Simulation of multi-company product development processes in CPN tools

van de Pol, Maikel M.L.

Award date:
2005

TECHNISCHE UNIVERSITEIT EINDHOVEN
Department of Mathematics and Computer Science

MASTER’S THESIS

Simulation of multi-company product development processes in CPN Tools

by
M.M.L. van de Pol

Supervisors:

Dr. Ir. H.J. Pels (TUE)
Dr. M. Voorhoeve (TUE)

Eindhoven, June 2005

Preface

This report is the final product of my graduation project that I performed from August 2004 till May 2005. It describes the research and the work that I've done to complete my graduation project to obtain the degree in Computer Science at Eindhoven University of Technology. The project was carried out at the Information Systems group of the Department of Industrial Engineering and Management Science. I would like to thank several people who supported me during this project.

First of all I would like to thank my supervisors H.J. Pels and M. Voorhoeve for the support I received from them during the entire project. Also, I would like to thank W.M.P. van der Aalst and K.M. van Hee for the feedback they gave me during the midway session. I've learned a lot from all the positive comments that I received from all of them.

During the first half of the assignment I worked together with Raymond Benders, a student from the department of Industrial Engineering and Management Science. He conducted research on the business model that I used as the basis for developing the simulation tool during this research. I would like to thank Raymond for the many pleasant hours we spent together working on our project. I like to thank Raymond and Michiel Dreverman, also a graduate student and our room mate, for all the fun we had during the time we were not working on our assignments.

Last but not least I like to thank my girlfriend, Janine Douwes, for the support she gave me during my graduation assignment and during the rest of my time at the University. She stimulated me to do my work, but also to take some time off when needed.

I hope you'll enjoy reading this report, as much as I enjoyed creating it.

Pleasant readings,

Maikel van de Pol

Abstract

This document describes the development of a generic simulation tool for measuring the performance of a multi-company product development process. The model is used to visualize the effects of collaborative engineering and product data exchange on the performance of the product development process. The model is created as standard building blocks in CPN Tools and is used at Eindhoven University of Technology.

Management Summary

Introduction

Product development processes are different from operational processes. They can be characterized by uncertainty, complexity and iterations and therefore they are hard to model using workflow languages. This complexity even increases when the process is spread over several companies. Collaborative engineering and product data exchange put extra demands on a simulation model. Most workflow languages have a local nature. This means that consequences of a task or task rejection that affect surrounding tasks can be modeled easily. Influences that have wider impact can be modeled, but often result in complex, spaghetti like diagrams. This complexity occurs especially when a rejection affects a large part of a workflow net or several parts of a workflow net.

The goal of this project is to:
Develop a generic executable simulation tool for product development processes, that is able to show the effects of collaborative engineering and product data exchange on the performance of product development processes. The tool is intended for researchers at Eindhoven University.

To be able to cope with this complexity inherent to product development processes, a new simulation model was developed. This simulation model consists of building blocks containing constructs to model development processes. Starting from the RapidPDM model that was initially created to show the effect of Product Data Management (PDM) on product development processes, an enhanced model was created. This new model is aimed at product development processes in general and not specific at product data management issues.

The model considers product development processes to be a flow of deliverables. On every deliverable one or more tasks are performed in a given order (Figure 1). A task is performed in a certain lifecycle phase. More than one task can be performed in one lifecycle phase.

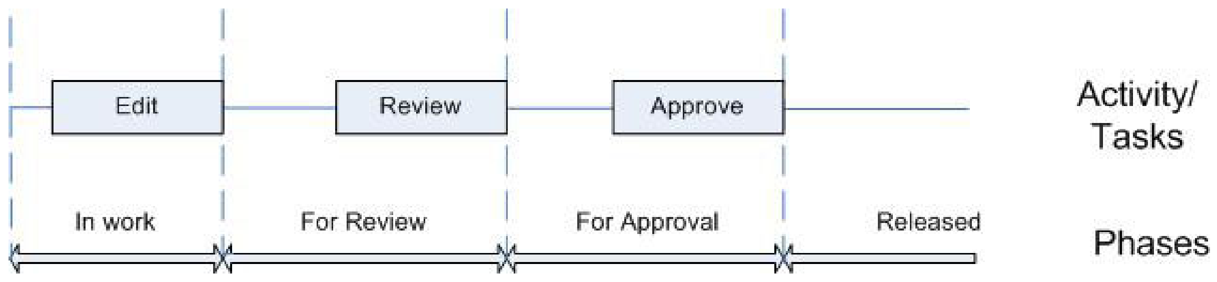


Figure 1: A lifecycle

Two types of relations exist between deliverables: precedence relations and interrupt relations. The first specify the order in which the deliverables are created, the second specify which deliverables are affected when a task or a deliverable is rejected. After the rejection or interruption of a task a new version is created (Figure 2).

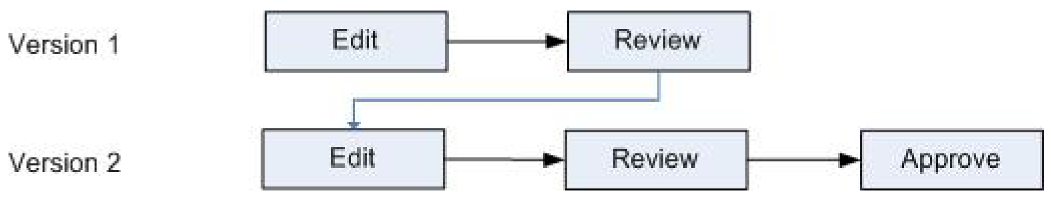


Figure 2: Versions

Design decisions

The RapidPDM model was built in Exspect and it uses a non-standard modeling approach. Instead of modeling every task as a transition or subnet, it consists of exactly one implementation of an 'abstract' task. This abstract task then takes sets of deliverables or tasks that are in the same state. For example the ‘assign engineer’ transition does not assign engineer 'John' to job ‘Edit’ of the deliverable ‘Specification’, but assigns engineers to all jobs that are in the state to have an engineer assigned simultaneously. ExSpect, using a functional programming language with constructs for handling sets is quite suitable for this way of modeling. This approach originates from a data-oriented view on product development processes. All data is available in data stores, and process steps alter this data.

During this research a different approach is used. Exspect is a tool that is not under active development anymore. Currently CPN Tools is the Petri Net modeling language that is used at Eindhoven University of Technology. The enhanced model is developed in CPN Tools to encourage the intended user, researchers at the university, to use the model. In the new model a process-oriented view is used. This view is based on a series of tasks that are performed in parallel or sequentially; a so-called workflow net. Every task in this workflow net possibly changes some data-elements. These data elements are mostly passed on from task to task and not stored globally.

The RapidPDM model contains some aspects that are unsuitable for the new model. The most important of which are:

- No multi company support. There are no means of distinguishing between engineers from different companies
- No affected deliverables. Interrupts only influence successive tasks. It is not possible to indicate that an input deliverable has to be adjusted, before work can continue.
- Only one task per phase. In some cases, several tasks are performed in parallel. For example: two reviewers receive a deliverable at the same time and review it simultaneously.
- Global values for post times. In the RapidPDM model the post time is represented by a global variable. This was used to show the difference between using a PDM system (post time = 0) and not using a PDM system (post time = x).

The enhanced model

The enhanced model contains three hierarchical levels. The highest level is the process level. At this level the deliverables are identified and the interrupt routing is defined. No trigger tokens are defined on this level, because precedence is defined on tasks and not on deliverables. The second level of the hierarchy is the lifecycle. Every deliverable has a lifecycle. A lifecycle consists of one or several tasks that are performed sequentially or in parallel. Also, deliverables are created and deliverables can be withdrawn at this level. The lowest level of the hierarchy is the task level. At this level most of the complexity is implemented. Precedence, resource allocation, acceptation and rejection of tasks, and interrupt handling takes place at this level.

Triggers are normally modeled using one trigger token for a trigger from deliverable A to deliverable B. In this model however, a different approach is used. All statuses of all deliverables are kept in one global place. Every task in the entire process can have a precondition that is based on this global place. The precondition specifies which deliverables and which statuses of those deliverables are needed for a task to start.

Interrupts are passed through the entire model instantaneously. An interrupt is created by the rejection of a task, for example a review. This interrupt is passed on to the successive deliverables and possibly to the affected deliverables. Affected deliverables are the input deliverables that have to be adjusted before work can continue. Interrupt constructs are modeled on every hierarchical level.

Implementation
Interrupts are the most difficult aspect of product development processes. Because of the complex and uncertain nature of product development processes, rejections and interrupts can take place anytime, anywhere and can possibly influence every part of the process. This difficulty was solved by handling interrupts inside the standard building blocks. The user of the tool is not concerned with how exactly the interrupts are handled. He just defines which deliverables are affected by the rejection of a deliverable.

The user interface provides the user with an abstraction from the interrupt and trigger complexity. The main advantage is that he is not concerned with all the complex details of the implementation of interrupts and triggers. The user just defines deliverables and connects a deliverable to deliverables that are affected by an error or a rejection in a review. He defines precedence by specifying which phases of input deliverables are needed for a task to start.

Conclusions
The new approach uses separate nets for separate deliverables instead of one abstract net for the set of all deliverables. The main advantages of the new modeling approach are easy configuration, flexibility and visualization of the process steps. These advantages give the user more insight in the product development process. The main disadvantage is that this approach produces large nets, which can cause performance problems in the simulation tool. These performance problems form a burden in using CPN Tools to simulate product development processes. Another disadvantage is that a lot of information is only readily available to a small part of the model. This makes some concepts, for example priority of tasks, more difficult to model.

Definition of Terms

RapidPDM	Name of the project (1998-2000) that developed a simulation model to show the benefits of a PDM system.
RapidPDM model	The model as it was developed in the RapidPDM project.
Enhanced model	The model as it will be developed during this graduation project.
Deliverable	A deliverable is a document or a physical object, for example a prototype that is created during the design process of a product. The RapidPDM model only speaks of documents.
Document	A document is a deliverable that is not a physical object. The RapidPDM model only speaks of documents.
Lifecycle	A lifecycle is the sequence of phases that a deliverable goes through during the design process.
Lifecycle phase	Every deliverable has a certain lifecycle. The lifecycle phase of a deliverable determines the status of the deliverable and triggers tasks on the same or other deliverables.
Activity	An activity is the generic term for the collection of tasks that are performed during one lifecycle phase of a deliverable
Task	A task is an operation that has to be performed on a deliverable. If all tasks on a deliverable are successfully completed, the deliverable is released.
Subtask	If an activity contains more than one tasks, then these tasks are also called subtasks.
Job	A job is an instantiation of a task on a specific deliverable with a specific engineer.

Table of Contents

1 Introduction

1.1 Assignment background

This assignment forms the graduation project of my study Computer Science at Eindhoven University of Technology. It will be carried out at the Information Systems department of the faculty Technology Management. My supervisors during this project are dr. ir. H.J. Pels (Technology Management) and dr. M. Voorhoeve (Computer Science)

1.2 Assignment

Product development is a complicated process that is usually subject to much uncertainty. Therefore it is very difficult to predict the course of such a process. Rapidly changing market trends force product development companies to work faster and at lower costs. Some of the most important industry trends that push the needs for better performance of the design process are [Pels, 2004]:

- **Shorter commercial product life cycles**: some decades ago a new product would stay in the catalogue for at least five years. Today the commercial life cycle is often less then 6 months.
- **Increasing product complexit**y: in order to provide more functionality and better product performance the complexity of the products, and thus the number of deliverables needed for manufacturing and support, increases.
- **Increasing product variet**y: flexible automation and modern ERP systems enable efficient production in small series. This enables the introduction of more variants of the same product family. Typical examples are cars with different colors, engines, door-configurations, and lots of options for electronic gadgets.
- **Personalized products**: in order to respond better to the individual needs of customers, products that used to be mass manufactured are now designed to customer order.

In practice Product Data Management Systems are being used to control the flow of documents that is created during the design process. These systems assist in for example the storage and retrieval of documents and keeping track of changes. Product Data Management Systems are expensive software packages and the advantages of such software packages are not always directly obvious. This is one of the reasons why the RapidPDM (1999) project developed a tool that can be used to simulate design processes and show the advantages of a well-organized document management system.

Product Data Management is a rapidly changing field of research. The RapidPDM simulation model is already five years old and at some points outdated. Besides, it does not support some key features of product development that have emerged during the past few years. The most important of these is collaborative engineering. More and more companies outsource parts of the design process and focus the design process at the core design activities. A simulation model needs to be developed to visualize the effects of product data management in collaborative engineering environments. A simulation model also provides more insight in product development processes spread out over several companies in general. Therefore the following graduation assignment is formulated:

Develop a generic executable simulation tool for product development processes, starting from the principles of the RapidPDM project that will be able to visualize the effects of collaborative engineering and product data exchange on the performance of product development processes.

In this formulation two terms are used that need explanation: collaborative engineering and product data exchange.

Collaborative engineering is used when two or more engineers from two or more different companies or organizations work together on the same design with considerable interaction during the process.

Product Data Exchange refers to the technology used to support the exchange of documents between companies or organizations.

The RapidPDM project is chosen as a starting point for two reasons. First, an implementation is available at hand, such that design decisions can be validated immediately. Second, this model is well documented and a lot of knowledge is available at Eindhoven University. Hardly any information is available on other attempts to model and simulate product development processes.

A disadvantage of the RapidPDM model is that it does not visualize the flow of deliverables in the executable model. Only visualizations of the output parameters are shown. Visualization of the process is a desired feature, because the model is also meant to give more insight in product development processes. Also Exspect, the implementation language of the RapidPDM model, is not under active development anymore.

The first part of the project was mainly based on the case that R. Benders described during his graduation project, see [Benders, 2004]. The first intention was to focus the entire project around this case, but only halfway it came clear that the company where the case was situated had no intention of using this tool on other cases. Therefore it was decided to aim the tool at future research within the department of Technology Management at Eindhoven University. This imposed the demand of making the simulation tool as building blocks within CPN Tools, such that researchers at this department would be easily encouraged to use the tool. Therefore the scope of the assigned was narrowed as follows:

The simulation model must consist of standard building blocks in CPN Tools from which, with little effort, a simulation model can be built. The model must be applicable in many different engineering environments and must show the effects of product data exchange and collaborative engineering.

Using building blocks as a modeling technique, the two disadvantages of the RapidPDM model are dealt with. Every step of the process is now modeled separately by copying the standard building blocks and connecting them to construct a workflow model. Moreover, CPN Tools is used as the implementation language instead of Exspect. In contrast to Exspect, CPN Tools is under active development. Good support is available from the University of Aarhus, Denmark.

To give direction to the research that will be performed to come to an enhanced model, some main *research questions* are formulated:

1. How has the field of product data management developed over the past five years?
2. What are the main characteristics that influence collaborative development? Which input parameters does the simulation model need? What functionality should be implemented in a simulation model to come to useful results?
3. What are useful metrics to measure collaborative product development performance? Which output parameters should the simulation model deliver?

1.3 Research plan

Simultaneous with this graduation project another student from Eindhoven University is performing a related graduation project. From June 2004 till December 2004 Raymond Benders is working on a process model for measuring performance in a collaborative engineering environment. He works from a management science point of view. Using input from his case he will provide suggestions to improve the business model that forms the basis for the RapidPDM simulation model.

His assignment however is mainly aimed at the exchange of product data during the development process. Therefore this assignment will focus on the collaborative engineering aspects of the business model and on the computer science aspects of the simulation model.

First the recent developments in product data management will be researched. This will be done by means of interviews and literature study. Secondly research will be performed to get insight into what engineering characteristics should be implemented in a model and which aspects can be left out. This will be done by evaluating the influence these characteristics have on the outcome of the engineering process.

The third aspect to be researched is which output parameters are of interest in collaborative engineering. The RapidPDM model uses the universal variables time, cost and quality, but especially quality has been shown to be hard to quantify. In some later reports the terms content, work and result are used. Interviews and literature studies will be used to find the best suitable output parameters.

2 Background information & Literature

2.1 Product Development

This section gives an introduction to the problems evolving during product development and explains why simulation is used to visualize the effects of collaborative engineering and product data exchange. Throughout this report the following definition of product development will be adopted:

2.1.1 Definition

[De Graaf, 1996] defines product development as*:*

Product Development is a sequence of design processes that converts generally specified market needs or ideas into detailed information for satisfactory manufacturable products, through the application of scientific, technical and creative principles, acknowledging the requirements set by succeeding life-cycle processes.

This definition refers to product development as a sequence of processes that create information, unlike manufacturing processes that create physical objects. But this is not the only difference. Because development processes create information that is not already available, otherwise it would not be developed, it requires a different approach then manufacturing. In manufacturing environments product specifications are available according to which the products can be made directly. Product development processes often start from scratch and this results in characteristics that are typical for product development.

Product development is sometimes confused with Research & Development [Helms, 2002]. Research & Development (R&D) is defined as fundamental research that does not result in 'new' products. The output of research & development is a new technology, new functionality or a new material, which is input for product development to apply in new products. The duration of an R&D activity is typically longer than that of a product development activity. It can take more than 10 years to develop a new material or technology while it often takes (‘only’) several months to several years to develop and bring a new product to the market. This research is not aimed at R&D but focuses solely on product development.

2.1.2 Characteristics

Because of their creative and iterative nature, product development processes have some common characteristics of which the most important ones are described by [Helms, 2002]:

Uncertainty
Development almost always handles products that do not yet exist. Market needs are converted to (information for) products, and this would not be needed if the product already existed in the desired form. But market needs can change over time, thus changing the input for the development process. Some experience from finished development processes can be used to predict the course of the development but some uncertainty always remains.

Iterations
Two types of iterations are identified by [Clausing, 1994]: *creative* iteration and *dysfunctional* iteration. Creative iteration implies that several iterations are required to conceive an idea, consider it from all perspectives instantaneously, and draw the final design. During the iterations the idea slowly evolves to the final design. Dysfunctional iteration involves amongst others the stream of changes caused by problems regarding the manufacturability and maintainability of the product or by additional customer requirements. As a result designers and engineers have to revise their work.

Product complexity
Over the last decades the complexity of products has increased, e.g. by the application of electrical and software components. The increased product complexity also affects the required development staff. The increased complexity can be illustrated by table 1, showing the number of lines of code in the Operating System Microsoft Windows. This major increase in complexity is mainly caused by the continuous growth of the functionality necessary to meet market expectations.

Year	Operating System	Lines of Code (Million)
1990	Windows 3.1	3
1995	Windows NT	4
1997	Windows 95	15
1998	Windows NT 4.0	16
1999	Windows 98	18
2000	Windows NT 5.0	20
2001	Windows 2000	35
2002	Windows XP	40

Table 1: Example of product complexity

These characteristics make it hard to use analytical tools and techniques for product development processes. This is why simulation is used to model them and to show the benefits of using product data management and collaborative engineering principles.

2.1.3 The design process

Product development projects involve the completion of lots of tasks. These tasks are not independent and have to be performed in a prescribed order.

In [Ulrich, Eppinger] three basic types of tasks are defined that make up the design process: sequential, parallel and coupled tasks. Sequential tasks are tasks that can only finish in the specified sequence. This does not mean they can not be performed in parallel, it only means that the later task can not finish before the earlier task has finished. Generally some parts of the later tasks can already be performed before the earlier task has finished.

Parallel tasks are tasks that can be performed independently. They can start and finish without interaction. Coupled tasks on the other hand are mutually dependant. Each task requires the result of the other tasks in order to be completed. Coupled tasks either must be executed simultaneously with continual exchanges of information or must be carried out in an iterative fashion.

2.2 Collaborative engineering

As already mentioned in section 1.1 collaborative engineering implies that two or more engineers from two or more different companies work together on the same design with considerable interaction during the process. For collaborative engineering to take effect, some form of concurrent engineering has to be implemented.

Concurrent engineering is the principle of performing development steps partly overlapped instead of sequentially. For example: some parts of the functional design are already created before the requirements phase has finished. Concurrent engineering requires people from different departments to work in close collaboration to develop parts of the design in parallel. Often interdisciplinary work teams will be formed in order to enable concurrent engineering. Collaborative engineering is when concurrent engineering is extended to more than one company. This requires considerable communication between the two companies. Because

this communication is very important for collaborative engineering to succeed, product data exchange is of great importance.
Product Data Exchange refers to technology used to support the exchange of documents between companies. It enables faster and better communication and therefore a higher degree of collaborative engineering. Optimal use of product data exchange makes a document released by company A immediately and in the right format available to the right person at company B.

2.3 Product Data Management

As mentioned in the previous section, communication is of vital importance in collaborative engineering. Successful communication of information requires well-organized data management within the companies involved. Product data management is one of the cornerstones of successful collaboration.

To give an overview of the difference in PDM some years ago and the contemporary views, some definitions are mentioned here. Many different abbreviations were introduced. This section defines the meaning of product data management used in this report.

2.3.1 Some definitions

CIMdata: ‘Product Data Management (PDM) is a tool that helps engineers and others manage both data and the product development process. PDM systems keep track of the masses of data and information required to design, manufacture or build and, then support and maintain products’. (CIMdata, 1995):

Goossenaerts, Pels: '*PDM is the discipline of making product data available and accessible to all parties involved in the product lifecycle and to support and enhance all business processes that create or use product data*'. (Goossenaerts, Pels, 1998)

These definitions focus on PDM as a tool or discipline for managing product or engineering data and making these data available to the engineering process. During the maturing years of PDM, lots of new terms arose like collaborative product definition management (cPDM), collaborative product commerce (CPC) and Product Development Management (PDM II). Most of these terms were mostly used by one company to express that PDM developed to a more mature tool or discipline then the primitive document management systems from the early PDM years. Over the past few years the term Product Lifecycle Management (PLM) became commonly used. Some recent definitions:

Product Lifecycle Management:
‘PLM is a strategic business approach that applies a consistent set of business solutions in support of the collaborative creation, management, dissemination, and use of product definition information across the extended enterprise from concept to end of life - integrating people, processes, business systems, and information.’ (CIMdata, 2004)

Product Life Cycle Management
‘At its highest definitional level, product life cycle management (PLM) is a process for guiding products from idea through retirement to deliver the most business value to an enterprise and its trading partners. PLM employs product information and business analysis to support product portfolio strategy, product life cycle planning, management of activities, and execution of those activities through each phase in a product's life. The applications that support the business activities enabled through PLM includes product ideation, design, engineering, manufacturing process management, product data management, and product portfolio management.’ (Gartner Group, 2004*)*

Product Lifecycle Management
'A business strategy that helps companies share product data, apply common processes, and leverage corporate knowledge for the development of products from conception to retirement, across the extended enterprise. By including all actors (company departments, business partners, suppliers, OEM, and customers), PLM enables this entire network to operate as a single entity to conceptualize, design, build, and support products.' (Dassault, 2004)

Product Data Management
'PDM is the product knowledge and business process platform of any PLM solution. A PDM solution automatically captures and manages product information and facilitates collaboration throughout the enterprise and across the value chain.' *(Dassault, 2004)*

Generally speaking the term 'Product Lifecycle Management' seems to be the most commonly used term. Product Data Management seems to be referred to as a part of product lifecycle management (Gartner group, Dassault).

