Modeling and Changing Business Process Models with Concurrent Task Trees

DI Thomas G. Gress, BSc

Innsbruck, October 2012
## Contents

1. **Introduction** 5  
   1.1. Background ........................................ 5  
       1.1.1. End-User-Development ........................... 6  
       1.1.2. Concurrent Task Trees .......................... 6  
   1.2. Problem Statement .................................... 7  
   1.3. Research Objective ................................... 7  
   1.4. Research Method ..................................... 8  
   1.5. Related Work ........................................ 10  
   1.6. Structure ........................................... 11  

2. **Backgrounds** 13  
   2.1. Concurrent Task Trees ................................. 13  
   2.2. Cheetah Experimental Platform .......................... 18  
       2.2.1. Provided Functionalities by CEP .................... 18  

3. **Requirements** 21  
   3.1. Modeling Requirements .................................. 22  
   3.2. User Interface Requirements ............................ 24  
   3.3. Backend Requirements .................................. 24  

4. **Implementation** 27  
   4.1. Architectural Overview and Integration into Cheetah Platform .... 27  
       4.1.1. Technical Integration into Cheetah Experimental Platform Architecture ................................. 28  
       4.1.2. Providing Analysis Capabilities for the CTT Editor ............... 33
Chapter 1.

Introduction

1.1. Background

A trend which can be identified in information systems of enterprises is the shift from data to processes [vdAHW03]. In the nineties process reengineering [Hol95] and other developments pointed management towards process driven approaches. A process is defined by [Dav93] as “a specific ordering of work activities across time and place with a beginning, an end and clearly identified inputs and outputs: a structure for action”. A business process stresses “how” work is done more than “what” the output is. This implies that potential for improving the process can be discovered by knowing how something is done. Workflows are strongly connected with business processes. [Hol95] defines a workflow as “the computerized facilitation of automation of a business process, in whole or in part”.

The need for managing business processes and workflows resulted in the definition of Business Process Management which was described as technology for [vdAHW03] “supporting business processes using methods, techniques, and software to design, enact, control, and analyze operational processes involving humans, organizations, applications, documents and other sources of information”. Conceptual design of business processes is often done by using business process models. The graphical representation of a business process in a formal manner leads to a reduction of ambiguity and potential for analysis of business processes.

In order to conduct sustained Business Process Management the involvement of humans as described in the definition of Business Process Management [Obj11]
has to be considered. In fact also [vdAHW03] states that business process models should be well understandable for the various stakeholders. The logical deduction and a further progress in the development of process models will be that stakeholders should not only be able to understand these process models, but shall also be in the position of having tools that allow them to create and alter process models themselves (cf. [Kol11]). Formal languages for modeling business processes combine the just mentioned issues and assure that alternative interpretations are avoided.

1.1.1. End-User-Development

A goal of today's research in the field of computer science is to achieve easy to use systems [LPKW06]. Although in respect to this aim the full potential is not yet entirely tapped, the further development of this trend goes towards End-User-Development. Easy to develop systems can foster the usage, but also the creation of new systems with strong considerations of end-users. People without programming skills or deep technical knowledge will be supported when delivering input for the development of new systems. Aim of End-User-Development is therefore to provide tools for non-professional software developers in order to take influence on software artifacts (cf. [LPKW06]). A first step in this direction is to involve end-users in the initial design phase of systems. One type of development which supports program creation and modification according to [LPKW06] and [Pat99b] is Model-based development. The user is encouraged to provide a conceptual description of the functionality he expects from the system. The system is then able to create an interactive application. A proposed modeling language for enabling users to participate in End-User-Development are Concurrent Task Trees (CTT) [Pat99a].

