, 2007.

[17] Asbjorn Rolstadas. Business process modeling and reengineering. In
Performance Management: A Business Process Benchmarking Approach, pages
148–150, 1995.

[18] Thomas Davenport. *Process Innovation: Reengineering work through infor-
mation technology*. Harvard Business School Press, Boston, 1993.

[19] Michael Hammer and James Champy. *Reengineering the Corporation: A
Manifesto for Business Revolution*. Harper Business, 1993.

[20] Rummler; Brache. *Improving Performance: How to manage the white space on
the organizational chart*. San Francisco, 1995.

[21] Henry J. Johansson. *Business Process Reengineering: BreakPoint Strategies
for Market Dominance*. John Wiley Sons, 1993.

[22] WFMC. Workflow mngt coalition terminology and glossary (wfmc-tc-
1011). Technical report, Brussels, 1996.

[23] Mark Burgess. *Analytical Network And System Administration*. John Wiley,
2004.

[24] *Cfengine reference manual (version 3.0.1b1)*.

[25] Mark Burgess. http://research.iu.hio.no/promises.php.

[26] Jan Bergstra; Jan Bergstra. A static theory of promises. 2008.

[27] Mark Burgess Jan Bergstra. Local and global trust based on the concept
of promises. 2006.

[28] Mark Burgess. An approach to understanding policy based on autonomy
and voluntary cooperation. 2005.

[29] Mark Burgess; Siri Fagernes. Voluntary economic cooperation in policy
based management. 2004.

[30] Mark Burgess; Siri Fagernes. Laws of human-computer behaviour and
collective organization. *IEEE Transactions on Network and Service manage-
ment*, 1999.

[31] Mark Burgess and Siri Fagernes. Laws of human-computer behaviour
and collective organization.   IEEE TRANSACTIONS ON NETWORK
AND SERVICE MANAGEMENT, 1999.

[32] Jog Raj Martin Owen. *BPMN and Business Process Management – Introduc-
tion to the New Business Process Modeling Standard*. Popkin Software, 2003.

[33] OMG. Business process modeling notation(bpmn) version 1.2, 2009-01-03.

[34] http://www.cfengine.com/pages/cfengine3.

[35] Matthew S. Disney. Exploring patterns for scalability of network administration with topology constraints. Master's thesis, Oslo University College, 2007.