Another important issue in the newer views on PDM is collaboration. An increasing number of development processes is performed by more than one company. Therefore a PDM system needs to be able to handle multi-company development processes, illustrated by the following statement:

'In response to the requirements of their customers, suppliers have continued to expand the scope of their product vision, putting much more emphasis on the concepts of collaboration and management of product definition, rather than on just managing product data.' (CIMdata, 2004)

2.3.2 Definition of PDM used in this report

While PLM is mainly focused on the business strategy of managing a product through its entire lifecycle, PDM remains the generally used term to indicate the document management part of PLM. Because the purpose of the simulation model is to show effects of collaborative engineering, the term PDM will be used in this report. Other parts of PLM, like manufacturing, product portfolio management, product disposal, do not pertain to the engineering process directly and are therefore outside the scope of the simulation model.

In this report product data management is defined as
'The discipline of making product data available and accessible to all parties involved in the product lifecycle and to support and enhance all business processes that create or use product data'. (Goossenaerts, Pels, 1998)

Using this definition emphasis is put on the creation of information and making it available and accessible to later tasks in the design process.

2.3.3 PDM concepts: lifecycles and versions

Two important concepts in product development processes are the deliverable lifecycle and versions. Every deliverable has a lifecycle. A lifecycle consists of a number of sequential phases. During the design process the deliverable goes through the lifecycle phases in the given order. Every phase except the released phase has an activity associated with it. An activity can consist of one or more tasks. Tasks are the smallest elements of the design process.

For example: after a deliverable is initiated it is in the 'Edit' phase. When all edit tasks have completed the deliverable is promoted to the 'For Review' phase. There it is checked for errors in two parallel review tasks that together make up the review activity. When no errors are found the deliverable is promoted to the 'For Approval' phase, where a manager gives his consent on the contents of the document. After this approval the deliverable is in the phase 'released'. A deliverable that is in phase 'Edit' is also said to 'have status' Edit.

Phases are used to control the progress of a deliverable in its lifecycle. Each phase contains one or more tasks. A task in phase 'For Review' can only start when the deliverable is in phase For Review. After completion of all tasks in a phase the deliverable enters the next phase and the deliverable is released for the tasks in the next phase of its lifecycle.

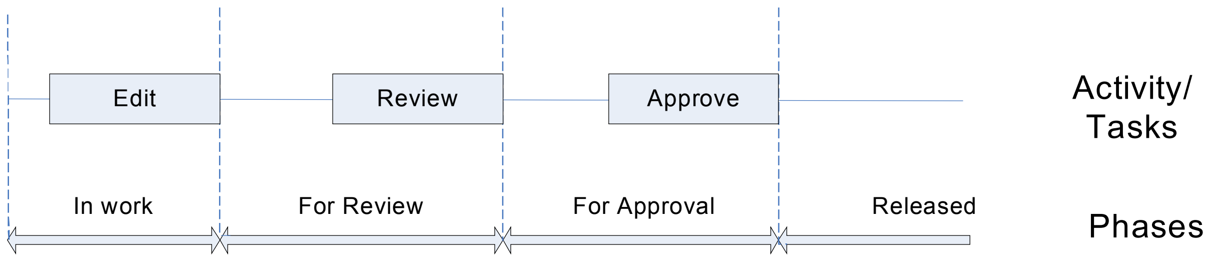


Figure 2.1: A Lifecycle

If a deliverable is rejected in a task, for example during a review, then a new version is created. This new version will start in the initial phase. The old version will remain in its last reached phase. The new version will normally go through the same lifecycle as the first version of the deliverable. Versions are mainly used for proper change control of a deliverable.

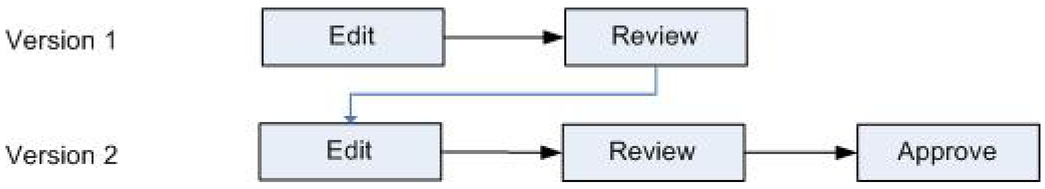


Figure 2.2: Versions

2.4 Simulation

According to [Smith, 1999] simulation is the process of designing a model of a real system or process and conducting experiments with that model. If the system or process is simple, the model may be represented and solved analytically. However, problems of interest in the real world are usually much more complex than this. In fact, they may be so complex that a simple mathematical model cannot be constructed to represent them. In this case, the behavior of the system or process must be estimated with a simulation.

In accordance with [Limam, 2001] simulation is used to understand the behavior of the system or process, to improve it and to predict performance / outcomes of the system or process. Understanding the behavior of the system is essential if one wants to control the output of the process. Improving the system can lead to significant corporate benefits like reduced costs and development times. Predicting the outcomes of process is only possible when there is a solid understanding of the system, which can be gained by performing simulations with different scenarios and analyzing the outcomes.

[Clark, 1993] defines three potential sources of advantage in the ability to move quickly in product development

- Quality of design

If a company has a development cycle of twelve month where its competitors have a development cycle of eighteen months, then there are six months left to gather information on market needs and customer preferences. This may lead to a better market conform configuration of the product that better fits the customers expectations.

- Product performance

Shorter development lifecycles allows for an earlier introduction of new technologies. In this case the advantage of speed lay not in customer expectation but in the ability to incorporate the latest technology in products sooner than the competition.

- Market Share and cost

A better product design or better performance of a product allows a company to achieve a higher price off a product or more value for the same price, increasing its market share, and thereby possibly gaining a high volume cost advantage.

3 RapidPDM Simulation model

3.1 Introduction

The RapidPDM simulation model was initially developed to 'visualize the effect of Product Data Management functionality and engineering process characteristic on the performance of product development processes in the quantifiable terms of time, costs, and quality' [Elzen, 1999]. This aim was part of the goal of the RapidPDM project to develop a PDM implementation methodology, supported with a set of computer-based tools.

3.2 Principles

In this section the principles of the RapidPDM simulation model are described. Section 3.3 describes the implementation of these principles. Note that some principles and implementation issues are not taken to the enhanced model. The main differences between the RapidPDM model and the enhanced model are described in section 3.4.

3.2.1 Product development as a flow of deliverables

Product development processes are modeled from a deliverable perspective. Deliverables (the RapidPDM simulation models speaks only of documents), like text files or CAD files, describe (physical) products. Unlike manufacturing processes, which create physical products, engineering processes create information. Before a product can be manufactured it has to be developed. The product is described in documents on which operations are performed. These operations are called tasks. Generally a task will consist of adding information to the deliverable or reviewing it. When an operation on a deliverable is completed this will enable the next operation on the deliverable and/or trigger operations on other (new) deliverables. For instance when the requirements of a product have been specified, design activities for the product can be initiated. These activities create new deliverables that describe the design of the product. Each task is performed in a separate lifecycle phase. The sequence of lifecycle phases a deliverable goes through is called the deliverable lifecycle. A product is fully specified and released for production, when all deliverables for the product are released for production. By focusing on the creation of information, the product development process is modeled as a flow of deliverables.

The generic simulation model is based upon the concept that product development consists of a workflow of engineering activities, which creates a set of deliverables that describe a product. The information specifying the product is classified into so-called domains. Domains can be regarded as separated product views in the product development process. Some examples of specific domains are quotation domain, requirement domain, functional domain, technical domain and physical domain. In each domain different kinds of information specify the product. Two deliverables from two different domains can contain information about the same product but from a different viewpoint.

Modeling the flow of deliverables starts with the decomposition in parts of the product that has to be specified by the deliverables. From this decomposition the flow of deliverables can be generated automatically. The final product, the components, the sub-components and the parts are the elements of the product structure of a product.

This principle is based on research performed by [Elzen, 1999] en [Jans, 1999] and continued by [Coolen, 2000].

3.2.2 Collaborative engineering

Concurrent engineering, and thus collaborative engineering, is translated in minimum phase difference and the propagating effect.

The 'minimum phase difference' describes the degree of concurrent engineering, which is applied during the development process. Every precedence relation between deliverables has a minimum phase difference. A minimum phase difference of two between deliverables 'one' and 'two', means that deliverable 'two' can be initiated (or can enter phase one) when deliverable 'one' enters phase three. Next, deliverable 'two' can enter phase two when deliverable 'one' enters phase four and so forth.

Each precedence relationship between deliverables can be set to 'propagating'. If an error is found in the preceding deliverable of the relationship, then the descendant deliverables also get a cycle and return to its first lifecycle phase 'in work'. Propagation in a concurrent engineering environment implies that errors may cause a lot of rework.

3.2.3 Data model

The following data model is the basis for the RapidPDM simulation model [Coolen, 2000]:

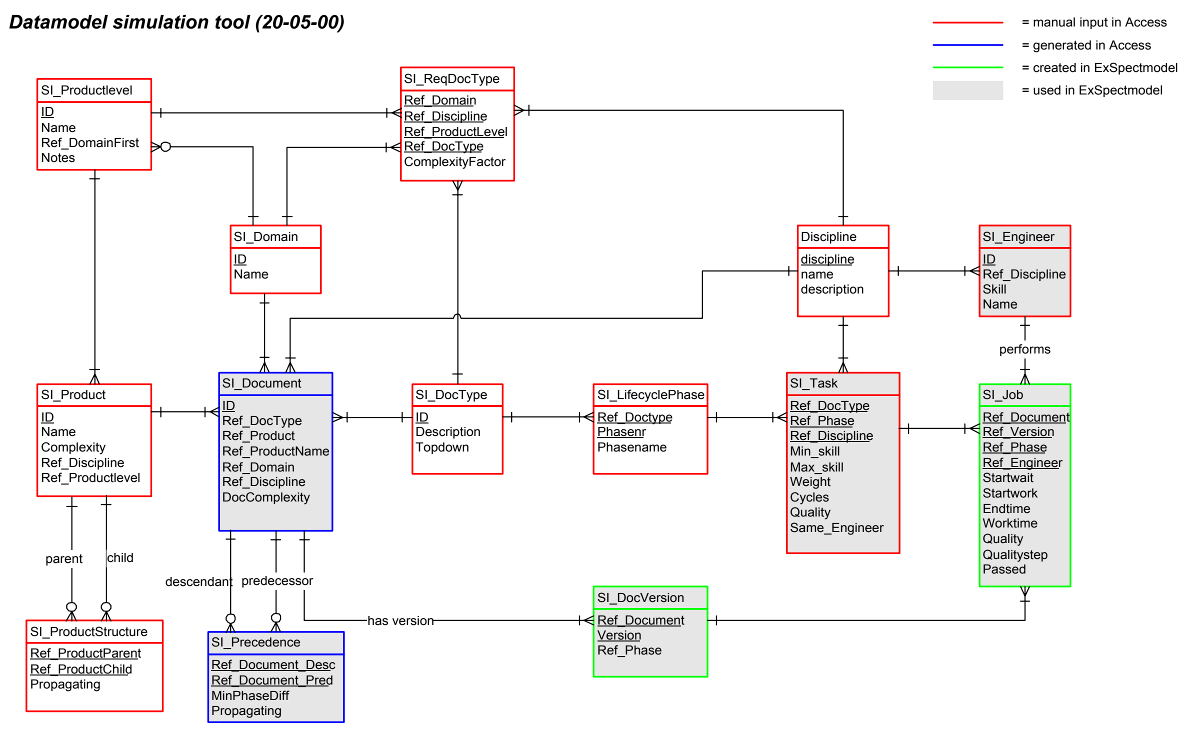


Figure 3.1: RapidPDM Data model

The main part of this data model consists of five input tables (see also Appendix B): Deliverable (Document), Task, Precedence, Engineer and Parameter_effect (not shown in the data model). The other tables are used to optionally generate the flow of deliverables specified by the Deliverable and Precedence tables (Product, Product Structure, Product Level and Required Deliverable Type) or for consistency in the Access tables (Domain, Deliverable Type, Lifecycle Phase and Discipline). The five main input tables are passed to ExSpect. In ExSpect the entities Deliverable Version (Document version) and Job are created for the execution of a simulation run. The results of the simulation in ExSpect are exported to Excel. In Excel the ExSpect output is processed and visualized.

3.2.4 Conceptual process model

This section contains an informal description of the process model that was used during the RapidPDM project.

The model starts with a set of deliverables in the initial phase. The execution of a task is called a job. A task on a deliverable can start when all preceding deliverables have reached a certain lifecycle phase. An engineer is assigned to the job, and a post time delay is calculated. Post time is the time an engineer has to wait until a deliverable arrives at his desk after being sent by the previous engineer. For the first job of a deliverable a search time is calculated. The search time is the time needed to collect all the information necessary to create the deliverable. Next the job is started. A job in process can be interrupted by a job with higher priority, for example an edit task can be interrupted by a review task. If a job is finished, two things can happen. The job finished successfully and the corresponding deliverable is updated to the next phase of its lifecycle. Otherwise, for example the job is rejected by the discovery of an error during a review, the corresponding deliverable returns to the first phase of its lifecycle. If a deliverable has reached its last phase, it is released. The model does not support parallel or coupled tasks (section 2.1.3), because the deliverable is automatically promoted to the next lifecycle phase after a task has completed.

Every place in figure 3.2 contains exactly one, possibly empty, set of jobs. Initially all deliverables are in the place 'Set of Deliverables'. When a new job is created, one deliverable is removed from this place and a job is created in the place 'Set of Jobs'. The transition 'Release deliverable' can only fire when all the tasks that belong to that deliverable have finished successfully.

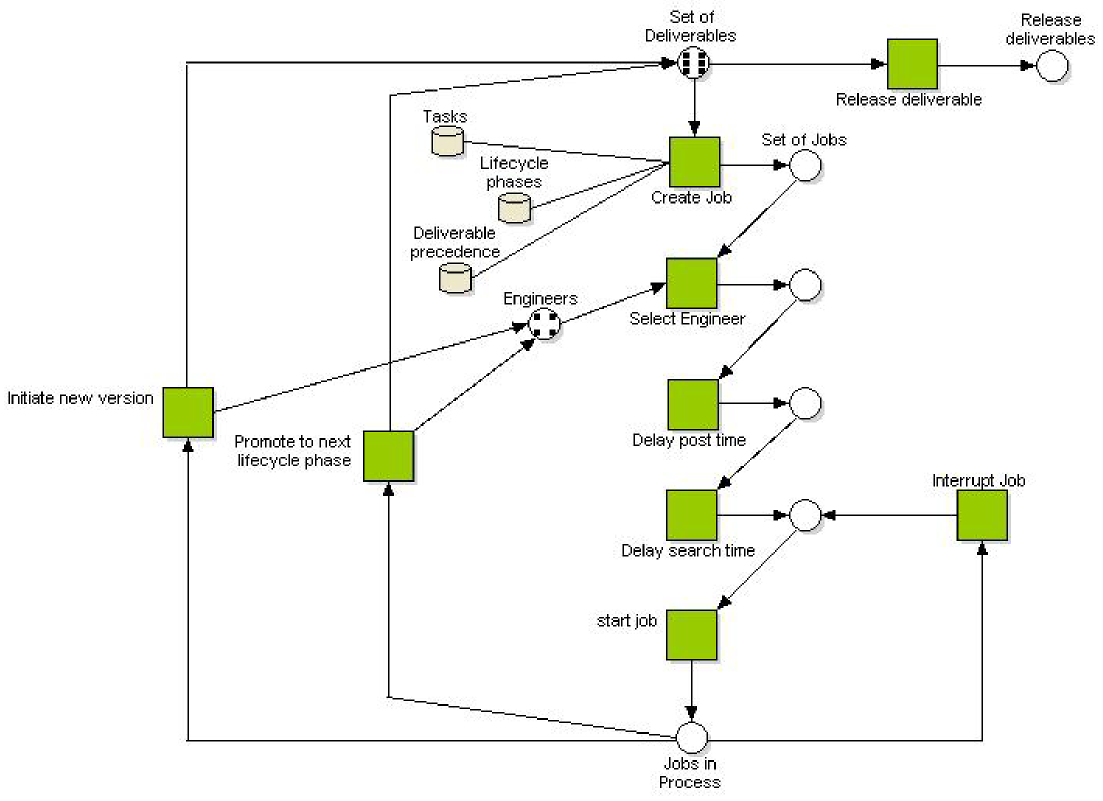


Figure 3.2: Conceptual Process model (RapidPDM)

3.2.5 Engineering characteristics

The Exspect model developed by Elzen and Jans contains the following engineering concepts:

Domain: Every deliverable belongs to a certain engineering domain (for example: functional specification, detailed specification). If the deliverables are generated from the product structure, each (part of a) product has a deliverable in every domain.

Post time: Time between the moment an engineer finishes a task and the moment it enters the work queue of the engineer performing the next task. Post time between tasks is specified in the precedence relation.

Search time: Time it takes for an engineer to collect all the preceding deliverables that are needed for his task. Only the first task of a deliverable has search time. The search time is linearly dependent on the number of preceding deliverables.

Discipline: Every task and every engineer has a discipline. The discipline, together with skill, is used to determine which engineer performs a task.

Skill: Every task has a skill level and every engineer has a minimum and maximum skill level. An engineer can perform a task when the required skill level of the task lies between his minimum and maximum skill level.

Deliverable generation: If the product structure is entered in Access, the deliverables and precedence rules can be automatically generated. Every (part of) a product has a deliverable in every domain.

Quality: The quality of a deliverable is created only in the first task. If a new version is created (caused by a rejection in a review), additional quality is created in the first task of the later versions.

Priority: A review task has priority over an edit task. Therefore an edit task can be interrupted by a review task. When the review task is completed, the engineer continues the edit task.

Complexity: Every deliverable has a complexity, from which the work times are calculated

Reuse factor: Reuse is the amount of information from previous products that is reused for the current product. If reuse is used, a deliverable enters the first phase, with a specified quality.

Deliverable type: Every deliverable belongs to a certain deliverable type. The deliverable type defines the lifecycle a deliverable goes through.

Weight: Every task from a lifecycle has a weight factor. This weight factor is used to determine the work time of a task. The sum of the weights of a lifecycle has to sum to 1.

Cycles: Every task except the first of a lifecycle can have one or more cycles. The number of cycles of a task specifies how many times the deliverable is rejected by that task. Two cycles cause three versions of a deliverable.

Same engineer: the person who created the deliverable should not perform a review. Therefore same engineer can be specified as ‘same engineer’, ‘not same engineer’, or ‘indifferent’. Same engineer always refers to the previous task.

Extra work time: If a new version of a deliverable is to be initiated while an engineer is working on the first phase of that document, then the document does not get a new version. Instead the task in progress in the first phase is extended with the time that would have been needed in the first phase of the next version. Extra work time only takes place when there is no 'input adjustment'. Because then first the input has to be adjusted before the work time can be extended.

3.3 Implementation

3.3.1 Input parameters

The model contains five main input tables implemented in Microsoft Access, which are used in ExSpect: deliverable, task, precedence, engineer, and global variables. Besides it contains additional input tables, which are used for consistency in access and for optional generation of the deliverables a precedence table. They are not used in the Exspect model. The output of the Exspect simulation is exported to an Excel document for further manipulation. Some standard visualization is generated in an Excel macro.

3.3.2 Process model

The process model is implemented in ExSpect. ExSpect contains a workflow model that is able to handle document lifecycles. It is a hierarchical model consisting of three components. One component processes the input from Access and one component generates the information needed for the output to Excel. The middle component is the process itself and consists of three parts. The first part generates jobs from tasks, the second part simulates the execution of a job and the third part handles rejected tasks.

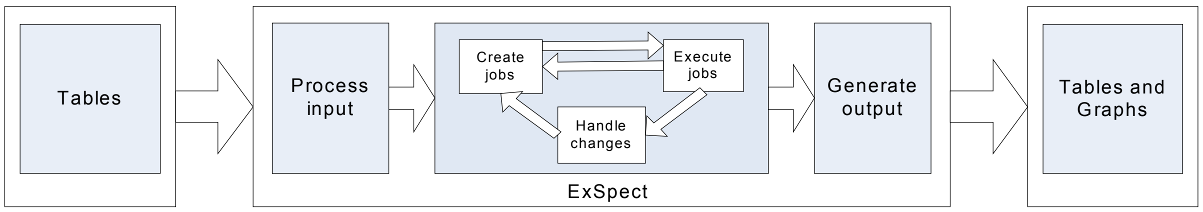


Figure 3.3: Exspect implementation

The Exspect model contains the implementation of an 'abstract' task that handles sets of deliverables (or sets of jobs). For example: the transition 'assign engineer' from the Exspect model has an input places containing a set of all jobs that are ready to have an engineer assigned. The 'assign engineer' transition then takes from this set all jobs for which an engineer is available and puts these jobs in the output place containing a set of jobs who have an engineer assigned and can be started. All other jobs from the input set are left in this set. Because every transition works on all jobs simultaneously, the model becomes complex. Every transition must be able to deal with all exceptions that can occur in any deliverable.