1.1.2. Concurrent Task Trees

Concurrent Task Trees (CTT), sometimes also referred to as ConcurTaskTrees, are a widely spread Task Modeling Language [Kol11]. The focus of this type of modeling lays on the usability and ease of use for End-User-Development. People
who are not familiar with Business Process Modeling (BPM) might eventually find it easier to model using CTTs than using common process modeling languages such as BPMN [Obj11]. Using graphical notations like CTT might eventually help to better highlight hierarchical structures [Pat99a]. The fact of being a hierarchical organized notation permits to apply visualization concepts that might help the user to better understand the modeled process [Kol11]. This is mainly important if the modeling is done by end-users who usually do not have that much experience in creating and understanding complex process models. Another advantage of End-User-Development is that, for example, software specification can be done by users and stakeholders of the software without having deep technical knowledge [LPW06]. Concurrent Task Trees could bridge the gap between domain experts and engineers of software. Surely this is also a matter of abstraction of the model [Pat99a], still CTTs could be a common language for different stakeholder groups [Kol11]. Using the above mentioned visualization concepts the understandability can be potentially fostered for some groups of users. Applying different layers of abstraction it is possible to create different views at different levels of granularity describing the same process or application.

1.2. Problem Statement

Concurrent Task Trees are mentioned as business process modeling language fostering End-User-Development [Pat99a]. Still there is no empirical data available testing this and it has to be determined if CTTs will provide a good basis for End-User-Development. In addition, it has to be investigated in how far CTT perform better compared to other, widely used business process modeling languages like BPMN [Obj11]. In order to assess whether BPMN or CTT is better suited for End-User-Development the two modeling languages shall be compared.

1.3. Research Objective

The aim of this work is to allow the assessment of the suitability of CTT in terms of End-User-Development and to enable a comparison between Concurrent Task Trees (CTT, cf. Section 1.1.2) and BPMN (cf. [Obj11]), to find out whether CTTs
Chapter 1. Introduction

are better understandable for (inexperienced) modelers than BPMN models. This can be done by conducting controlled experiments. In order to compare the two languages an experiment should be designed using the Cheetah Experimental Platform (CEP, see Section 2.2). CEP was especially designed for conducting such experiments and is already supporting BPMN through the BPMN Modeler. To provide support for CTTs in the CEP the CTT Editor plug-in will be developed enabling modeling of CTTs. The CTT Editor should be designed in order to follow most of the described features in [Kol11]. By providing support for the CTT within CEP direct comparison with BPMN should be possible. This should allow to determine differences in the understandability and ease of use between both CTT and BPMN.

To ensure that the CTT Editor will be easy to use and a valid basis for experiments it should be tested with a running example. Additionally an evaluation applying the Technology Acceptance Model (TAM, cf. [Dav86]) should be conducted. The comparison of CTT with BPMN itself is out of scope of this thesis.

1.4. Research Method

As an appropriate research method for this work the framework introduced by Hevner et. al. [HMPR04] was chosen since it focuses on technology-based design. Hevner proposes a conceptual framework for understanding, executing and evaluating Information System (IS) research. Figure 1.1 shows how the framework is designed.

Goal is to develop artifacts like the CTT Editor plug-in which will satisfy the utility specifications provided from the CEP community. Building on the existing BPMN Modeler of CEP this project is an adaptational task which still needs numerous new features in order to support CTT modeling. The development phase will be run through several times in alteration with the evaluation and justification phase in order to assure a correct working of the IS and its compliance with the utility specifications.
1.4. Research Method

Figure 1.1.: IS Research Framework from [HMPR04]

The environment part of the IS Research Framework by [HMPR04] defines the business needs which in our case are given by the research objective. A second source for environmental specifications are the authors of [Kol11] since they will conduct the experiments using the CTT Editor. This is also the source defining the utility requirements for the CTT part of Cheetah Experimental Platform.

The knowledge base provides the applicable knowledge for the development and evaluation phase. Software development techniques like the V-model [HT00] are contents of the knowledge base which foster the development of the IS as also the unit testing framework (cf. Section 5.1). JUnit [Bec04] is. Also the knowledge about process modeling (e.g., [Pat99a], [PMG01], [LPKW06]) and process modeling languages (e.g., [Obj11], [PMM97]) is a central part of the knowledge base.
Chapter 1. Introduction

1.5. Related Work

Concurrent Task Trees are a modeling language which was firstly introduced by [PMM97]. Since then the support for the language evolved, some new elements were added (cf. [Pat99a]) and a software tool for modelling CTTs was introduced (cf. [PMG01]).