3.3.3 Output parameters

The RapidPDM model uses the three most commonly used parameters as output for the simulation: time, cost and quality. A graph is shown showing three lines: percentage released deliverables against time, worked hours against time and attained quality against time.

Time

Time is a critical factor in getting the product on the market as soon as possible. Smaller throughput times result in shorter time-to-market of a new product, resulting in more turnover for the producing company. The simulation model shows a graph of percentage of released deliverables against time. In this graph the complexity of a deliverable is taken into account, such that larger deliverables have more impact on the time graph.

Cost

The development cost of a product, expressed in engineering hours, has a large impact on the total product cost. Other costs, like equipment and software packages, can often be neglected. Cost is an essential factor in product development. The simulation model shows a graph of percentage of worked hours against time. Cost will always be higher than or equal to time, because all work that has been released (time) has also been done (cost).

Quality

Quality of the released deliverables is an important factor, while proper product data management decreases the number of errors in deliverables. Quality is defined as (Q-E)/Q, where Q is the total complexity of the deliverable and E is the number of remaining errors. For each deliverable the quality after the first version is specified and for each new version the percentage of remaining errors found is specified.

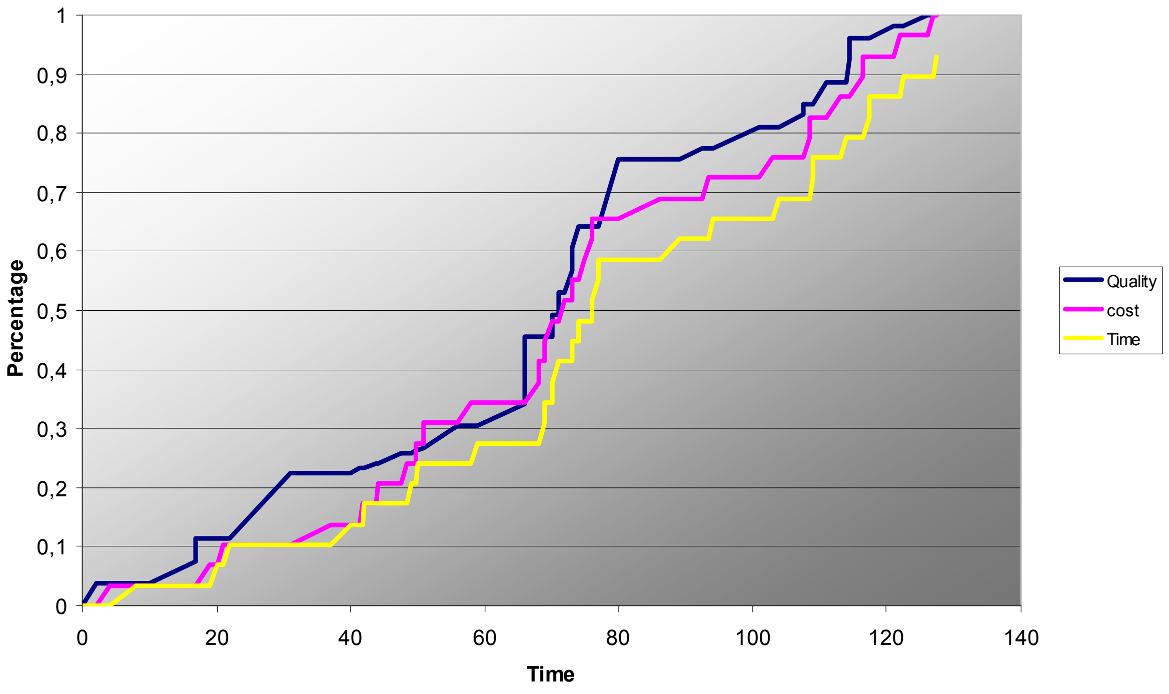


Figure 3.4: Example Excel output

3.4 Design considerations

3.4.1 Implementation approach

As stated in section 3.3.2, the implementation approach is based on one abstract task that works on sets of deliverables. The advantage is that the size of the model does not vary with the size of the process that is to be modeled. Only the input and output tables become larger. Also the user is not concerned with implementation details. All modeling is done in tables, giving a clear overview of the input parameters. The disadvantage is that the business model is hard coded in Exspect and a thorough knowledge of the implementation is needed to make small adjustments to the Exspect model. Another disadvantage is that the user can't 'see' the flow of deliverables during a simulation run.

3.4.2 Design decisions

The RapidPDM model has some shortcomings and some superfluous aspects.

1. No multi company support

The RapidPDM simulation model has no means of specifying engineers from more than one company.

2. No 'project phases'

The RapidPDM model specifies a domain, which can be interpreted as project phases. They are, however, not used in the ExSpect simulation. At this point it is not yet clear whether the project phases need to be incorporated in the model.

3. Deliverable and precedence generation

The model can generate the deliverable set and the precedence rules from the product structure. In practice this option is not used, probably because most companies do not have the same deliverable set for their products. For each company new deliverable and precedence generation rules have to be made to get a proper deliverable set. It is easier to enter the deliverable set directly.

4. Definition of quality

In the RapidPDM model the quality is calculated by means of a fixed percentage quality increase for each version of the deliverable. This produces a fixed value between 0 and 1 for the quality of the deliverable. This value, however, is difficult to interpret and does not contribute to a better understanding of the development process.

5. Discipline per deliverable

The RapidPDM model defines a discipline on the deliverable level. Therefore each task is performed by an engineer from the same discipline. The discipline should be defined on the task level, such that each task of a deliverable can require an engineer from a different discipline.

6. Reuse factor

The variable reuse factor appeared to be of no use in this simulation model. A different way of looking at reuse could be developed.

7. Affected deliverables

When a document is rejected, all the following documents will get a new version. However it is not taken into account that an input deliverable might have to be changed. In the enhanced model some sort of feedback leading to new versions of previous documents should be implemented.

8. Global post time

The RapidPDM model defines a global post time. In a multi-company situation, however, post times vary from deliverable to deliverable and even from phase to phase. Therefore post time should be defined on a lower level.

9. Availability of engineers

The RapidPDM model uses only engineers who work fulltime on one project. In reality this is not always the case. At this point it is not yet clear whether engineers who work part-time on a job need to be incorporated in the model.

10. Only one task per phase
In the RapidPDM simulation model the phase of a deliverable is automatically updated after the completion of a task. When two tasks are performed in parallel (section 2.1.3), the phase has to be updated only when both tasks are completed. In this case the set of parallel tasks is called the activity and the phase can only be updated after completion of the activity.

11. Complexity
In the enhanced model, complexity is not used. Instead work times are directly entered in the model instead of derived as a percentage of the deliverable complexity.

12. Deliverable lifecycle phases can not be skipped
Certain tasks are performed only once. For example a prototype may be build only once, because building a second version of the prototype may be too expensive. Therefore errors found in the first version are corrected in the product without a second prototype. Although risky this method does occur in practice.

13. Same engineer refers to previous phase
The simulation model can handle tasks that have a different engineer than the previous task. It would be more useful to indicate that a task (for example a review) cannot be performed by the same engineer as the first task (normally the edit task). The same engineer should thus refer to the first task of a deliverable instead of the previous task

14. Priority
The concept of priority is introduced in the RapidPDM model to be able to interrupt a job, when a more important job is to be performed. However this concept was never well-defined. For example: questions like 'is it always the case that a review has a higher priority than an edit-task?' and 'is it possible for every task to be interrupted by a task with higher priority?' remain unanswered in the documentation of the model. Therefore this concept will be reevaluated.

15. Search time
The original model has search time as a separate parameter. This parameter was used to show the benefits of a PDM system. This parameter is of no use for general performance measurement, because the time to search for input documents is incorporated in the work time of a task/deliverable.

16. Skill level
Skill will not be incorporated in the new model. In most cases the skill level of an engineer will not be used. In the models where the skill level is used, it is very easy to add this concept to the model, by adding a skill attribute to the engineer. Then the precondition of a start transition is strengthened to pick an engineer with an appropriate skill level.

4 Developing an enhanced model

4.1 Why start from the RapidPDM principles

The RapidPDM project was chosen as a starting point for several reasons. First of all there was an implementation at hand, which made it easy to directly experiment with different approaches to the design questions of the case that was used in the first half of this project. Design decisions could be validated in a relatively short period of time and if they were found to be less suitable, they could be altered. If a new model had to be designed from scratch, it would have taken much longer to build a complete, working prototype of the model and it might not have been possible to test the model on the case, due to the time span (only the first half of this project) of the case.

Besides the RapidPDM model was well documented and a lot of knowledge was available at the information systems section of the department of Technology Management. Hardly any information was found on other attempts to model product development processes and if there was information available it was scarce.

During the case study it seemed only possible to model the basics of product development processes using standard workflow language constructs. Much of the complexity of these processes can only be modeled either by laborious modeling or by a complex process. The last option is what this project aims at by creating building blocks with most of the complexity within the blocks, such that the user can built a model out of 'black boxes' and is hardly concerned with this complexity within these boxes.

4.2 A different modeling approach

The RapidPDM model is implemented in ExSpect. However, ExSpect is not used in the normal way. Instead of directly modeling each task as a transition or sub process, there is exactly one abstract implementation of a task. This abstract implementation is based on sets of jobs that are in the same state. For example the transition ‘assign engineer’ does not assign engineer A to job ‘edit’ of the deliverable ‘specification’, but assigns engineers to all jobs that are in the state to have an engineer assigned. ExSpect, using a functional programming language with constructs for handling sets is quite suitable for this way of modeling. There are some disadvantages.

The abstract task has to deal with all the possible exceptions that can happen during the handling of one task, making the abstract task very complex. In itself this is not a problem because the user of the model is not concerned with the implementation in ExSpect. He just sees the input in MS Access and the output in MS Excel. But if he wants to make small changes to the underlying business model, he needs deep inside knowledge of the ExSpect implementation. This makes it hardly impossible for the average user to do anything more with the model than what was implemented at design time. The complexity of product development processes and the fact that the model has to be applicable in many different engineering environments (section 1.2) require small adaptations on the business model to be possible. Therefore this project aims at a solution based on a standard workflow approach.

The standard workflow approach offers another advantage. The process can be tracked, because every deliverable has its own transitions. The user can simulate the model step by step and see what happens. This provides the user with more insight in the model he is simulating.

The main disadvantage of the standard workflow approach is that it easily causes performance problems. The complexity of the building blocks and the number of blocks needed may cause a simulation to load and run very slow. This problem is inherent to this approach.

4.3 Simulation engine

Two simulation engines have been evaluated: ExSpect and CPN Tools. ExSpect and CPN Tools are academic simulation tools. These alternatives were chosen because of the familiarity and knowledge of these tools at Information Systems groups at the department of Computer Science and the department of Industrial Engineering and Management Science at Eindhoven University of Technology.

4.3.1 ExSpect (Deloitte, TU/e)

The simulation package ExSpect was chosen for the development of the RapidPDM generic simulation model for product development processes. This choice was based on criteria mentioned in [Jans 1999]. Regarding the developed simulation tool it can be concluded that ExSpect is not used in the usual manner because the engineering process itself has not been modeled but the control of engineering processes has been modeled. The choice for ExSpect offered the following advantages for the development of the simulation tool [Jans, 1999]:

- Storage of (complex) data elements. The generic simulation model requires storage of data structures and ExSpect is very well suited for that purpose.
- Defining complex behaviors. With the use of the ExSpect programming language it was possible to define complex behaviors of the processors. The generic model does not contain many processors because most of the complexity of the model can be included in the processor relations.
- Graphical view of the output. Performance measurement of the simulation model is shown graphically.
- Easy change procedures of the input settings. The different levels of PDM functions can easily be set before the start of a simulation run.

However, while using the ExSpect model it came clear that the Exspect implementation is too slow to perform stochastic analysis. A single simulation run, takes about 30 seconds for a small to medium size development process. Also, ExSpect is not under active development anymore. Therefore a different simulation package is considered.

4.3.2 CPN Tools (University of Aarhus, Denmark)

CPN Tools is considered as an alternative for ExSpect. The main reason why CPN Tools is eventually chosen is that it is under active development and is actively used at the Department of Technology Management at Eindhoven University of Technology, in contrast to ExSpect which has not been under development for some years.

Also, the first two advantages of ExSpect mentioned in 4.3.1 hold also for CPN Tools. The third advantage appeared to be of little use, because the graphical output was mainly generated from the Excel datasheet and not directly from the Exspect. Excel offers a much better environment for processing output data. CPN Tools is also able to output its data to Excel, although in a more laborious manner. The fourth advantage does not hold for this project because the different levels of PDM are not of interest in this research.

4.4 Research for improvements

This paragraph describes how the enhanced model will deal with the shortcomings and redundancies discussed in section 3.4. These improvements originate mainly on research performed by [Benders, 2004]. During his assignment the RapidPDM simulation model was adopted to form a prototype of the enhanced simulation model. Therefore some features were already implemented and this prototype supported in the research on the process model.

4.4.1 Implementation approach

The enhanced model will use the 'normal' workflow method to model the product development processes. This means that each task or deliverable has its own transitions and places and that the tasks and deliverables are connected by linking places. All input parameters are now set in the corresponding place, transition or arc, instead of in the five central input tables that were used in the RapidPDM model. A standard implementation of a single task or parallel tasks will be available but also standard implementations of a lifecycle containing one, two, three or four sequential tasks. The user of this tool can then use these building blocks to make his own complete model of an entire development process in much less time than would be needed if he had to design the model from scratch.

4.4.2 Design Decisions

1. No multi company support

In the enhanced model a 'company' will be defined. Engineers belong to exactly one company. Within the companies different roles will be distinguished that engineers can fulfill.

2. No ‘project phases’

The pilot project of [Benders, 2004] showed no need for the introduction of project phases. Also Project phases have no influence on the outcome of a simulation. Therefore there are no plans for incorporating project phases in the enhanced model at this moment.

3. Deliverable and precedence generation

The product structure and deliverable and precedence generation will be omitted in the enhanced model. In the enhanced model the deliverables and precedence rules will be entered manually. Precedence rules will be entered by means of a 'required phase'. A 'required phase' belongs to a task and specifies that another deliverable should have reached at least a specified phase. A task can have more than one 'required phases'.

4. Definition of quality

Quality was defined as a quality percentage that is added in each task. Modeling quality this way assumes that the quality of a document (or the number of errors in a deliverable after execution of a task) is known beforehand. Some reports more recent than the RapidPDM project speak of content instead of quality. Each task then can add content to a deliverable. The case project did not show any need for a parameter like quality. Because little knowledge is available on the parameter quality in this environment, it is not used in the enhanced model. If desired, a quality like parameter can easily be derived from the output file.

5. Discipline per deliverable

The enhanced model will specify disciplines that belong to a company and can be fulfilled by an engineer. Every task has to be performed by an engineer from a specified discipline. This makes modeling more accurate than modeling the discipline on document level of the RapidPDM model.

6. Reuse factor

The parameter 'reuse factor' was not used in the case project. In the RapidPDM project it was used in an attempt to show the benefits of a PDM system on the reuse of deliverables of previous projects. Because not enough knowledge is available on how to incorporate 'reuse' in a simulation model, it will not be implemented.

7. Affected deliverables

Affected Deliverables specify which input deliverables have to be adjusted when an error is found in a document. When a deliverable is rejected in a certain task, it will get a new version, but also the affected deliverable gets a new version. The propagating effect, where later deliverables get a new version when older deliverables are rejected, will be preserved.

8. Global post time

The RapidPDM model defines a global post time. In multi-company situation, however, post times vary from deliverable to deliverable and even from phase to phase. Therefore post time should be defined on a lower level. The new model implements post time on the task level.

9. Availability of engineers

The availability of engineers will not be a separate concept in the enhanced model. If it is needed in can be simulated with the existing concepts. If a situation is simulated in which an engineer needs time before he starts a task, then this time slack can be modeled using post time. If he needs eighty hours for a task that requires forty hours of work, because he has to perform other tasks within that time span, then the work time of the task can be set to eighty instead of forty. The output file can still export forty as the actual work time.

10. Only one task per phase

The enhanced model will contain separate building blocks for 2, 3 or 4 parallel task. These building blocks can then replace one of the single task activities in the predefined lifecycles.

11. Complexity

As mentioned in section 3.4.1 complexity will not be incorporated in the enhanced model. Work times are directly entered in the model and not as a percentage of the complexity of the deliverable.

12. Deliverable lifecycle phases can not be skipped

Certain tasks are performed only once. For example a prototype may be build only once, because building a second version of the prototype may be too expensive. Therefore errors found in the first version are directly incorporated in the final product. This method is risky, but does occur in practice. In the enhanced model this can be modeled as en exception to the normal situation. For example, by an extra transition that skips a task in the second version.

13. Same engineer refers to previous phase

The RapidPDM model can handle tasks that have a different engineer than the previous task. The enhanced model does not support this construct directly. However it can be modeled as an exception by keeping a history of engineers for a certain task and setting a precondition on the engineer selection.

14. Priority

Priority, interrupt a job to start working on another job, will not be incorporated in the enhanced model. The case showed no need for the concept. It is assumed that an engineer finishes his task, before he starts on the next task. This concept can not be modeled as an interrupt, because no new version is created and the same task is restarted which is a fundamental difference to a normal interrupt handling. Implementing it would require a complex administration of work queues of engineers. Future research is needed to find a way of implementing priority in the enhanced model, if the need for it arises.

15. Search time
The original model has search time as a separate parameter. This parameter was used to show the benefits of a PDM system. Search time has a trivial effect as it is defined as time per input document times the number of input documents. This parameter is of no use for general performance measurement, because the time to search for input documents is incorporated in the work time of a task/deliverable. Therefore the global parameter 'search time' will be removed in the enhanced model.

16. Skill level
The skill level of an engineer is not incorporated in the enhanced model. In the general case the skill level will not be used and it is a feature that is easy to add afterwards.

4.5 Output parameters

From the original output parameters time, cost and quality (section 3.3.3) quality has been shown to be of little use (section 4.4.2). In the case described in [Benders, 2004] the output parameter quality was not used at all. Because no useful alternative was found for the quality parameter it will be omitted in the enhanced model. If needed, it can always be easily derived from the output generated by CPN Tools.

The enhanced model will write a log to the file results.xls that can be read with MS Excel. This log file contains all the start, end and work times of all tasks. From this information the total work time and total throughput time can easily be derived. Also when other cases demand other output parameters, they can probably be derived from the excel output. If this is not the case then additional log files can be generated from CPN Tools. Because of the lack of more test cases, no additional output parameters were found for the enhanced model at this time.

4.6 Stochastic behavior

The Exspect model is not capable of simulating stochastic behavior. All input variables are deterministic. Even if every input variable would have a parameter specifying the distribution, it would probably be unfeasible to perform stochastic analysis, because the length of a single simulation run. One run of a medium sized simulation takes already 30 seconds. For stochastic analysis to be useful, many runs of the same simulation are necessary to come to reliable results.

CPN Tools appears to be much faster in simulation such that stochastic analysis might be possible. Besides it is easier in the CPN Tools approach to specify different stochastic behavior for different deliverables.

5 Conceptual design

This chapter describes the toolbox on an abstract level. Implementation issues will be discussed in the next chapter.

5.1 Overview

The enhanced model will consist of standard building blocks. These building blocks can be copied (cloned in CPN Tools) from the file containing these blocks to the model. The main building blocks will contain one complete lifecycle, such that one building block corresponds to one deliverable.

Two types of relations exist between deliverables: triggers and interrupts. Triggers start work on deliverables and interrupts stop work on deliverables. On the highest level the lifecycles (deliverables) are connected with interrupt places. Triggers are not modeled as tokens, but as preconditions on the triggered deliverable. Thus, all arcs and places on the highest level represent only interrupts.

The model is designed to be a generic set of building blocks for product development processes. It is not the intention that these building blocks cover all possible features of a development processes. The user can easily append most features to the model. The model would become unnecessarily large and unmanageable, when all possible features would be incorporated in the building blocks. For example: the building blocks define only one discipline per engineer. Some cases, however, might require engineers that belong to several disciplines. The user of the model can then define a second 'discipline'-attribute for an engineer and use it in the model.

5.2 Data model

This section describes the entities and relationships between them that are used in the model and in this report.

5.2.1 Enhanced data model

The following model contains the entities that are used in the enhanced model:

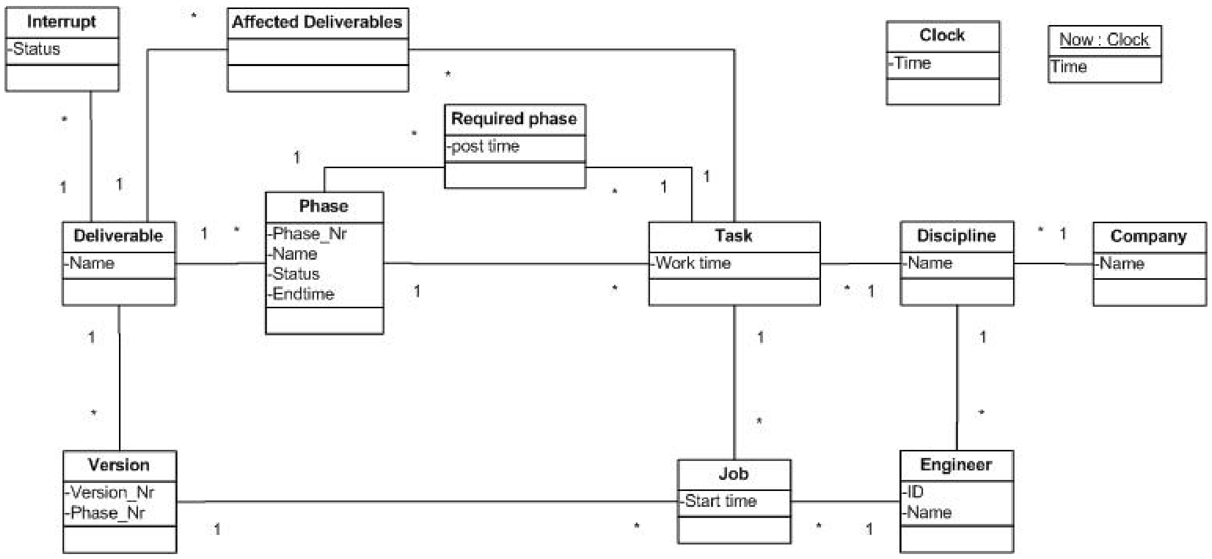


Figure 5.1: Data model

A 'Deliverable' is identified by a unique name and can have several versions. An 'Interrupt' is created by one deliverable and has a status. A 'Version' belongs to one deliverable and has a version number and a phase_Nr. A deliverable has one or more phases. A 'Phase' belongs to one deliverable and is characterized by a deliverable and a phase number. A phase also has a name, a status and an end time. Different tasks can be executed in parallel or sequentially during a phase. The collection of tasks in one phase of a deliverable is also called an activity (not shown in figure 6-1). A 'Task' has a work time. The 'Required phase' defines which status of the input deliverables is needed for a task to start. A required phase relation can have a post time. A 'Discipline' is identified by its name. Each task has a discipline according to which an engineer is selected. A discipline belongs to one company and can be fulfilled by one or more engineers. Each 'Company' is identified by its name. An 'Engineer' has an ID, a name and a discipline. Engineers with more than one discipline are not part of the standard building blocks. An engineer can be responsible for executing a number of jobs. A 'Job' is executed by one engineer and is executed on one version of a deliverable. The class 'Affected Deliverables' is used to indicate which input deliverables are affected by a rejection of a task. The class 'Clock' has exactly one instance 'Now' and is used to denote the current model time.

5.2.2 Differences with RapidPDM data model

The most important differences to the RapidPDM data model are the following:

- Entities Productlevel, Product, Productstructure and ReqDocType are not needed
- Entity Domain is not used
- Document is now called a Deliverable
- Precedence is replaced by Required_phase
- DocType is not used
- LifecyclePhase is now called Phase
- DocVersions corresponds to Version
- Task, Discipline, Engineer and Job remain the same
- Company is added in the enhanced model
- Affected Deliverables is added in the enhanced model

5.3 Conceptual process model

5.3.1 Processes, lifecycles, activities and tasks

The simulation toolbox is divided in three parts: process, lifecycle and task. As stated earlier, product development processes produce deliverables that specify a product. The process forms the highest level of the hierarchy. The deliverables are the building blocks of the process and form the second layer of the hierarchy. The lifecycles of the deliverables consist of one or more tasks. The tasks are specified on the lowest level: the task level.

To minimize the modeling effort for the user of the tool, different standard lifecycles will be created to be used as standard building blocks of a model. For example a lifecycle consisting of 1, 2, 3 or 4 sequential tasks will be part of the modeling toolbox. Besides there will be different types of tasks, for example tasks consisting of 1,2,3 or 4 parallel tasks will be available in the toolbox to create more complex lifecycles.

In the description of the conceptual model a short-hand notation is used to specify the behavior of the conceptual Petri Nets. This short-hand notation does not intend to give a formal specification of the model. The goal of this chapter is to give an understanding of how the model works without using implementation terminology and implementation details.

`'in(job)'`	is defined as 'the token named job is consumed from input arc job'.
`'no(interrupt)'`	is defined as 'no token is available on arc interrupt'
`'job -> accept'`	is defined as 'the input token job is copied to output arc accept'.
`'new(interrupt)'`	is defined as 'a new token interrupt is created in the transition'.
`'out(job)'`	is defined as 'the token named job is produced on output arc job'.

Functions are used to check for model properties. For example: the boolean function "`required_phase(job)`" indicates that all input deliverables have to be in the required phase. The meaning of functions is defined right after the specification.

In the pictures below, the small boxes in the lower left corner of a transition indicate a subnet. A small box in the lower left corner of a place indicates that the place is a global place.

Below is a fictitious example of a task to demonstrate the notation:

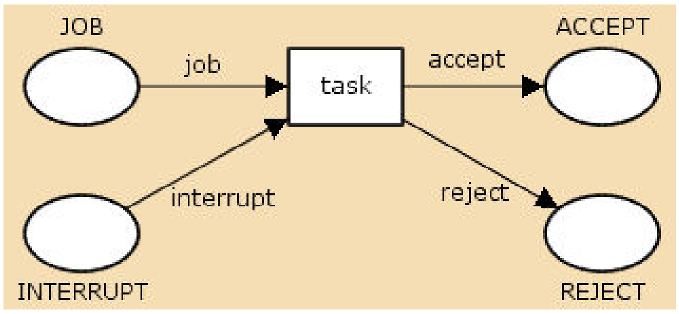


Figure 5.2: Example conceptual net

specification	Explanation:
Task:	
`PRE: in(job)`	Fire as soon as a token is available on arc job.
`if in(interrupt) then`	If an interrupt token comes in then...
`  job.status = rejected`	set job status to rejected and...
`  job -> reject`	copy input token job to output arc reject.
`else if accept(job) then`	Else if the job finishes successfully...
`  job -> accept`	copy input token job to output arc accept.
`else if not accept(job) then`	Else if the job finishes unsuccessfully...
`  job.status = rejected`	set job status to rejected and...
`  job -> reject`	copy input token job to output arc reject
`end`	

Function definitions:
`accept(job) = The job finishes successfully`

5.3.2 Process

The process is the highest modeling level in the toolbox. At this level the deliverables are specified and the interrupt relations between the deliverables are defined. Although there are two kinds of relations (precedence and interrupts) there is only one place between two successive deliverables: an interrupt place. Precedence relations are defined on the task level. This means that the Petri Net on the process level does not specify the order in which the deliverables are being created. This results from the fact that precedence is defined on tasks and not on deliverables. However, if a precedence relation exists, then there will normally also be an interrupt relation. This preserves the visual aspect of the order in which the deliverables are created.

Thus, on this level, deliverables only produce interrupt tokens. Normally this interrupt token is passed on to the next deliverable in the process, but it can also be passed back to a preceding document (see affected deliverables, section 3.4). This way, interrupt routing is taken care of on this level.

Example (figure 5.3): a process consisting of three deliverables. First a specification is created. No initial marking is needed on this level for the specification to start. Based on this specification, the design is made. Eventually a prototype of the design is manufactured. This example does not imply that the specification has to be finished before the design can start. Precedence of tasks and the degree of concurrency is defined on the task level. If the prototype is rejected, then a new specification is made followed by a new design and a new prototype. The arcs only indicate interrupt relations between deliverables.

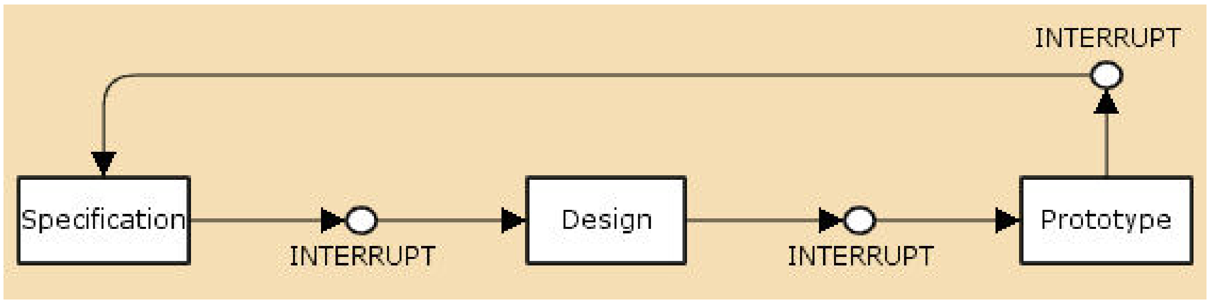


Figure 5.3: Example of a process

5.3.3 Lifecycle

A process consists of deliverables. Each deliverable has a lifecycle. A lifecycle has a starting place 'start' and a finishing place 'released'. However, a token in the released place can always be withdrawn by an incoming interrupt token. In between these two statuses, the lifecycle tasks are performed.

A lifecycle has an incoming interrupt place and an outgoing interrupt place. An incoming interrupt always has to be handled instantaneously. After all, the successive deliverables have to be notified of the interrupt as well. Therefore the incoming interrupt place is connected to each one of the tasks (subnets) in the lifecycle. Whatever phase the deliverable is in, the interrupt can be handled immediately by that lifecycle phase or by the withdraw transition.

A task can end in an accept, which means that the deliverable is promoted to the next phase, or in a reject. If a task ends in a reject, then a new version is created and an interrupt out token is generated. On the process level, this interrupt out token can be passed on to one or more deliverables.

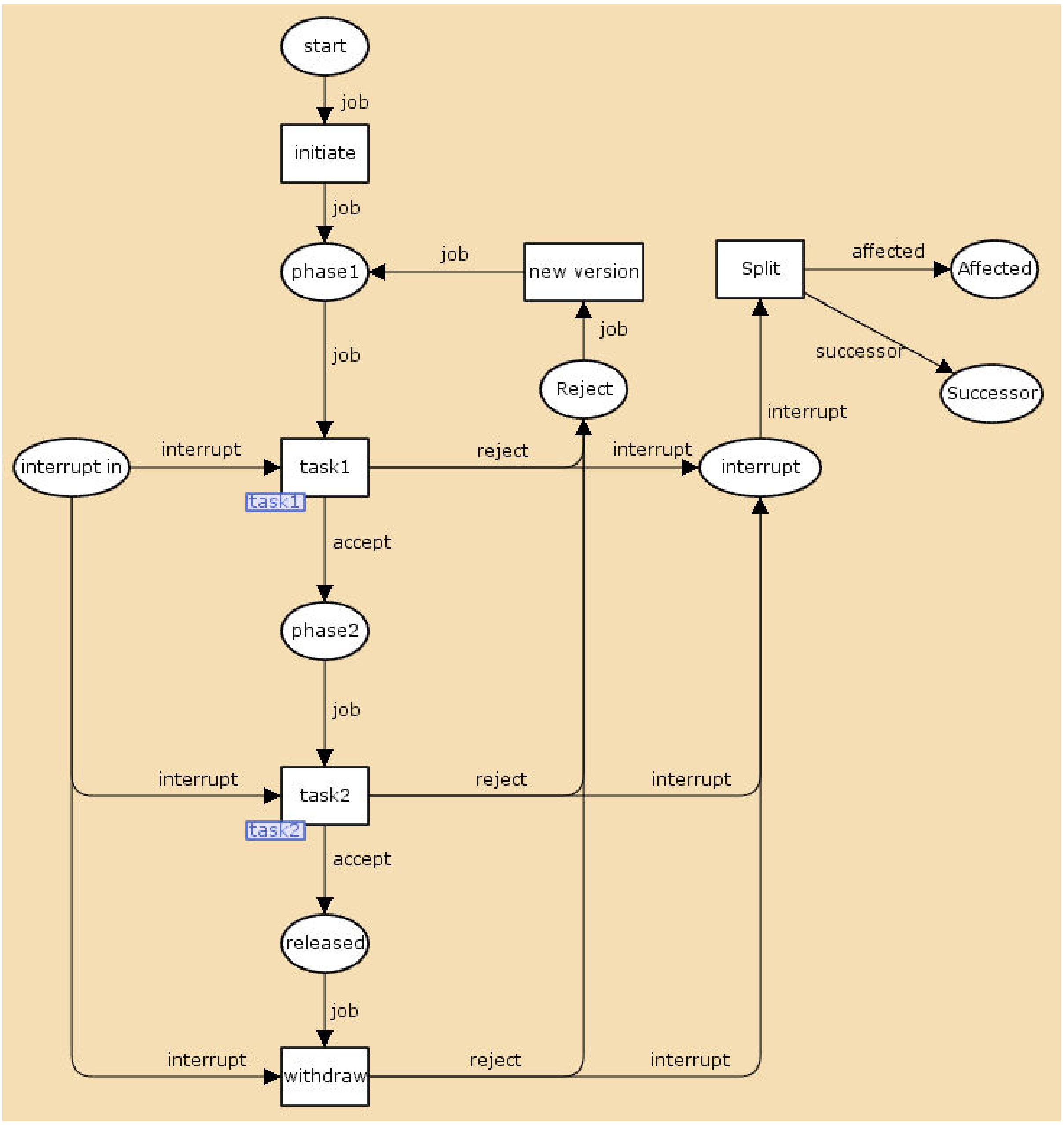


Figure 5.4: An example lifecycle

```
Initiate:
PRE: in(job)

  job.version = 1
  job.phase = 1
  job -> job

Task1, Task2 (subnets):
PRE: in(job) and required_phase(job)

  if in(interrupt) then
    job -> reject
    interrupt.status = interrupted
    interrupt -> interrupt
  else if no(interrupt) and accept(job) then
    job -> accept
  else if no(interrupt) and not accept(job) then
    job -> reject
    new(interrupt)
    interrupt.name = job.name
    interrupt.status = rejected
    out(interrupt)
  end
```

Withdraw:

```
PRE: in(job, interrupt)

  job -> reject
  interrupt.status = interrupted
  interrupt -> interrupt
```

New version:

```
PRE: in(job)

  job.version = job.version + 1
  job.phase = 1
  job -> job
```

Split:

```
PRE: in(interrupt)

  if interrupt.status = rejected then
    interrupt -> affected
    interrupt -> successor
  else if interrupt.status = interrupted then
    interrupt -> successor
  end
```

Function definitions:
required_phase(job) = all input deliverables have reached the required phase.
accept(job) = The job finishes successfully

During a task an interrupt can come in at any time during execution. Therefore 'in(interrupt)' should be read as: 'At some point during the execution of the task an interrupt comes in on arc interrupt'.

5.3.4 Activity

An activity is the set of all tasks that are performed in a lifecycle phase. In most cases an activity will contain only one task and then only the notion of task will be used. Only when a lifecycle phase contains more than one task, the notion of an activity consisting of different tasks is used. For example when two reviews are performed in parallel there is one activity 'review' and this activity consists of two tasks 'review1' and 'review2'. In this case the tasks 'review1' and 'review2' are also called subtasks.

5.3.5 Task

A task is the lowest level of the conceptual model. A basic task is modeled as a start and a stop. In between an interrupt can occur. Before a task has started it can be cancelled. Every task requires one or more engineers. On the task level most of the working details of the simulation model are specified. For example: constraints can be put on the discipline of the engineer and the work time is specified by constraints on the stop-action.

A task has two input places: start and interrupt in. A token in place start means that the task can start as soon as the precondition holds. A token in 'interrupt in' means that the task can be cancelled or interrupted. There are three output places: accept, reject and interrupt out. When the task completes successfully, a token is produced in place accept, when the task is completed unsuccessfully or is interrupted or cancelled, a token is produced in the places reject and interrupt out.

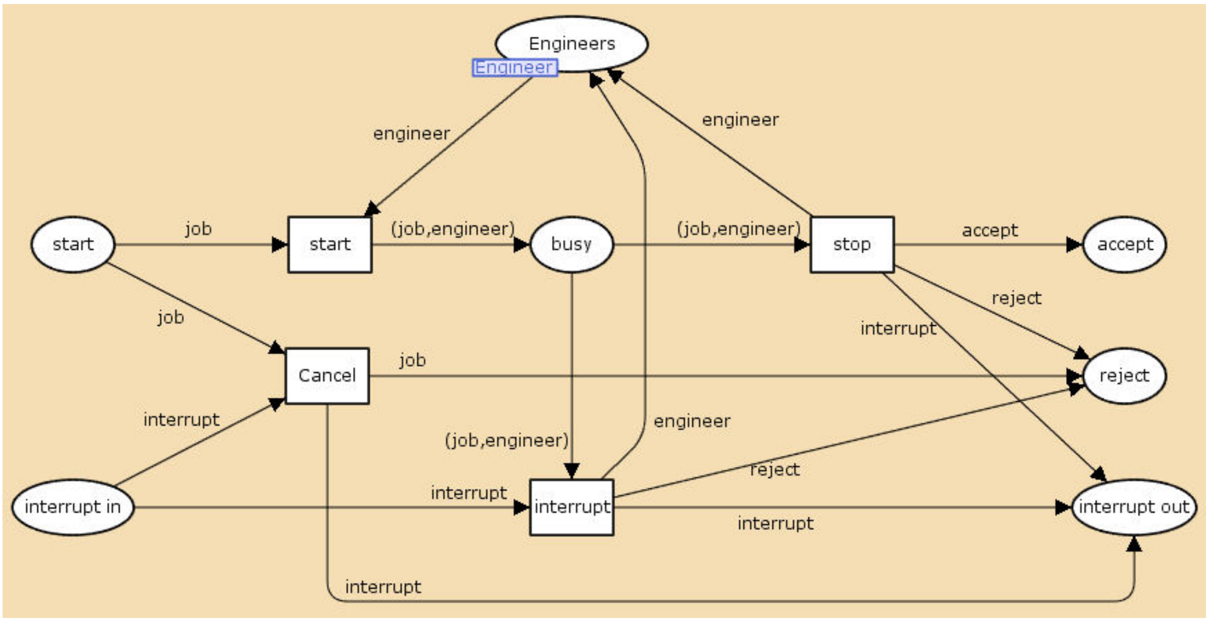


Figure 5.5: A conceptual task

The behavior of a task is specified as follows:

```
Start:
PRE: in(job, engineer) and
       required_discipline(job, engineer) and required_phase(job)

job.starttime = now.time
job.endtime = now.time + job.worktime
job, engineer -> (job, engineer)

Stop:
PRE: in((job, engineer)) and job.endtime = now.time

if accept(job) then
  job -> accept
  engineer -> engineer
else if not accept(job) then
  job -> reject
  new(interrupt)
  interrupt.name = job.name
  interrupt.status = rejected
  out(interrupt)
  engineer -> engineer
end
```

```
Cancel:
PRE:   in(job, interrupt)

job -> job
interrupt.status = interrupted
interrupt -> interrupt

Interrupt:
PRE:   in((job, engineer), interrupt)

job -> reject
interrupt.status = interrupted
interrupt -> interrupt
engineer -> engineer
```

Function definitions:
required_discipline(job, engineer) = This engineer is capable of performing this job
required_phase(job)= All input deliverables have reached the required phase for this job
accept(job) = This job finishes successfully

The scope of the place "interrupt" is the entire deliverable. If a token in "interrupt" is not consumed by a firing of a transition in this task, it will be consumed by a firing of a transition of another task within the same deliverable.

5.4 Sequential, parallel and coupled tasks

In section 2.1.3, the concepts of sequential, parallel and coupled tasks were introduced. This section describes how these concepts are incorporated in the conceptual model.

Sequential tasks are modeled in the process lifecycle. Take two tasks A and B from the same deliverable. Suppose that task A has to precede task B and task A has to be finished before task B can start. task A will be in an earlier lifecycle phase than B. In this case task A will precede task B in the lifecycle. Task B can only start when there is a token in its input place, produced by task A (or one of the tasks that can be executed between A and B, but certainly after task A has finished).

Note that this definition is a little different from the definition in section 2.1.3 from Ulrich and Eppinger. That definition states that task B can finish after task A has finished. This implies that task B can start before task A has finished. This makes a simulation almost impossible, because every task can possibly start at any time, including the beginning of the simulation.

Parallel and coupled tasks are modeled within an activity. Parallel and coupled tasks imply that within one lifecycle phase, more than one task has to be finished before the deliverable can be promoted to its next lifecycle phase. In the case of parallel tasks, the tasks are performed without interaction and independent of each other and in the case of coupled tasks, the tasks are performed (partly) simultaneously and with some sort of interaction. This interaction can take place in different forms, for example:

- task A and B must start simultaneously
- task A and B must start simultaneously and end simultaneously
- task A has to start before task B can start
- task A has to finish before task B can finish
- combinations and variations (more than two tasks) of the previous examples

The model will contain standard building blocks for parallel tasks, but not for coupled tasks. The reason for this is that there are too many variations of coupled tasks and these variations require only slight changes from the parallel task building blocks. For example if two tasks have to start at the same time, the precondition has to be strengthened such that the tasks can only start if both tasks are enabled. Introducing extra building blocks would make the set of building blocks unnecessarily complex. The user (a researcher at Eindhoven University) is expected to be able to model such slight changes.

5.5 Interrupts

Every deliverable has exactly one input interrupt place. Because, in the case of affected deliverable, cyclic interrupt paths are possible, the origin of the interrupt has to be checked. When the incoming interrupt was caused by the deliverable itself, then the interrupt is ignored. This ‘own interrupt’ can only occur in the first phase of a lifecycle.

An incoming interrupt has to be handled instantaneously. A lifecycle or a task can not be finished before the interrupt is handled, because successive documents have to be interrupted by the same interrupt token. This interrupt thus has to be passed on to the successive deliverables at the same time as it arrived at a deliverable. Because the lifecycle of a deliverable does not know which of its tasks is being executed and therefore does not know where to send the input interrupt, it will leave the interrupt in the input place. The task that is being executed at the time on which the interrupt arrives is responsible for handling the interrupt. Because at any given time at least one task is being executed or the 'withdraw' task is enabled, the interrupt will always be handled immediately.

6 Implementation

6.1 The process model

For a better overview the implementation is discussed bottom up from task to lifecycle instead of top down like the conceptual model in chapter 5.

6.1.1 Tasks

This section describes the various aspects of the task implementation. It uses parts of the CPN model to illustrate the implementation. A complete task implementation can be found in Appendix C3.

Basics and interrupts:
A task in a product development process can be interrupted at any time. For example: a task can be interrupted by an error found in an input document. The standard way to model this situation in Petri Nets is to model a start and a stop transition. The 'start' transition puts a token in the 'busy' place with a timestamp equal to the current model time plus the work time of the job. However, the token in the 'busy' place is not available whatsoever before the job has ended. An extra trigger place is needed to enable the stop transition when the job is finished, such that the job token itself is put in the 'busy' place without a timestamp to make it always available for an interrupt. When an interrupt occurs the trigger token is left behind and has to be removed once it has become available.