Since the introduction of CTTs numerous authors applied CTT. The main application field of CTTs is the design of user interaction with systems, i.e., user interfaces (e.g., [MBP03], [MRVA08], [JGS07]). CTTs are also mentioned in the area of process modeling (e.g., [PST98], [Kol11], [Agb11]). This work will complement existing research by providing the base for empirical evaluation of CTT for process modeling and a comparison with BPMN.

The understandability of process modeling languages is a central aspect of this work. Some publications already focused on this area (e.g., [RBRB06], [MRC07], [WMR10]). For example, [CGP+05] investigated the impact of model size, [Car06], [GL06] the impact of complexity, [ZPM+11] the impact of hierarchy, and [MNvdA07], [KJS06] the impact of errors on process model understandability.

Closely related to this thesis is also existing research on process modeling notations comparing business process modeling languages. [LKO06], [SAJ+06] and [LYP02] compare the different modeling languages by providing generic metamodels, [Whi04] compares modeling languages in respect of workflow patterns.

Regarding existing work which focused on CEP experiments [PZW10], [ZHPW12], [PWZ+12], [ZPW11] and [TZW+12] can be mentioned. [PZW10] provides an examination of the “Process of Process Modeling” using CEP. [ZHPW12] and [ZPW11] focuses on declarative approaches and “Test Driven Modeling” (TDM) investigating the impact of TDM on process model maintenance. [PWZ+12] compares imperative process modeling with declarative modeling approaches. [TZW+12] investigates understandability of process models.
1.6. Structure

This work is structured in five sections. Section 1 described the introduction and general circumstances of this project.

Section 2 provides further information about Concurrent Task Trees and Cheetah Experimental Platform.

Section 3 contains the features which are requested from the CTT Editor plug-in.

Section 4 describes the creation of the CTT Editor including the algorithms which were developed.

Section 5 will treat the practical use and evaluation of the CTT Editor in the context of an experimental setup and a modeling example including an experimental workflow. In addition, it is described how the CTT Editor has been evaluated in respect to its ease of use.

Section 6 presents discussion, outlook and summary of this master thesis.
Chapter 2.

Backgrounds

This chapter describes some backgrounds and fundamentals which foster understanding of other parts of this thesis. Section 2.1 introduces background and structure of Concurrent Task Trees. Section 2.2 describes the Cheetah Experimental Platform which was used as an underlying framework for the CTT-Editor.

2.1. Concurrent Task Trees

Application Fields of CTT

CTTs were introduced 1997 by [PMM97]. Since then numerous application fields and publications picked up the CTT notation. Especially in the field of User Interface Development the number of publications is not irrelevant (e.g., [SFA07], [SJFS03], [PFW07]). But also for modeling non-user interface related issues CTT was considered a good choice (cf. [MLD08]).

Structure of CT Ts

Concurrent Task Trees are organized, as the name suggests, as a tree structure. Usually, CT Ts are organized top-down, i.e., the root node is put in place at the top of the model. The root node stands for the whole process which is going to be modeled. More precisely, every node in a CTT graph is referred to as a task. The task which is described by the root node can then be further described by
adding more information to the model. This is done by adding children using hierarchical relations. These hierarchical relations are expressed as edges in the graph model. In order to refine the root node reasonably it should have two or more tasks as children. These children should completely describe the root task. Each task can then be further refined. Tasks without children are also called leaves of the tree. In this work tasks which are connected by temporal relations are also referred to as siblings.

**CTT Task Types**

CTTs support a number of different task types. These include user tasks, interaction tasks, cooperation tasks, application tasks and abstract tasks. The graphical differentiation of these task types is done by adding small icons to the task. Every task is further described by a name which labels the task and describes its function (see Figure 2.1).

*User Tasks* describe tasks or activities which are done by a person without interacting with the system or another user (cf. [PMM97] and [Kol11]).

*Interaction Tasks* are employed if the user has to interact with the system or application. By definition this instance of interaction has to be started by the person executing the task [PMM97].

*Cooperation Tasks* describe interactions between more than one user [PMG01].