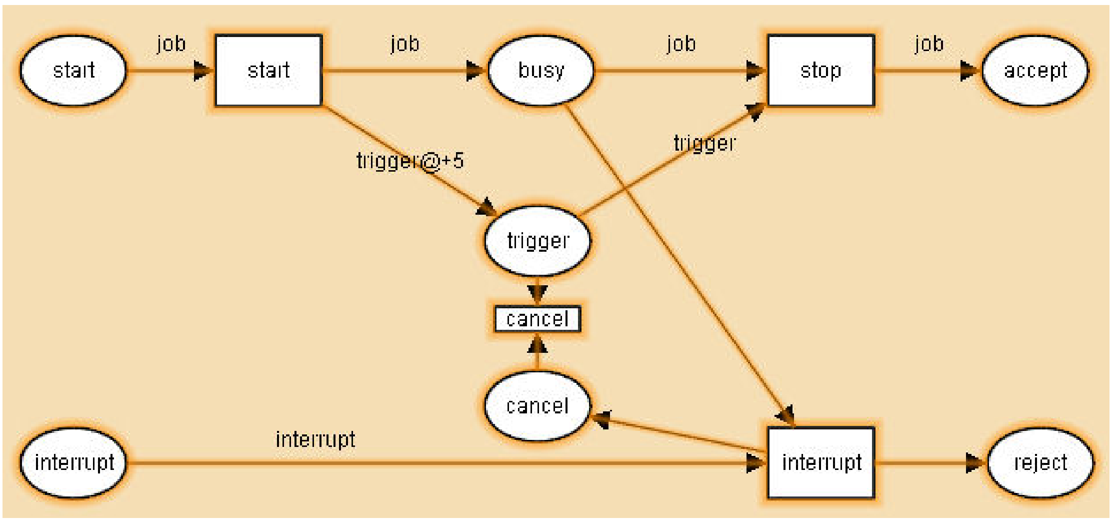


Figure 6.1: Interrupts

Own interrupt
An ‘own interrupt’ is an interrupt that originates from the deliverable where it arrives. This can only occur in the first phase, because before an interrupt is sent out, the interrupt generating deliverable gets a new version and thus will be in the first phase when its own interrupt comes back as an incoming interrupt. This is because interrupts are always handled instantaneous, i.e. no time passes during the moment the interrupt is generated and the moment the interrupt as an entity disappears from the model.

Differences to the conceptual task model
In the conceptual model a task has three output places: accept, reject and interrupt. In the implementation the places 'reject' and 'interrupt' are joined together. The two places are replaced by one place of the type 'product'. Product is a standard construct that combines

two data objects in one place in CPN Tools. Because a token is always produced or in both places or in none, the model becomes simpler when these two places are combined.

The conceptual model has one stop transition. The implementation has a separate accept and a separate reject transition. This was done to simply the model and to easily specify when a reject takes place. Rejections are specified by the preconditions of the start and reject transitions.

In the implementation an extra transition 'restart' is introduced. A restart is a special case of an interrupt, only used in the first phase of a deliverable. This restart has the same behavior as an interrupt except that no new version is created. The restart was not modeled in the conceptual model to keep the conceptual model orderly. The output of a simulation would be the same when a restart was modeled as an interrupt. The only difference is that in that case an extra version would be created. The restart also solves an implementation problem. If a start and cancel transition of one task are enabled simultaneously, then a non-deterministic choice is made. In real life the cancel would always take place. The only solution that would not harm the neatness of the model is to model a restart in the first phase of a lifecycle.

Cancel
A cancel is equal to an interrupt except that the job hasn't started yet. In case of a cancel, the token is taken from the place 'Start' instead of the place 'Busy'. Also no engineer is put back in the place 'Engineers'. A cancel, like an interrupt, produces a reject token and an interrupt token. The first task of a lifecycle has no cancel. Instead incoming interrupts are ignored as long as the task has not started yet.

Restart instead of complete interrupt handling
Sometimes when a job has only just started theoretically (current time is still equal to the start time) and an interrupt comes in, a new version is not necessary. The engineer can just restart his job. This is only the case when it is the first task of a deliverable. Instead of doing a complete interrupt handling, a 'restart' is sufficient. This means that the job is returned to 'begin' and the engineer is made available. Because a time trigger was already set out to trigger the stop transition, a token is put in the place 'for cancel'. In a standard building block this restart can only fire when the task was only theoretically start, which means that the current model time is equal to the start time of the model. If an engineer can also restart the task after, for example, one hour of work, then the guard on the transition restart has to be weakened.

Parallel tasks do not have a standard restart option, even if the parallel tasks belong to the first version. The reason is that the restart of one task in an activity can influence the other tasks. All tasks in the same activity get the same interrupt, but it is possible that one task is waiting for an available engineer where the other task has already finished. Then the first task is restarted, where the second has already left the task. Because a situation where a restart of parallel task is necessary is not likely to occur and modeling them would make the standard building blocks unnecessarily complex, they're left to the user of the tool.

Note: a restart can only occur in the first phase of a deliverable. If an interrupt occurs during a later phase then first the earlier phases have to be reworked, before work on the later phase can be resumed.

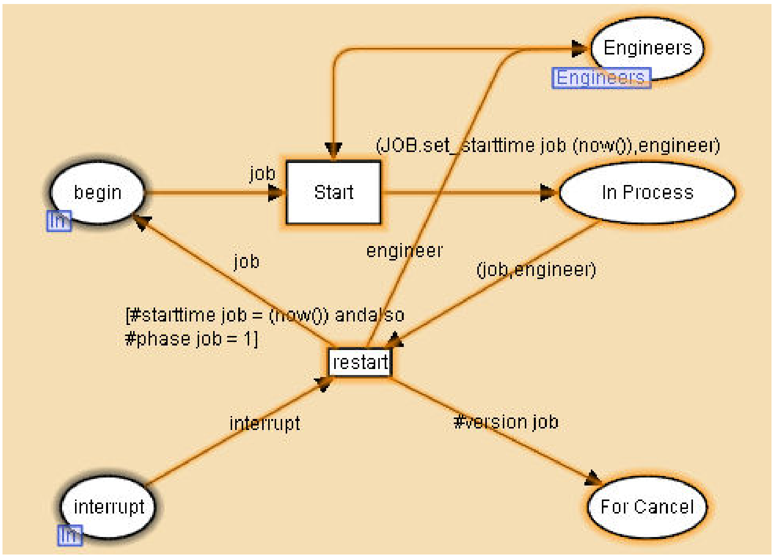


Figure 6.2: Restart a task

Engineers

Engineers are implemented by means of two global places: 'Engineers' and 'Busy Engineers'. In the basic model without task priority, engineers can also be modeled by a single place. But when an engineer can be taken of one task to start work on another task, it is easier to have one global place where all busy engineers are available during simulation.

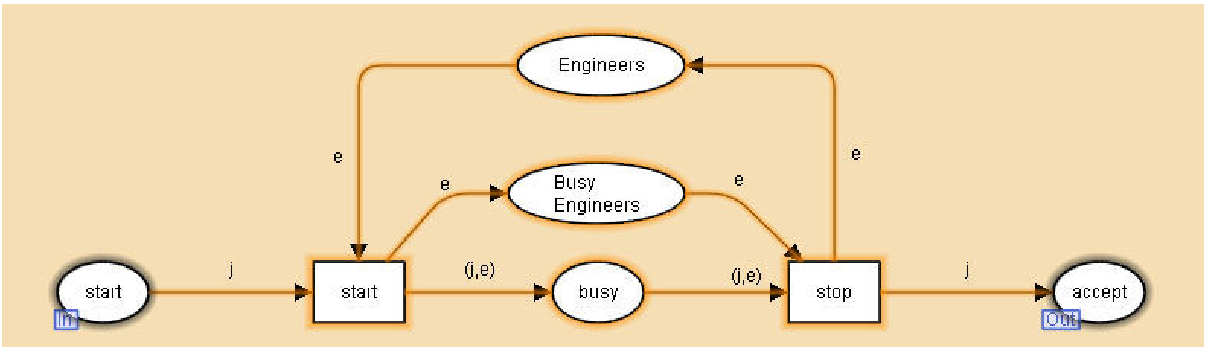


Figure 6.3: Engineers

Role

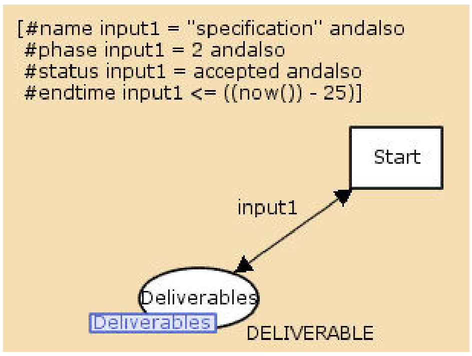
Roles are not explicitly modeled in the enhanced simulation model. Roles are defined as the combination of a discipline and a company, but in the enhanced model a precondition can be put on a start transition specifying which discipline is needed. Therefore the concept role is not incorporated in the model. A problem only occurs when one engineer can take on different roles. In that case a workaround is possible. Instead of one discipline an engineer can have two or more disciplines or maybe even a list of disciplines. Because this situation is not likely to pose problems, it is not modeled in the standard building blocks.

Required phase

Required phase is implemented by means of preconditions on the start transition of a task. The place 'Deliverables' keeps track of the statuses of the lifecycle phases of all deliverables. Based on the status of its input deliverables a new task can start.

6.1.2 Post time

Post time can be modeled based on preconditions on the start transition. The fusion set 'Deliverables' contains not only the status of all deliverables, but also the timestamp on which the last work on the phases of that deliverables was done. Based on that information post time can be put in the precondition of a start transition.



```
[#name input1 = "specification" andalso
 #phase input1 = 2 andalso
 #status input1 = accepted andalso
 #endtime input1 <= ((now()) - 25)]
```

Figure 6.4: Post time precondition

There is however a problem when a moment during the development process exists where no task is being performed but there is a task waiting for a deliverable to arrive. The entire system is waiting for the post time to pass. However, in CPN Tools a simulation is terminated when there are no more tokens with a timestamp in the future and no transition is enabled. This situation is the case when the system is waiting for a post time to pass. The system is waiting for a precondition to become valid, but time does not advance. Therefore a construct is needed to advance time when post time is holding the system.

The workaround in the simulation model is that an 'accept' transition puts a token named posttime in the place 'posttime' with timestamp equal to the current time plus the post time, whenever the deliverable is posted after that task. On the main page of the process, a transition takes this dummy token and replaces a token in the deliverable place that is part of a fusion set. Because all start transitions depend on this deliverable place, they are all rechecked for enabling. Note that this workaround does not take many resources, because it uses only one extra firing per post time.

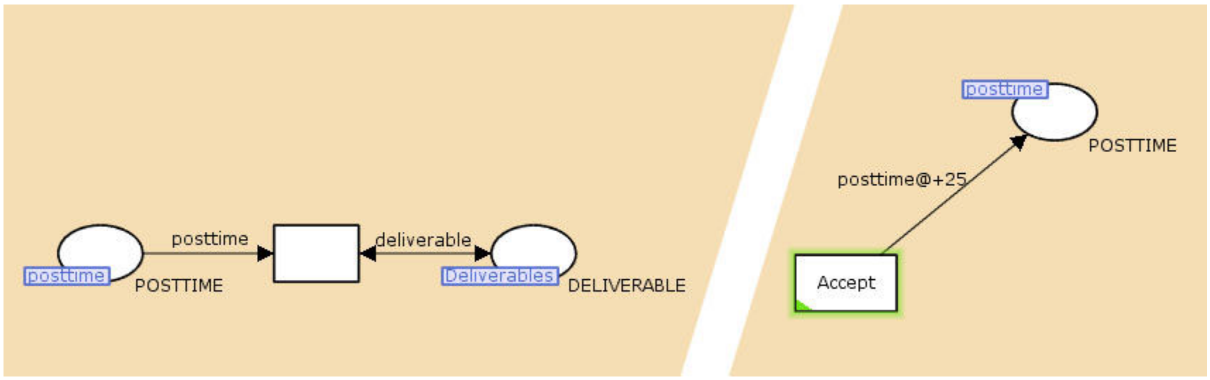


Figure 6.5: Post time

Work time reduction

Work time reduction is not modeled in the standard building blocks. It is easy to add the concept of work time reduction to the model using an if...then...else... construct on the timestamp of the token that triggers the 'accept' and 'reject' transition.

Exchange method
Exchange method is not needed as a standard structure in the enhanced model. In the enhanced model it is possible to define a constant representing an exchange method. This constant can then be used instead of the post time value, making the exchange method an unnecessary construct.

Same engineer
"Same engineer" is not modeled in the standard building blocks, because the assigning of an engineer is already done per task and it is easy to incorporate the "same engineer" principle, using an if...then...else... construct.

Version 1 only
Version 1 only is not modeled in the standard building blocks, because it is easy to add to the model. If a task is to be performed on the first version only then the start task can be restricted to the first version only by strengthening the precondition. An additional transition can be added that will skip the entire task and promotes the deliverable to the next lifecycle phase.

Number of cycles
The number of cycles a task goes through is specified by the preconditions on the accept and reject transitions. If for example a task is rejected in the first version and accepted in the second and later version then the precondition of the accept transition would be strengthened by 'version > 1' and the precondition of the reject transition would be 'version = 1' instead of 'false'. This way it is even possible to accept a task in the first version, reject in the second version and accept the task in the third version.

6.1.3 Parallel tasks
The implementation of parallel tasks is a similar to that of single tasks. The differences originate from the extra administration needed for synchronization. A complete implementation can be found in Appendix C4.

Control layer
The main difference between single tasks and parallel tasks is that, because of the extra administration, an extra hierarchical layer is introduced. The implementation consists of a ‘control’ layer and a ‘work’ layer. In the control layer the activity is split into subtasks and interrupts are handled. It is assumed that tasks in one activity belong together. This means that they are never separately interrupted and acceptation or rejection depends on all subtasks. It is also assumed that all subtasks have the same start condition and therefore the precondition is specified in the control layer. It is possible to put the precondition in the subtask if this assumption appears to be not valid in certain situations.

Many different variations exist for proceeding to the next lifecycle phase in the case of parallel tasks. For example, the deliverable can be promoted when all subtasks have finished, when one subtask has finished or when a specified subtask has finished. In the building blocks it is assumed that the deliverable is promoted when all subtasks are accepted. Variations are left to the user of the model.

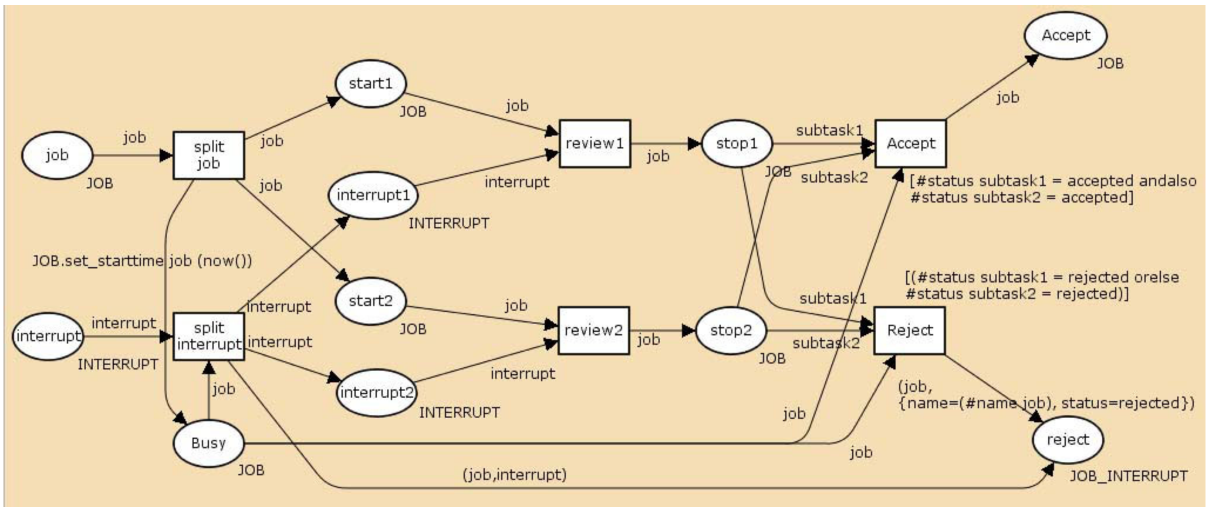

job
JOB
split job
start1
JOB
review1
stop1
Accept
JOB
interrupt1
INTERRUPT
[#status subtask1 = accepted andalso #status subtask2 = accepted]
JOB.set_starttime job (now())
interrupt
INTERRUPT
split interrupt
start2
review2
stop2
Reject
[(#status subtask1 = rejected orelse #status subtask2 = rejected)]
interrupt2
Busy
(job, {name=(#name job), status=rejected})
reject
JOB_INTERRUPT
(job,interrupt)

Figure 6.6: Parallel tasks, the control layer

Work layer

The implementation of the subtask is much simpler because most administration is done at the control layer. There are two main differences between a subtask and a single task. The subtask has one stop transition where the single task has a separate accept and reject transition. If a subtask is rejected in some version then the arc from Stop to Accept has to be adjusted by the user. The control layer does have a separate accept and reject. The second difference is that there is no interrupt out. The propagation of interrupts is handled in the control layer and therefore the subtask can just reject the job and drop the interrupt in case of an interrupt.

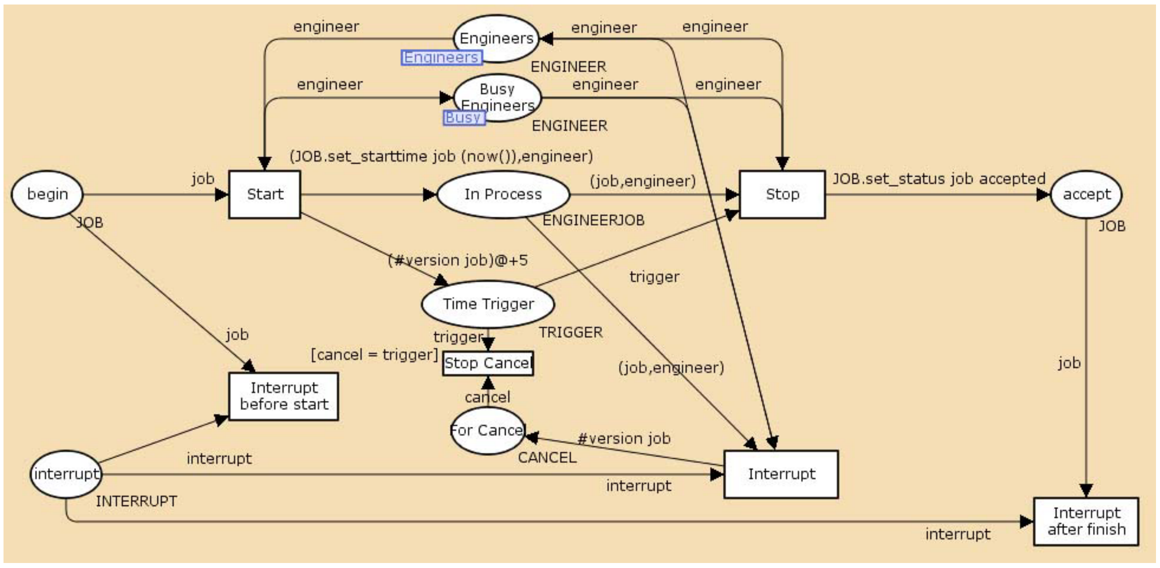


Figure 6.7: A subtask

6.1.4 Lifecycles

A complete implementation of a lifecycle is shown in Appendix C2.

Triggers modeled in 'data'

Triggers are modeled using one single global place containing all statuses of all deliverables, instead of a separate place for every trigger. Trigger tokens in product development processes are too complex to create an easy model with one place per trigger.

If triggers were modeled using tokens, then it would be as in figure 6.8. Two alternatives of implementing Figure 6.8 are discussed below:

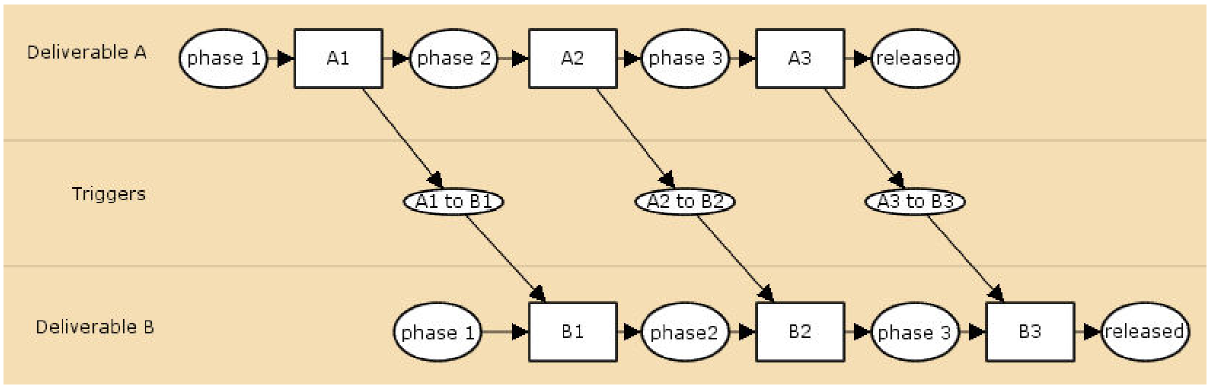


Figure 6.8: A general example of triggers modeled as tokens

ALTERNATIVE 1
If B3 creates a new version of deliverable B then B2 has to start without a trigger from A2. Thus, a new version doesn't need triggers, unless of course a new version is caused by a change in document A. Summarizing:

New version by document itself -> don't wait for trigger (or ‘remember' the trigger)
New version by other document -> wait for new trigger

But when a task has several input documents it has to administer which one of those N documents caused the new version and wait for a new version for that document. This gets problematic when more than one input document generates a new version. The job has to administer a list of withdrawn triggers.