*Application Tasks* express the execution of the task being performed solely by the application system. Following [PMM97] there still might be an interaction with the user since displaying of information to the user is included in this task type.

*Abstract Tasks* are used if complex tasks are described or if the above mentioned task types do not apply for the task which has to be described.

The type of a parent task is deducted by the type of the child tasks. If all child tasks are of the same type, then the parent task clearly has to have the same
2.1. Concurrent Task Trees

type. If the child tasks use different task types the parent task should be defined as abstract task.

Temporal Relations
In order to add temporal relationship information to tasks with the same parent, task are connected with edges. These edges are called Temporal Relations. In the following a subset of temporal edges which will be used in the CTT Editor will be described (see Section 4.1.1 for an argumentation of the choice of this subset). The Enabling Temporal Relation describes a sequential flow between tasks. The first task is executed, then the second follows. In further parts of this work this edge type will also be referred to as Enabling Edge.
The Interleaving Temporal Relation permits the two connected tasks to be executed in any order [PMM97]. Other authors state that this type of temporal relation also allows parallel, i.e., concurrent execution of both tasks [Kol11]. From here on this relation will be referred to as Concurrent Temporal Relation or simply Concurrent Edge.
The Choice Temporal Relation describes the fact that the user may decide which one of the two connected tasks shall be executed. In the following parts of this work this edge will be referred to as Choice Edge.

Two further types of temporal operators exist in the subset used by the CTT Editor. These two types are not expressed as edges in the model but are directly associated to a task.

An Optional Task is marked as optional by putting the task’s name between two square brackets. For a model execution this means that the task can both be executed or not.

The creation of a repeatable fragment is done by flagging a task as iteration task. An Iteration Task is denoted by adding an asterisk after the name of the task (cf. [PMM97]). This shows that the task might be executed several times in a loop. Following the authors of [PMM97] there are two possibilities for denoting an iteration, either as (infinite) iteration giving no constraints on how often the task has to be performed, or as finite iteration giving an exact number of cycles for that task. On this occasion the difference between denoting a parent task as
iterative or all child tasks of a parent as iterative when using concurrent relations has to be mentioned. Section 3.2 of [PMM97] describes the difference. Generally each task can be set to be optional or repeated independently of its task type. Still an optional task is not allowed to be combined with an optional edge. Figure 2.2 shows a simple example of a CTT using an iterative task (“Write Chapter”) and an optional task (“Publish Thesis”). The iterative task may be repeated several times, the author may write many chapters. The optional task describes an action whose execution is not mandatory.

Figure 2.3 presents another simple example of a Concurrent Task Tree using three different task types. The main purpose of this model is described by the top node, the root node, called Book Room. This main task has two sub-tasks which are connected by an enabling edge, so first the room type has to be chosen and only then the booking can be confirmed. These two tasks are then further refined; The Choose Room Type interaction task has two child tasks which are connected by an optional edge. This means that either Choose Standard Room OR Choose Suite can be executed.

**Ambiguity Problem**

As described by [PMM97] the CTT notations encounters an ambiguity problem. This problem will be an issue when converting CTT models to BPMN in Section 4.3. The ambiguity problem is concerned with tasks which are located at the same level and are connected as siblings. As long as the siblings are connected
with the same edge type no problem will occur. As soon as the siblings have different edge connections the question arises which edge has the highest priority and has to be resolved first. Figure 2.4 shows an ambiguity problem. The four tasks T1 to T4 in Figure 2.4 can be interpreted differently. Reading them from left to right would mean that T1 is executed, afterwards you can choose between executing T2 or T3 and T4 in parallel. The different possibilities are listed here:

- \( T1 \rightarrow ((T2 || T3) ||| T4) \)
- \( (T1 \rightarrow T2) || (T3 ||| T4) \)
- \( ((T1 \rightarrow T2) || T3) ||| T4 \)
- \( T1 \rightarrow (T2 || (T3 ||| T4)) \)

[PMM97] resolved the the ambiguity problem by introducing the following priority of operators defined by LOTOS standard (cf. [BB87]). Choice edges have the highest priority, followed by parallel edges. Enabling edges are the least important. Another solution which is proposed by [PMM97] would be to use an
Chapter 2. Backgrounds

Figure 2.4.: Ambiguity Problem

additional task hierarchy which would resolve the ambiguity problem too. The solution shown in Figure 2.5 gives the higher priority to the concurrent edge by introducing an additional hierarchy level. Assuming that some users will not care about disambiguation the LOTOS standard solution will be applied.