But this causes the problem of the implementation of the transitions that sometimes do and sometimes don't use a token from a trigger place. A transition can only fire when all of its input places contain a token. This causes many different start transitions, for every possible combination of triggers, or one very complex transition, dealing with the variety of possibilities of triggers, especially when one task has more than one input trigger.

ALTERNATIVE 2
The trigger is preserved in the corresponding trigger place. When deliverable A gets a new version it has to remove the trigger, such that deliverable B will need a new fresh trigger. But when task A1 and task A3 also have triggers, then there many possibilities for the number of triggers to be removed. One has to model some sort of vacuum cleaner that would considerably harm the neatness of the system. Also these triggers and the removal of them can not be build inside the standard building blocks and thus have to be build by the user of the toolbox, making it very user unfriendly.

SOLUTION
This all can make the relations between deliverables quite complex. But when building standard building blocks, one just want to make the interfaces between the blocks, the part that has to be build for every model, as easy as possible. Therefore it was chosen to model precedence relations as data instead of as trigger tokens. This implies there is one global place containing all statuses of all deliverables.

This way precedence is specified by a precondition on a transition instead of as an input token on a transition. These preconditions are based on a global place, containing the status information of all deliverables. This global place will be updated automatically within the standard building blocks and therefore requires much less modeling by the user of the tool.

Experience indicates that most of the complexity of precedence relations is solved by using 'data modeling' instead of a technique based on trigger tokens. This is caused by the absence of support for optional input arcs in Petri Net based tools and in this case in CPN Tools. But even when optional input arcs were supported, it will still be easier and more orderly to use the data modeling approach.

A disadvantage of this method is that at the highest level, there are no trigger tokens going from deliverable A to deliverable B. This diminishes the advantage of a graphical modeling tool. But in the tool there are also interrupt tokens going from one deliverable to another. When a precedence relation (trigger) exists between two documents then there will, with a very high probability, also be an interrupt relation between the two delivers. This will introduce an arrow from the preceding deliverable to the successor deliverable reestablishing the visual aspect of precedence in the model. Besides if one wants to see a trigger token from deliverable A to B, one can model this token and subsequently ignore it in deliverable B

Propagation of interrupts
There are two types of output interrupts: affected and successor interrupts. Successor interrupts will always get an interrupt, when the current deliverable is rejected or interrupted. However, affected deliverables will only get an interrupt when the current deliverable is rejected and not when it is interrupted. Affected deliverables are only affected by rejections of the corresponding deliverable and not by interrupts of that deliverable.

When, for example, a prototype is rejected, there is chance that all or some input deliverables will have to be reworked. The deliverables that are directly affected will get an interrupt directly from the prototype. Possible successors of the affected input deliverables will get an interrupt from the affected deliverable. This way a treelike propagation of interrupts takes place. This tree like propagation can result in circular references, such that an incoming interrupt has to be checked on origin. When the origin of the interrupt is the deliverable where it arrives, the interrupt has to be ignored. Also an interrupt is ignored when the deliverable hasn't started yet and is still in the first phase.

When a task has produced an interrupt, it will create a new version of its deliverable. Work on this new version of the deliverable can normally start immediately. There can be a problem when there are affected deliverables. It can happen that rework on the deliverable starts before affected deliverables have updated their status indicating that the deliverable that caused the interrupt, is enabled. In this case the deliverable that caused the interrupt is started, but he is not allowed to start; only the status of the input deliverables has not yet been updated. Now a restart is needed.

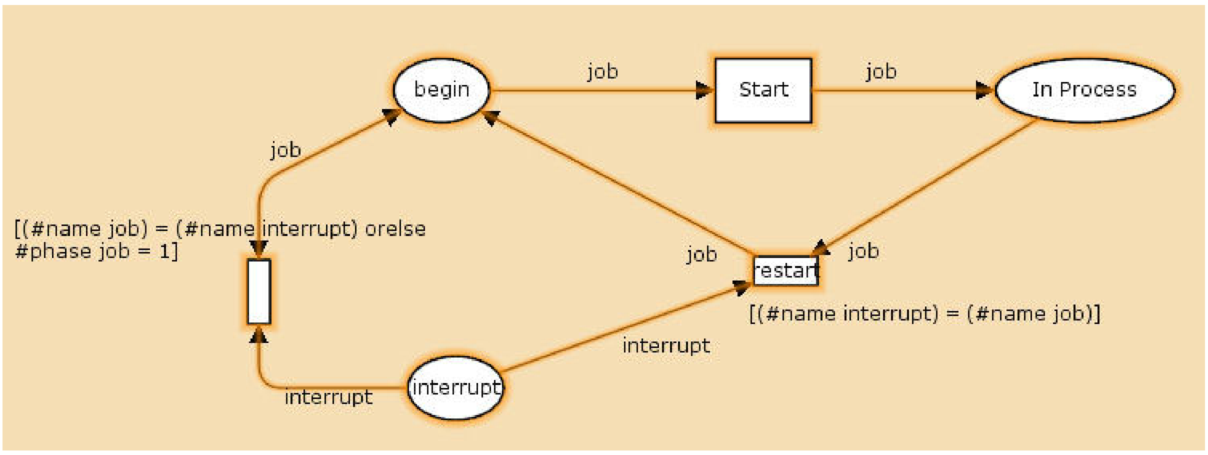


Figure 6.9: Interrupt exceptions

6.1.5 Process

A complete implementation of a lifecycle is shown in Appendix C1.

Why all documents are initiated immediately:

Normally, one would only initiate a deliverable when work on that deliverable can start. However there are several technical reasons for initiating all deliverables in the development process immediately:

1. No exception is needed to handle incoming interrupts before the deliverable was initiated
2. A guard on the initiate task is needed that is the same as the guard of the first task of the deliverable. The guard is certainly needed in the first task, because second and later versions also need the guard. This implies that the guard on the initiate task is superfluous.

6.1.6 Output

A simulation of a model created with the toolbox will automatically generate an output file named ‘export.xls’. An example export file is shown in appendix D. All start, accept, reject and withdraw transitions will automatically write a line to this log file. Visualizations in the form of graphs are left to the user.

6.2 User Interface

Section 6.1 describes how the model was built in CPN Tools. This section describes how to build a model, starting from the building blocks described in section 6.1. It is assumed here that the reader is familiar with the basics of the user interface of CPN Tools. Information on the CPN Tools basic user interface can be found at:
http://wiki.daimi.au.dk/cpntools-help/getting_started_with_cpn_.wiki

1. Open an 'empty' model

The first action is to open a new model containing all the declarations that are needed. Open 'example (empty).cpn' and save it with a desired filename. The 'example (empty).cpn' opens the following window. It contains one binder on which the 'process' page is displayed, without any deliverables or tasks.

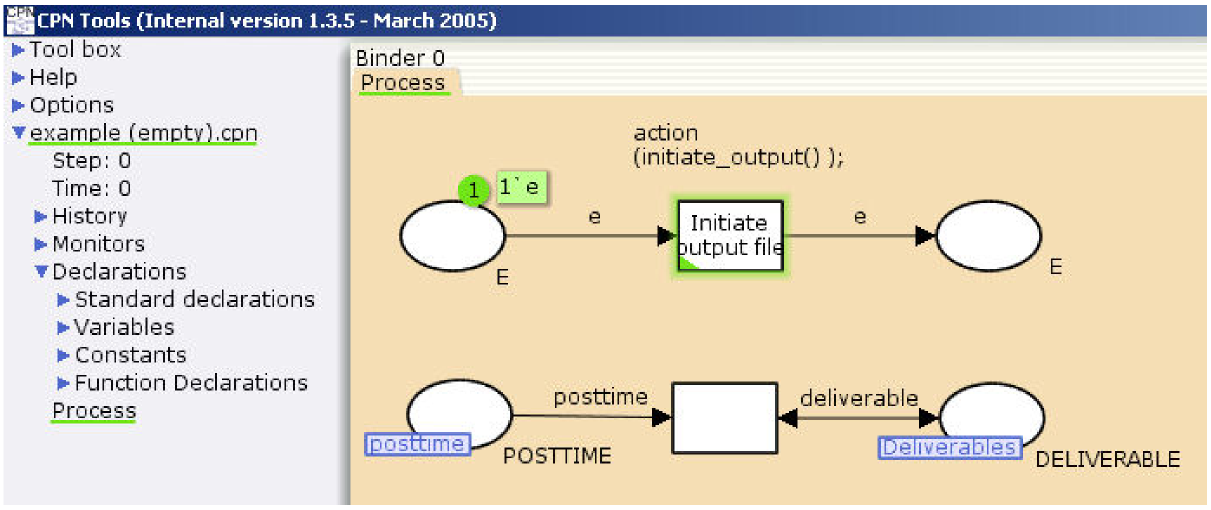


Figure 6.10: An empty model

2. Add deliverables

To start adding deliverables to the newly created model, first the 'building blocks.cpn' file needs to be opened. The 'lifecycles' page appears. Select the appropriate group, for example '2task', and clone the corresponding lifecycle and place it in the new model. Repeat this step for every deliverable.

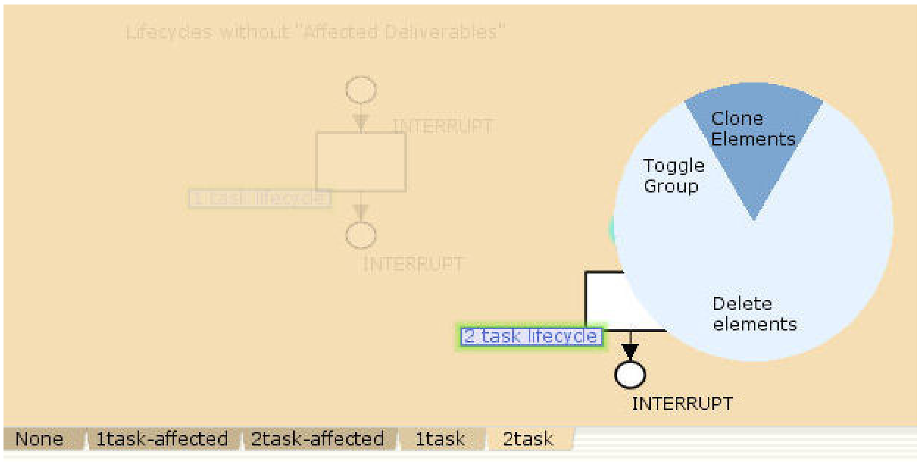


Figure 6.11: Clone a lifecycle

3. Connect interrupt chain

Define the interrupt relations. The following figure shows an example of a net where deliverable1 is followed by deliverable2 and deliverable3. If deliverable3 is rejected then an interrupt is passed to deliverable1. Note again that at this level the tokens only represent interrupts, not triggers.

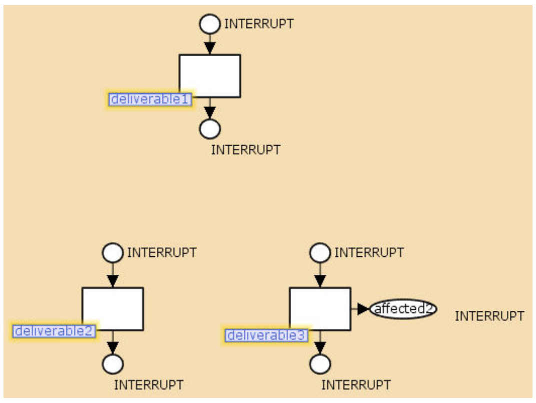


Figure 6.12: Before

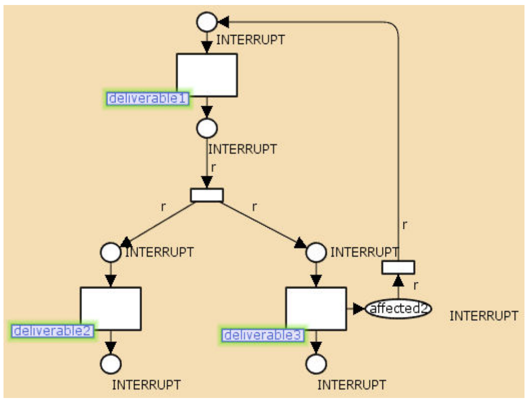


Figure 6.13: After

4. Adjust the model

Now almost all transitions, places and arcs are created. The model must be finished by specifying precedence (preconditions), work times, post times, engineer requirements and possible alternations on the standard building blocks:

- For every precedence relation put a precondition on the start transition of the corresponding task.
- Specify work times for every task
- Optionally specify post times for precedence relations
- Optionally specify which engineers may perform a task
- Optionally specify which tasks can have a rejection and in which version
- Optionally make other adjustments

5. Run a simulation

Use the build-in simulation tools to run the model. A log file 'export.xls' is automatically created which can be read and used to create output graphs and extra output parameters.

The user interface provides the user with an abstraction from the interrupt and trigger complexity. The main advantage is that he is not concerned with all the complex details of the implementation of interrupts and triggers. The user just connects a deliverable to deliverables that are affected by an error or a rejection in a review. He defines precedence by specifying which phases of input deliverables are needed for a task to start.

6.3 Important notes

6.3.1 Interrupts and triggers

It is important to remember that on the process level the arrows only indicate the flow of the interrupts. In most cases this flow of interrupts will coincide with the flow of triggers in the model, such that the top model can be regarded as the flow of triggers. However the real triggers still have to be defined as preconditions of the start transition of the tasks and there might be models in which the interrupts tokens don't exactly follow the triggers. An experienced user will be able to reduce the net of interrupt tokens to a simpler net, knowing that some interrupt tokens don't influence the outcome of the simulation.

6.3.2 Timed precondition

Because triggers are modeled as preconditions on start transitions, it is possible that the entire system comes to a state where no task is being performed and the whole system is waiting for a deliverable to arrive somewhere, i.e. the system is waiting for a post time to run out. For example the model time is 25 and a start transition becomes enabled at time 50 and there is no token in the system with a timestamp higher than 25. Normally CPN Tools would stop the simulation because there is no token with a timestamp higher than the current time.

The workaround is that there is an extra post time token on the top level that will make sure that the CPN Tools net doesn't stop simulation and continues at least till the moment that no deliverables are underway. This brings a risk.

If for whatever reason it is decided that CPN Tools changes the way it handles timed preconditions, it could be possible that a performance problem occurs. If, for example, CPN Tools in some future version increments time by small amounts until the timed precondition becomes true instead of just jumping to the nearest timestamp of a token, then this will take a lot of unnecessary resources.

At this moment no such change of handling timed preconditions is expected.

6.3.3 Cloning in CPN Tools

If a page, containing subnets, is cloned within a single CPN Tools file, then the subnets in the cloned page will refer to the same subnets as the original page. In CPN this is called a new instance of the same page. Changes to the original will also affect the cloned page and vice versa. In this model this is generally not wanted. Namely, every task will have its own parameters and is not identical to another task. The workaround is that when cloning building blocks to a new model, the building blocks must always be cloned from a separate file to a model in a new file. Also every clone operation needs to be carried out separately. For example if two 3-task lifecycles are needed in the model, then the 3-task lifecycle building block will have to be copied twice and pasted twice. Copy it once and paste it twice will not work, because then there will be two instances of the same page again.

6.3.4 Standard input arc

The current standard building blocks contain one input arc for each start transition, where normally only an arc is needed per input condition. However, because there is always a token in the place 'Deliverables' and no condition is specified initially on this arc, this arc does not influence the simulation. This arc is only placed for easily creating input conditions. Only the textual condition itself needs to be placed while the arc is already there. For every extra input condition a new arc is needed

6.4 Bugs in CPN Tools

CPN Tools is under active development, as follows from the list of known bugs (more than 200) of which more than 20 bugs with status 'High priority'. Some of these bugs are merely new features and some are not applicable to the enhanced model. There are however some bugs that, at this moment, form an obstacle when using the model. These bugs will have to be solved before the model can be used in an acceptable way. Note that the bugs mentioned here are all known and verified bugs in the latest external version 1.2.2. Some of the bugs are already fixed in the latest internal version (currently 1.3.9).

The current status of these bugs can be found at:
http://www.daimi.au.dk/~cpntools/bin/bugs/todo.php?bugslongview=1&bugsbuild=1.2.2
or via http://wiki.daimi.au.dk/cpntools.

1 CPN Tools bug ID 1664:
Problem: Model checking large nets takes too much time/resources
Solution: No solution yet, problem is being worked on

2 CPN Tools bug ID 1542:
Problem: Fusion sets sometimes do not update properly on different pages
Solution: Close and reload model

3 CPN Tools bug ID 1822 (Fixed in 1.3.9):

Problem:	Cloning pages and saving them gives two instances of tasks
Solution:	Use xml editor to remove double instances

4 CPN Tools bug ID: 1506 (Fixed in 1.3.6):

Problem:	arc tags wrongly positioned after cloning
Solution:	Re-layout the cloned drawing

5 CPN Tools bug ID: various, see web page

Problem:	Unexpected internal errors
Solution:	Often: Reload model

7 Conclusions

Modeling product development processes is possible in CPN Tools. The complexity, uncertainty and iterations of such processes are dealt with inside the building blocks that were created during this project. The drawback is that CPN Tools is unable to handle medium to large-sized nets.

The main problem to be solved when modeling product development processes is the handling of interrupts. Because interrupts can occur at any time and at any place and in principle can influence every task in the entire process, they are hard to model using a standard workflow approach. This problem is solved by building interrupt constructs on every level of the hierarchical building blocks model. These constructs pass the interrupts instantaneously through the entire model.

A derived problem is the problem of triggers that can be withdrawn anytime, causing a complex administration of trigger tokens and withdrawn trigger tokens when a standard workflow model is used. This problem was solved by using preconditions on start transitions instead of trigger tokens. The preconditions are based on a global information store where the statuses of all deliverables are administrated.

The user interface provides the user with an abstraction from the interrupt and trigger complexity. The main advantage is that he is not concerned with all the complex details of the implementation of interrupts and triggers. The user just connects a deliverable to deliverables that are affected by an error or a rejection in a review. He defines precedence by specifying which phases of input deliverables are needed for a task to start.

The main advantage of the process-oriented workflow approach as used in the enhanced model compared to the data-oriented approach used in the Exspect model are easy configuration, flexibility and visualization of the process steps. The abstract approach requires one very complex model of a task. Special cases need an exception in an already complex model. The process-oriented workflow approach avoids this complexity by spreading out the complexity of different task over different parts of the model.

The main disadvantage of CPN Tools is the large number of bugs that still exist in the latest version (currently 1.3.9). In particular the performance problem CPN Tools has dealing with large nets. Because of this problem it was impossible to test the enhanced model on industrial sized nets. Because large nets are inherent to the process-oriented workflow approach, it remains uncertain whether this approach is feasible even without the CPN Tools performance problems

8 Future research

The biggest burden during this research was the inability of CPN Tools to check and simulate large nets. Because of this burden, the model could not be validated for large nets. Only when a solution is found to this problem, it is possible to check whether it is feasible to do stochastic analysis on complex product development processes. Finding a solution to this problem is in the hands of the developers of CPN Tools at the University of Aarhus, Denmark.

Initially the goal of this project was to develop a simulation model to be used during the case study of R. Benders. When the case company was no longer involved in this research halfway the project, it was decided to build a CPN model for the general case. Because of the short time that was left, it was impossible to find other companies to validate the model. Although the CPN Tools implementation was not validated, the underlying business model was. However, a real validation process needs more than one validation. Thus the model should be applied to more companies before more solid conclusions can be drawn and more insight can be gained on product development processes.

Availability of engineers, priority and reuse are concepts that are not yet incorporated in the model. During the case no need for them arose and no proper definition for using them in a simulation model was found. The need for these concepts has to prove itself and a solid understanding of these concepts has to be developed, before introducing them into an already complex model. For the cases where these constructs would come handy, workarounds are sometimes possible, such that these situations can be modeled using this tool.

At this moment engineers from the same discipline are assigned random to tasks that are waiting to be performed. If engineers have to be assigned using an assignment rule, for example shortest processing time first, then some sort of list has to be collected of tasks that are waiting to be performed. This makes the model complex, because in principle the separate tasks don't know of each others existence. In the Exspect approach this would be much easier. In that approach almost all information is available to every part of the process. Future cases have to indicate whether more elaborated options for engineer assignment are needed. In those cases the Exspect approach could be preferable.

In the current situation it can take quite some work to adjust work times, post times and engineer requirements, because they are all specified on the lowest level. Because the model consists of standard building blocks, it must be possible to specify the model largely in a spreadsheet or database and then have the CPN model generated automatically. This way most input parameters can be gathered in one spreadsheet or in a few database tables. Especially for larger models this can significantly reduce the time needed to build an initial model.

Appendix A: List of references

[Aalst]	'Workflow Patterns: On the Expressive Power of (Petri-net-based) Workflow Languages', W.M.P. van der Aalst and A.H.M. ter Hofstede, paper
[Benders]	'Performance of a multi-company product development process: simulation of the effects of product data exchange', December 2004 Raymond Benders, Graduation report TU/e
[CIMdata]	'PDM to PLM: Evolving to the future', February 2004 By: Ken Amann, Director of Research, CIMdata, Inc, CIMData 2004 http://www.coe.org/newsnet/feb04/industry.cfm#1
[Clark]	'Managing new product and process development: Text and Cases', 1993 Kim B. Clark and Steven C. Wheelwright; Harvard business school
[Clausing]	'Total Quality Development: A Step-By-Step Guide to World-Class Concurrent Engineering', D. Clausing, ASME Press, New York, 1994
[Coolen]	'Development of a simulation model for the RapidPDM project', 2000 Tom Coolen, Graduation report TU/e
[De Graaf]	'Assessing Product Development: Visualizing Process and Technology performance with RACE', September 1996 Robert de Graaf, Doctoral thesis TU/e
[Elzen]	'Simulation of product development processes', 1999 Rob van den Elzen, Graduation report TU/e
[Jans]	'Simulation of product development processes in Exspect', 1999 Erik J.A. Jans, Graduation report TU/e
[Limam]	'Simulation of Operational Processes'; July 2001. Limam, S, course material for SOP 1B330 TU/e
[Smith]	'Simulation: The Engine Behind The Virtual World', 1999 Smith, R.D., Volume 1 in the Simulation 2000 series
[STT]	'Vernieuwing in productontwikkeling', 1999 Arie Korbijn, Stichting Toekomstbeeld der Techniek
[Ulrich, Eppinger]	Product Design and Development, 2000 Karl T. Ulrich and Steven D. Eppinger

Appendix B: RapidPDM access data model

The following information is specified in Access and used in ExSpect:

Deliverables

ID	Ref_DocType	Ref_Product	Ref_ProductName	Ref_Domain	Ref_Discipline	DocComplexity

The deliverable table lists all the deliverables that must be created during the development process. This list can be generated or edited manually.

- ID: unique deliverable identifier
- Ref_DocType: type of the deliverable: determines the life cycle phases
- Ref_Product: the id of the product that is specified in this deliverable
- Ref_ProductName: the product name is copied for convenience
- Ref_Discipline: the discipline of the product
- Ref_Domain: the domain in which the deliverable is to be created
- DocComplexity: fraction that this deliverable takes of the total complexity of the product. The workload for a deliverable is calculated from the complexity.
- ReuseFactor: the reuse factor of the product: if the product is reused for a certain percentage, then the deliverable enters the first phase with this quality.

Tasks

Ref_DocType	Ref_Phase	Ref_Discipline	Min_skill	Max_s	Weight	Cycles	Quality	Same_Engineer

In each life cycle phase one or more tasks must be executed on a deliverable. This table controls the amount of work spent in each phase and the quality effects

- Ref_DocType: the deliverable type
- Ref_Phase: the life cycle phase for that deliverable type
- Ref_Discipline: the discipline of the required engineer (has no effect in the model: the discipline is decided by the discipline of the product).
- Min_skill: the engineer to be assigned to this task must be of the specified discipline and have at list the minimum skill level
- Max_skill: the engineer must be of the specified discipline and have at most the maximum skill level
- Weight: if the complexity of the deliverable is C then the task will take Weight*C hours of work
- Cycles: the number of cycles that is made in this phase: cycles = n implies that the task will be rejected n times. The first and the last phase will never be rejected: cycles have no effect
- Quality: defines the quality effect of the phase: the first phase will initially deliver the deliverable with the indicated quality (60%). The second phase will detect 80% of the remaining errors (0.8*0.4 = 32% of the total nr of specifications) and cause a change order. A new version will be created that starts in phase 1, where the quality will be improved (to 60%+32%= 92%) with this fraction and which will take a proportional amount of hours effort (32% of the original workload of the deliverable). If two cycles were specified then the 2^{nd} cycle would improve the quality with .08*.8=6.4% to 98.4%.
- Same_engineer: indicates whether this task must be assigned to the same engineer as the previous task or may not be assigned to the same engineer as the previous task.

Precedence

Ref_Deliverable_Desc	Ref_Deliverable_Pred	MinPhaseDiff	Propagating

Precedence specifies what deliverables are required as input for other deliverables

- Ref_Deliverable_Desc: id of the descendant deliverable: the descendant requires the predecessor as input
- Ref_Deliverable_Pred: id of the predecessor; the required input deliverable
- MinPhaseDiff: to override the minimal phase difference as set in the parameter table.
- Propagating: indicates whether this precedence link will propagate changes: if Yes then a change with a certain fraction of complexity will cause a reduction of quality of the descendant and a change order for the descendant.

The precedence table can be generated optionally by the output form. Precedence follows the product structure: if top down then the parent will precede the child, else otherwise. Precedence also follows the domains: the earlier domain precedes the latter domain. Precedence can be modified by hand: links can be edited or deleted. This manual modification will be erased when the precedence is generated again.

Engineers

ID	Ref_Discipline	Skill	Name

The available capacity of engineers is defined in this table. An arbitrary number of engineers can be defined for each discipline.

- ID: unique identification of engineer
- Ref_Discipline: discipline of engineer
- Skill: skill level of engineer: decides on the tasks he may execute.
- Name: name of the engineer: the actual names of engineers in client company may be chosen here for easy reference during analysis.

Parameter Effect

Variable	Value	Description
Reuse	0	Level of Reuse (0,1,2,3)
Reuse_percentage	60	Initial quality of deliverables that are reused (-100)
min_phasediff	1	Level of Concurrency (CE=1/Non-CE=5)
Search_time	0	search time per per input deliverable
Post_time	5	Time to go from one engineer to another

This table is used to set the parameters for process and PDM functionality. Predefined table. Only the column value may be changed:

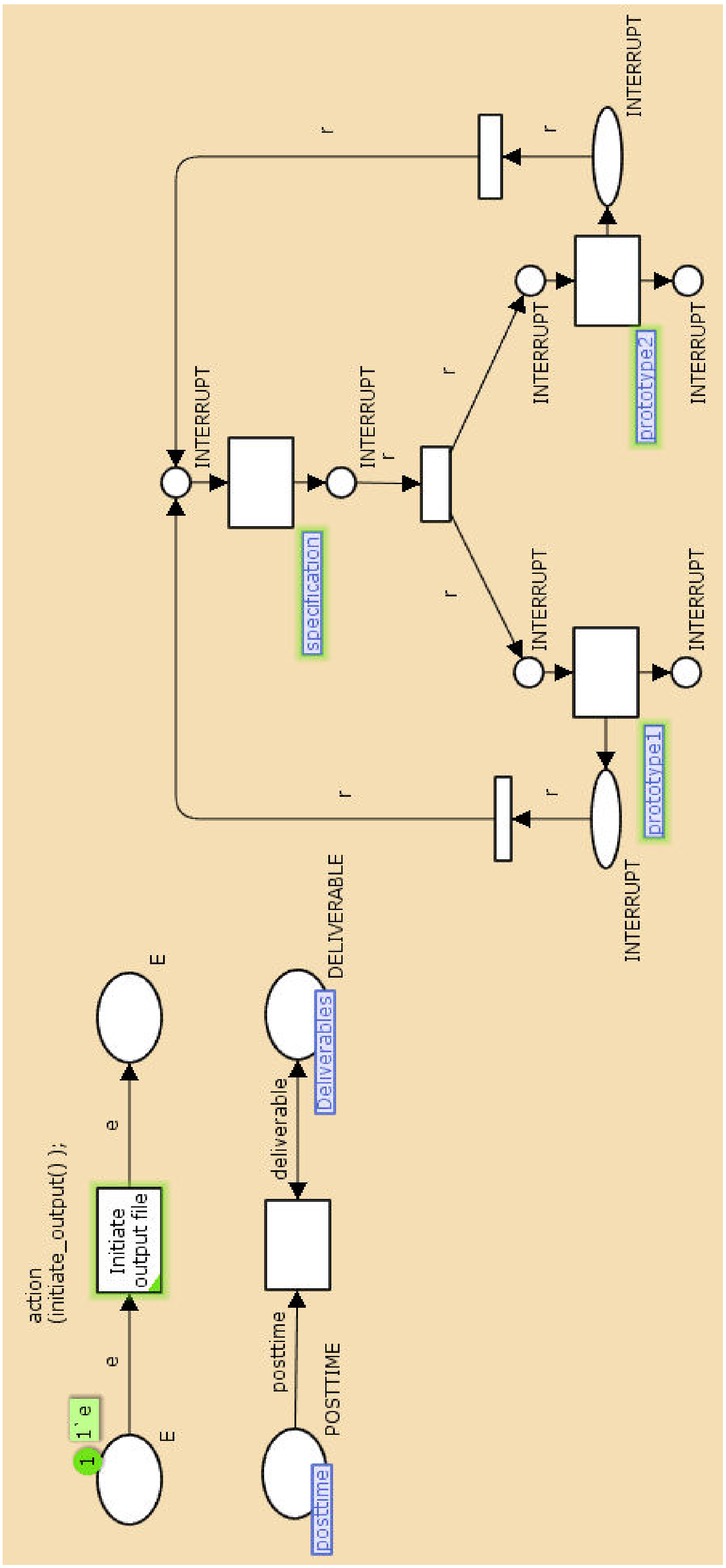
- Reuse: sets the level of reuse: a higher level will cause more products to be reused depending on the values set in the product table
- Reuse_percentage: sets the percentage of reuse in the reused products: this percentage of the complexity of each deliverable will be taken as initial quality for deliverables of reused products
- Min_phasediff: this parameter sets the level of concurrent engineering. Tasks in a life cycle phase may start when all preceding deliverables have reached a phase with a sequence number that is at least the min-phase difference higher: value 1 means the e.g. phase 2 may start if all preceding deliverables have at least phase 2+1=3. Sequential engineering is obtained when min-phasediff is equal to the highest phase

number. Full concurrent engineering is obtained with value 1: review may start when input deliverables have been reviewed. The model will also accept value 0: editing a deliverable may start as soon as the editing of the input deliverables has started. This is more a theoretical than a practical option.

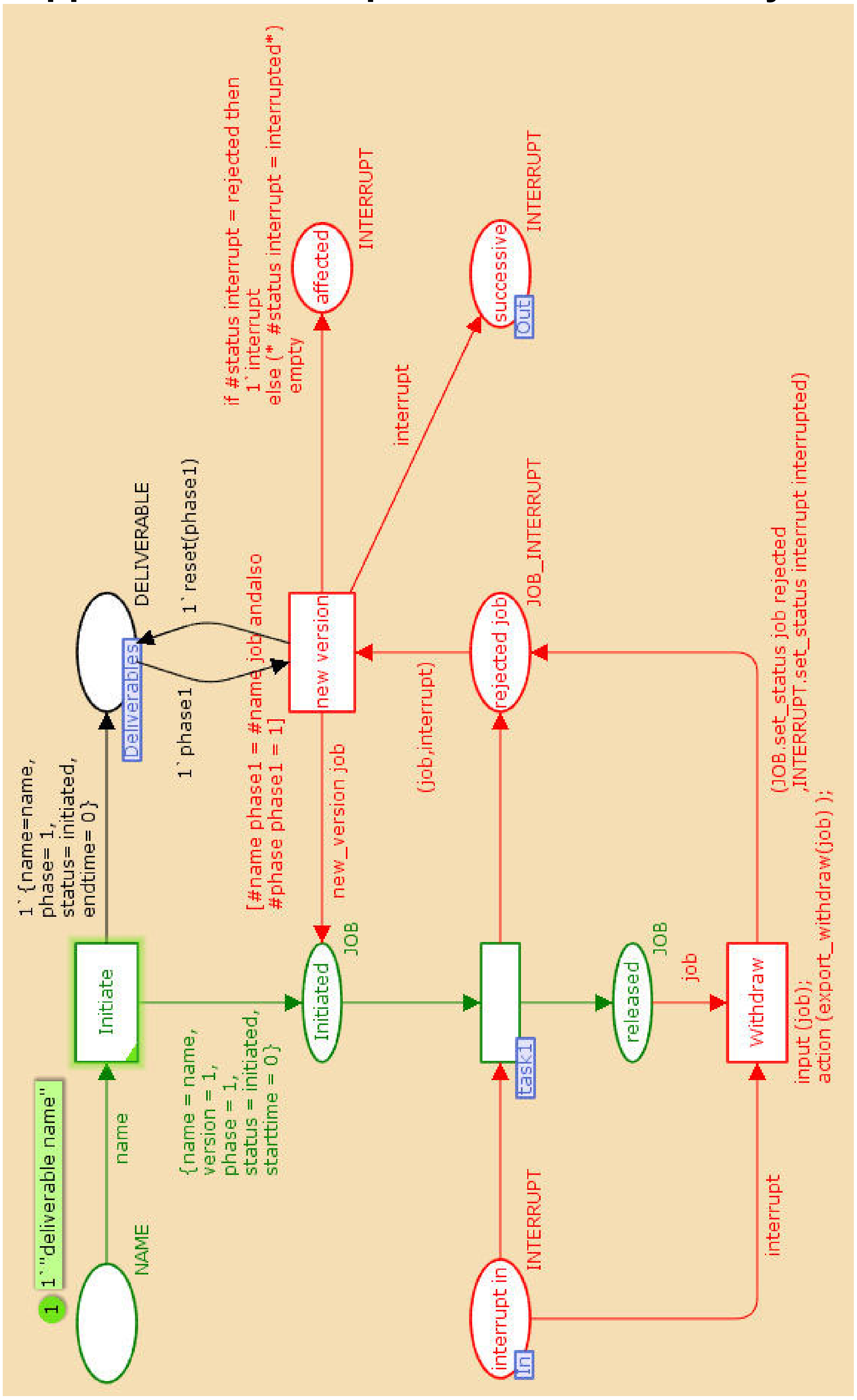
- Search-time: this time will be added to the working time of each preceding task for searching the preceding deliverable.
- Post_time: this time will be taken as slack for transport by the post system between the moment a preceding deliverable is promoted to the required life cycle phase and the moment it becomes available for the descending deliverable. A typical time is 16 hours (two days).

Note: in the original model the deliverables were optionally generated from a set of products and deliverable generation rules. Because this part of the model will be left out in the newer version, it is not be extensively described here.

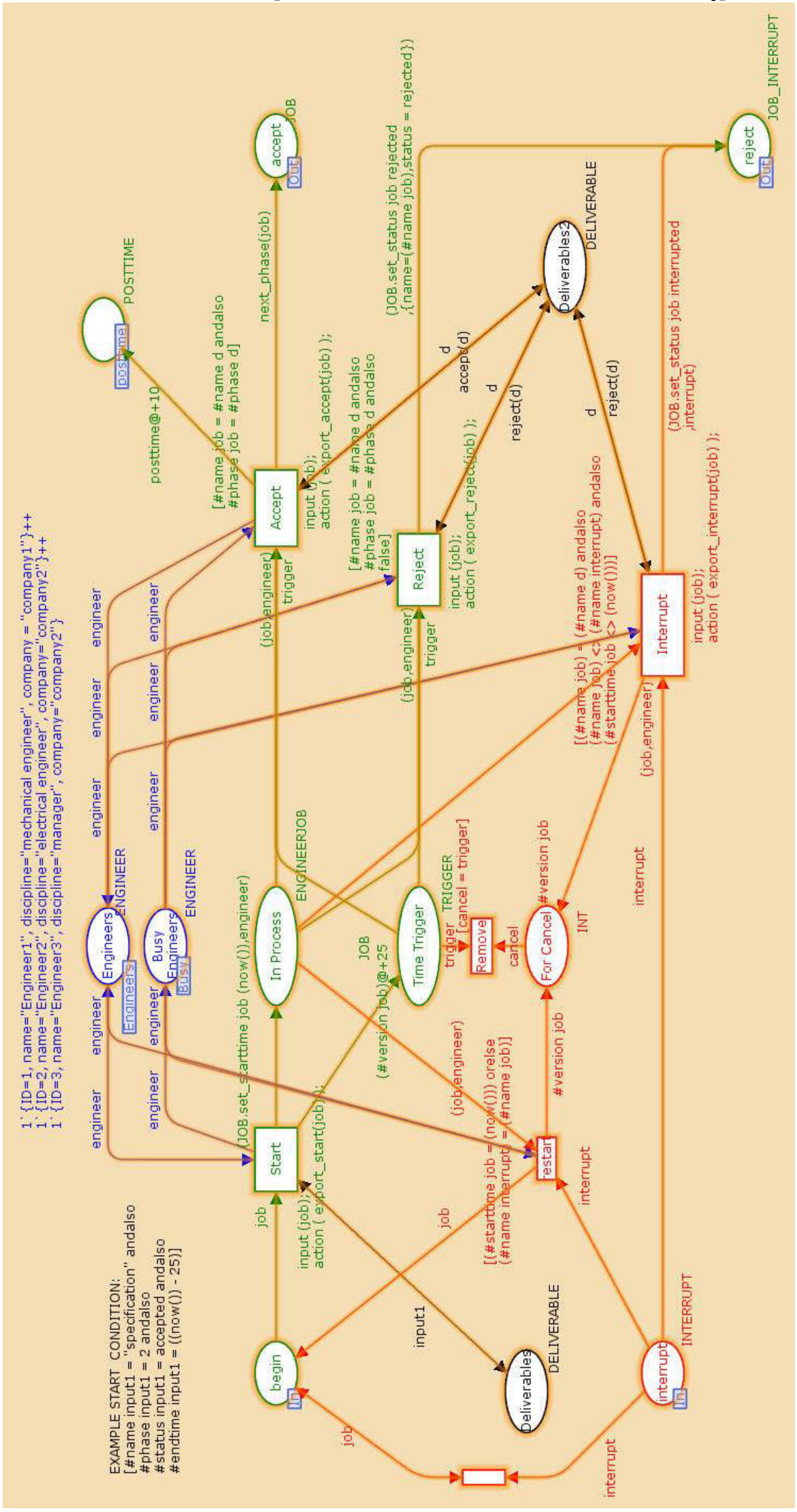
Appendix C1: Implementation of a process



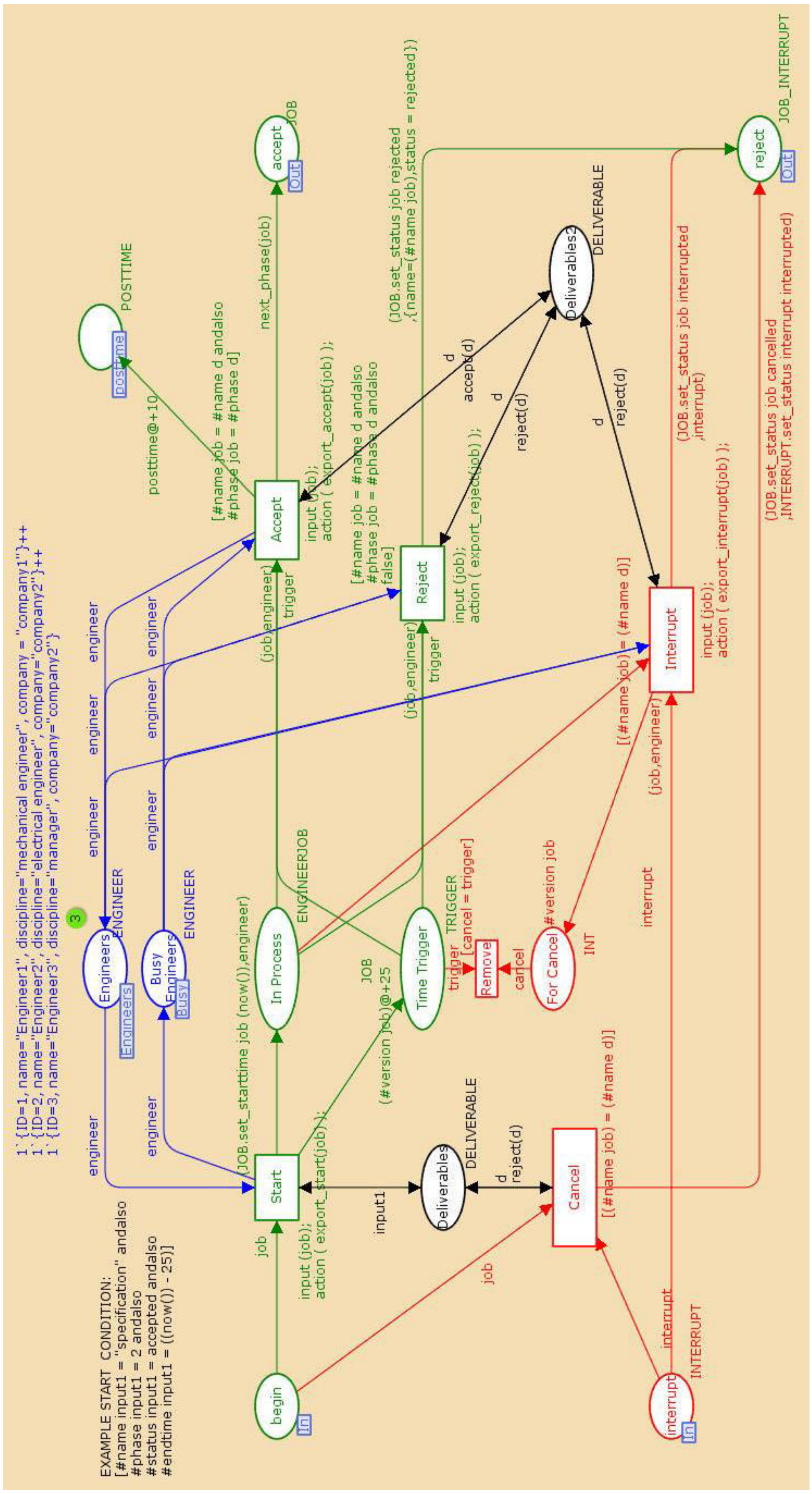
Appendix C2: Implementation of a lifecycle


1
1`"deliverable name"
NAME
name
Initiate
1`{name=name, phase= 1, status= initiated, endtime= 0}
DELIVERABLE
Deliverables
1`phase1
1`reset(phase1)
[#name phase1 = #name job andalso #phase phase1 = 1]
{name = name, version = 1, phase = 1, status = initiated, starttime = 0}
Initiated
JOB
new version
new_version job
if #status interrupt = rejected then 1`interrupt else (* #status interrupt = interrupted*) empty
affected
INTERRUPT
interrupt
successive
Out
INTERRUPT
rejected job
JOB_INTERRUPT
(job,interrupt)
task1
released
JOB
job
Withdraw
input (job); action (export_withdraw(job));
(JOB.set_status job rejected ,INTERRUPT.set_status interrupt interrupted)
interrupt in
In
INTERRUPT
interrupt

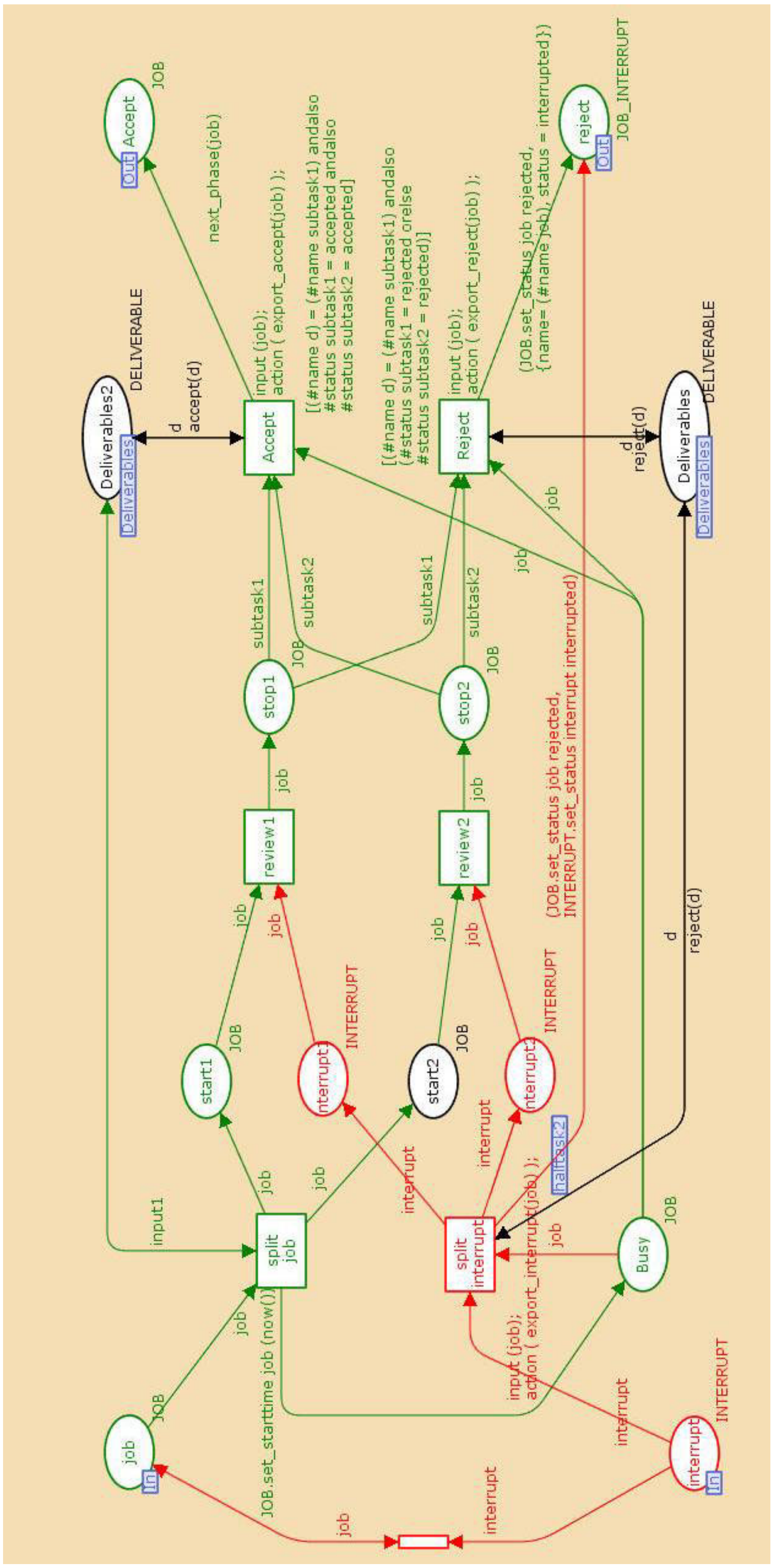
Appendix C3a: Implementation of a task (phase=1)


1`{ID=1, name="Engineer1", discipline="mechanical engineer", company = "company1"}++
1`{ID=2, name="Engineer2", discipline="electrical engineer", company="company2"}++
1`{ID=3, name="Engineer3", discipline="manager", company="company2"}
EXAMPLE START CONDITION:
[#name input1 = "specification" andalso
#phase input1 = 2 andalso
#status input1 = accepted andalso
#endtime input1 = ((now()) - 25)]
POSTTIME
posttime
posttime@+10
accept
Out
JOB
next_phase(job)
[#name job = #name d andalso
#phase job = #phase d]
Accept
input (job);
action (export_accept(job));
(JOB.set_status job rejected
,{name=(#name job),status = rejected})
Deliverables2
DELIVERABLE
d
accept(d)
reject(d)
[#name job = #name d andalso
#phase job = #phase d andalso
false]
Reject
input (job);
action (export_reject(job));
(JOB.set_status job interrupted
,interrupt)
reject
JOB_INTERRUPT
[(#name job) = (#name d) andalso
(#name job) <> (#name interrupt) andalso
(#starttime job <> (now()))]
Interrupt
input (job);
action (export_interrupt(job));
(job,engineer)
trigger
engineer
Engineers
ENGINEER
Busy Engineers
Busy
In Process
ENGINEERJOB
(JOB.set_starttime job (now()),engineer)
Time Trigger
TRIGGER
[cancel = trigger]
Remove
cancel
For Cancel
INT
#version job
JOB
(#version job)@+25
interrupt
Start
input (job);
action (export_start(job));
job
restart
[(#starttime job = (now())) orelse
(#name interrupt = (#name job))]
begin
In
input1
Deliverables
interrupt
INTERRUPT

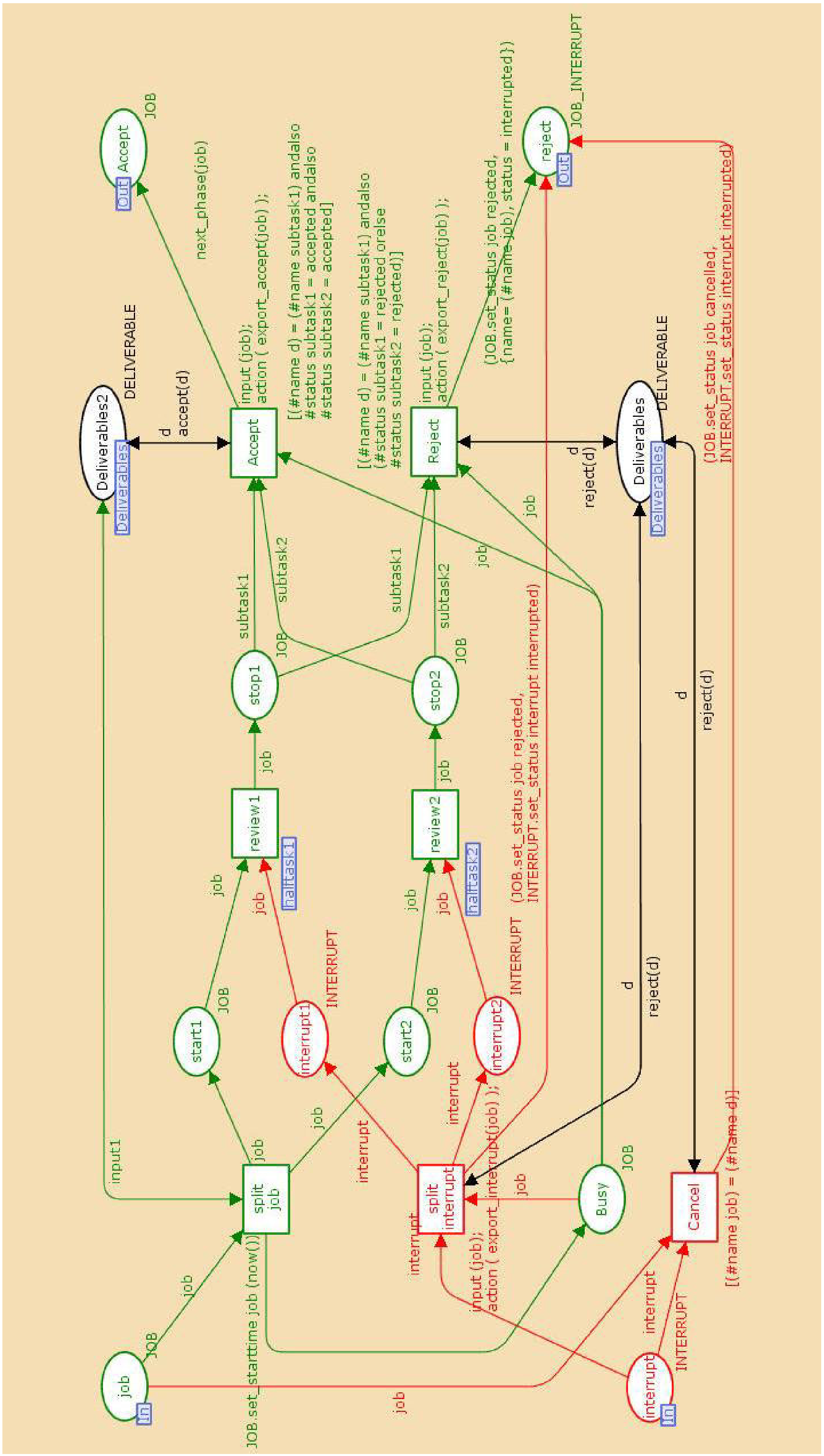
Appendix C3b: Implementation of a task (phase>1)


EXAMPLE START CONDITION:
[#name input1 = "specification" andalso
#phase input1 = 2 andalso
#status input1 = accepted andalso
#endtime input1 = ((now()) - 25)]
1`{ID=1, name="Engineer1", discipline="mechanical engineer", company = "company1"}++
1`{ID=2, name="Engineer2", discipline="electrical engineer", company="company2"}++
1`{ID=3, name="Engineer3", discipline="manager", company="company2"}
begin
In
job
Start
input (job);
action (export_start(job));
JOB.set_starttime job (now()),engineer)
In Process
ENGINEERJOB
Engineers
ENGINEER
Busy Engineers
Busy
engineer
Accept
[#name job = #name d andalso
#phase job = #phase d]
input (job);
action (export_accept(job));
next_phase(job)
accept
Out
JOB
posttime@+10
posttime
POSTTIME
(job,engineer)
trigger
Reject
[#name job = #name d andalso
#phase job = #phase d andalso
false]
input (job);
action (export_reject(job));
(JOB.set_status job rejected
,{name=(#name job),status = rejected})
reject
JOB_INTERRUPT
Deliverables2
DELIVERABLE
d
accept(d)
reject(d)
Interrupt
[(#name job) = (#name d)]
input (job);
action (export_interrupt(job));
(JOB.set_status job interrupted
,interrupt)
(JOB.set_status job cancelled
,INTERRUPT.set_status interrupt interrupted)
(job,engineer)
(#version job)@+25
JOB
Time Trigger
TRIGGER
[cancel = trigger]
Remove
cancel
#version job
For Cancel
INT
input1
Deliverables
DELIVERABLE
Cancel
[(#name job) = (#name d)]
interrupt
INTERRUPT

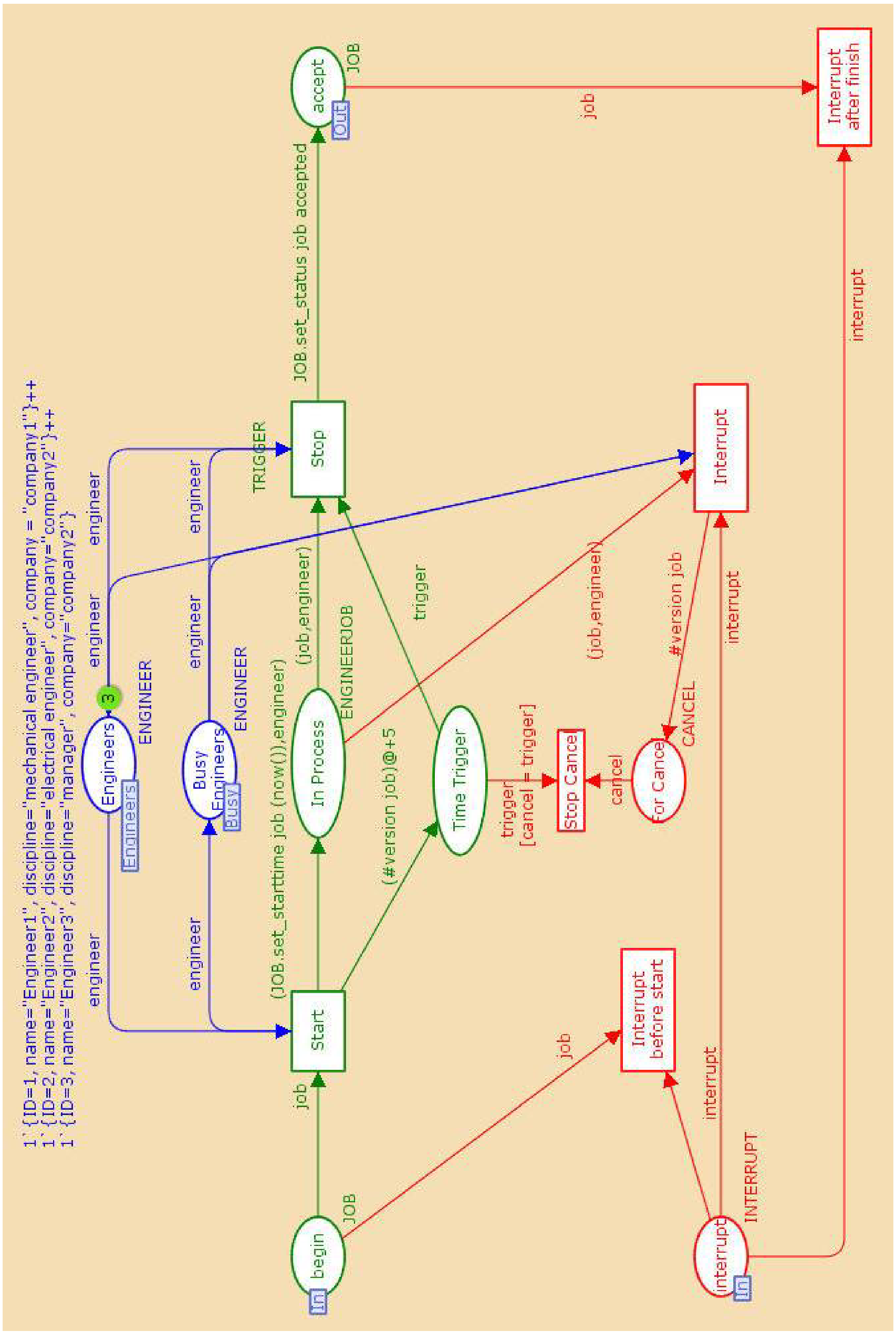
Appendix C4a: Parallel tasks (phase=1)


Accept
Out
JOB
next_phase(job)
input (job);
action (export_accept(job));
[(#name d) = (#name subtask1) andalso
#status subtask1 = accepted andalso
#status subtask2 = accepted]
Deliverables2
Deliverables
DELIVERABLE
d
accept(d)
Accept
Reject
[(#name d) = (#name subtask1) andalso
(#status subtask1 = rejected orelse
#status subtask2 = rejected)]
input (job);
action (export_reject(job));
(JOB.set_status job rejected,
{name= (#name job), status = interrupted})
reject
Out
JOB_INTERRUPT
Deliverables
Deliverables
DELIVERABLE
d
reject(d)
subtask1
subtask2
JOB
stop1
stop2
job
review1
review2
start1
start2
interrupt1
interrupt2
INTERRUPT
(JOB.set_status job rejected,
INTERRUPT.set_status interrupt interrupted)
split job
split interrupt
haltTask2
input (job);
action (export_interrupt(job));
Busy
input1
JOB.set_starttime job (now())
job
In
interrupt
In
INTERRUPT

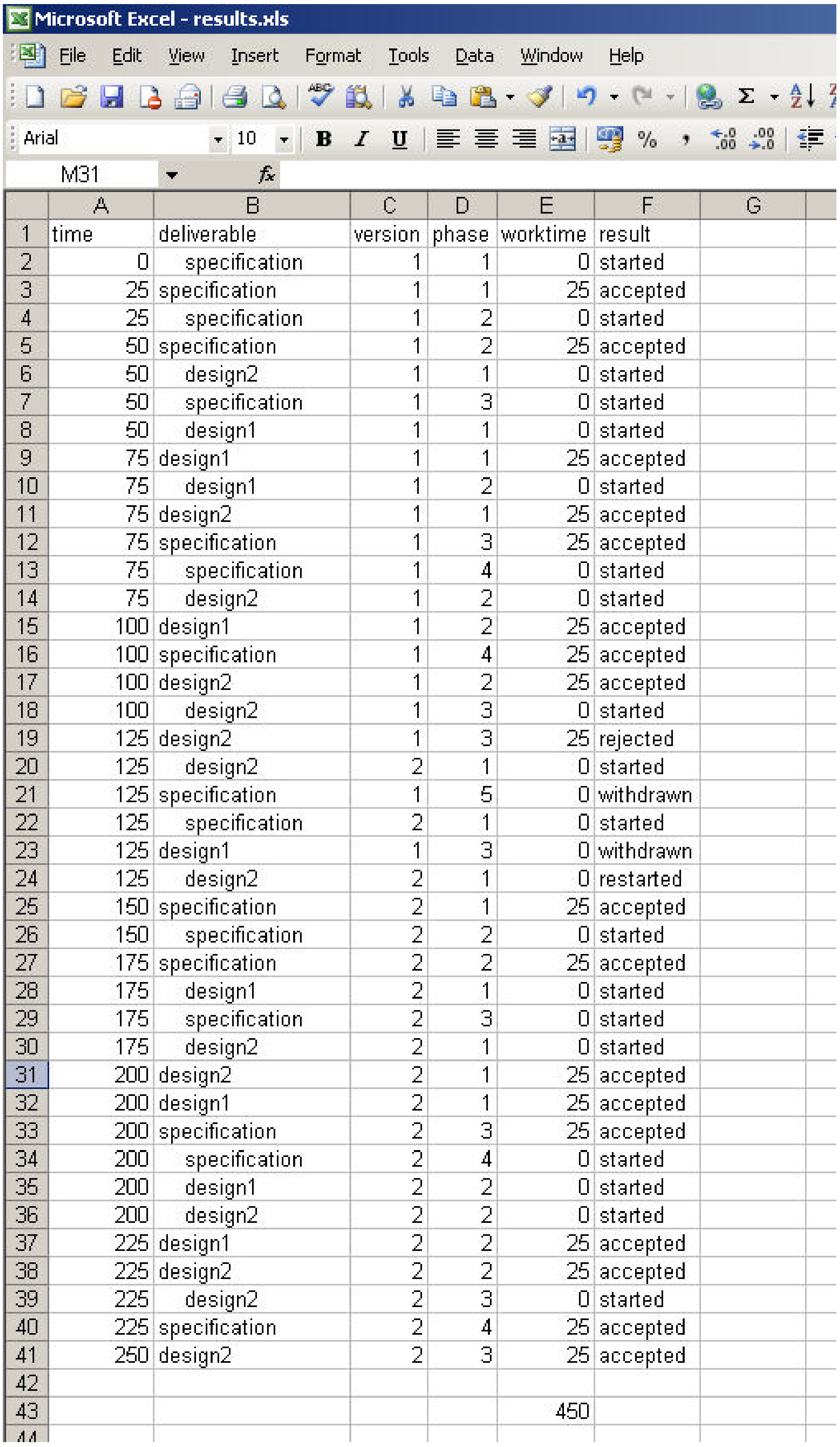
Appendix C4b: Parallel tasks (phase>1)


job
In
JOB
input1
split job
JOB.set_starttime job (now())
start1
start2
interrupt1
interrupt2
INTERRUPT
review1
review2
halttask1
halttask2
stop1
stop2
subtask1
subtask2
Accept
Reject
Deliverables2
Deliverables
DELIVERABLE
accept(d)
reject(d)
next_phase(job)
input (job); action (export_accept(job));
[(#name d) = (#name subtask1) andalso #status subtask1 = accepted andalso #status subtask2 = accepted]
[(#name d) = (#name subtask1) andalso (#status subtask1 = rejected orelse #status subtask2 = rejected)]
input (job); action (export_reject(job));
(JOB.set_status job rejected, {name= (#name job), status = interrupted})
reject
Out
JOB_INTERRUPT
(JOB.set_status job rejected, INTERRUPT.set_status interrupt interrupted)
split interrupt
input (job); action (export_interrupt(job));
Busy
interrupt
Cancel
(JOB.set_status job cancelled, INTERRUPT.set_status interrupt interrupted)
[(#name job) = (#name d)]

Appendix C4c: Implementation of a subtask



Appendix D: Example Export file


Microsoft Excel - results.xls
File Edit View Insert Format Tools Data Window Help
Arial 10
M31

	A	B	C	D	E	F	G
1	time	deliverable	version	phase	worktime	result	
2	0	specification	1	1	0	started	
3	25	specification	1	1	25	accepted	
4	25	specification	1	2	0	started	
5	50	specification	1	2	25	accepted	
6	50	design2	1	1	0	started	
7	50	specification	1	3	0	started	
8	50	design1	1	1	0	started	
9	75	design1	1	1	25	accepted	
10	75	design1	1	2	0	started	
11	75	design2	1	1	25	accepted	
12	75	specification	1	3	25	accepted	
13	75	specification	1	4	0	started	
14	75	design2	1	2	0	started	
15	100	design1	1	2	25	accepted	
16	100	specification	1	4	25	accepted	
17	100	design2	1	2	25	accepted	
18	100	design2	1	3	0	started	
19	125	design2	1	3	25	rejected	
20	125	design2	2	1	0	started	
21	125	specification	1	5	0	withdrawn	
22	125	specification	2	1	0	started	
23	125	design1	1	3	0	withdrawn	
24	125	design2	2	1	0	restarted	
25	150	specification	2	1	25	accepted	
26	150	specification	2	2	0	started	
27	175	specification	2	2	25	accepted	
28	175	design1	2	1	0	started	
29	175	specification	2	3	0	started	
30	175	design2	2	1	0	started	
31	200	design2	2	1	25	accepted	
32	200	design1	2	1	25	accepted	
33	200	specification	2	3	25	accepted	
34	200	specification	2	4	0	started	
35	200	design1	2	2	0	started	
36	200	design2	2	2	0	started	
37	225	design1	2	2	25	accepted	
38	225	design2	2	2	25	accepted	
39	225	design2	2	3	0	started	
40	225	specification	2	4	25	accepted	
41	250	design2	2	3	25	accepted	
42							
43					450		
44							