2.2. Cheetah Experimental Platform

The Cheetah Experimental Platform (CEP) (cf. [PZW10]) is a software framework providing support for the conduction of controlled experiments in the field of business process modeling. The focus of CEP lies on fostering the empirical research providing tools for controlled experiments assuring good data quality through user guidance. CEP also provides methods for evaluating gathered data and therefore promoting experimental analysis. CEP will be used as an underlying framework for this thesis.

2.2.1. Provided Functionalities by CEP

Features of CEP are, for example, detailed logging of experiment actions, surveys, tutorials and analysis features. Experiments are conducted as experimental workflows. These workflows are organized in activities which have to be run
through by subjects of the experiment. Such activities in fact are usually surveys about demographic background, domain knowledge, tutorials, feedback sessions and modeling activities. These modeling activities are technically realized by using the GEF (Graphical Editing Framework) which is provided as a part of the Eclipse infrastructure [MDG+04]. Currently CEP supports modeling of BPMN models with the in Section 4.1.1 described set of BPMN items. In order to configure the experiments and enable or disable certain features of the CEP a configuration class can be put in place for each experiment.
Figure 2.5.: Disambiguation
Chapter 3.

Requirements

This section describes the requirements for the creation of the CTT Editor. The distillation of requirements regarding the CTT Editor was done by including different sources. One source for gathering requirements was the input given by the future users of the CTT Editor. As described in [Kol11] the CTT Editor should be used for user experiments aiming at comparing CTT modeling to Business Process Model Notation (BPMN) models. Therefore, some requirements were given by the authors of [Kol11]. These requirements especially concerned the user interface design and functionalities.

Some further requirements were given by the technical circumstances, mainly the integration into Cheetah Experimental Platform (CEP). The Concurrent Task Tree Editor is intended to be an integrated part of this existing framework for experimental research on business process modeling. Features provided by the CEP like the analysis functionality should be able to fully access and incorporate the Concurrent Task Tree Editor (cf. [PZW10]). Furthermore, the CTT Editor has to be included into experimental workflows executed through CEP (see Section 5). Thus, the choice of other technical parameters in respect to, for example, data models or running platforms is given by the CEP framework.
Chapter 3. Requirements

3.1. Modeling Requirements

Modeling of CTT in CEP  First of all, the new software extension for the CEP has to support Concurrent Task Trees. This is the fundamental requirement for the software. Basically, the CTT Editor requires an internal representation of the supported CTT elements. Since the main goal of the implemented editor should be to compare CTT with BPMN, only those elements need to be supported which can be mapped to BPMN (see Section 4.1.1).

Aim of the CTT Editor is to provide a user interface which permits the user to model a process using CTTs. This might be done in more ways: using change primitives or change patterns.

The first possibility is the editing using change primitives (cf. [WRRM08]). In this editing mode the user should be able to perform editing operations using atomic actions (e.g., adding/deleting a node or adding/deleting an edge). Additionally, the user interface should permit the modeler to change the name of the tasks.

The second possibility of editing CTTs with the CTT Editor should be provided by change patterns. Instead of using change primitives like adding a node or an edge more complex change operations should be provided [WRRM08]. Change patterns aggregate change primitives to high-level change operations. The change pattern support shall impose a “correctness by construction” approach. This is done by providing patterns which include correctness-preserving steps. [WRRM08] describes the change pattern approach. CTTs, however, are organized as hierarchical structure and therefore the change patterns have to be modified in some cases. In [Agb11] some of the proposed change patterns from [WRRM08] have already been adapted to CTTs. The CTT Editor should base its change patterns support on this work and should adapt and expand the concepts where necessary.

This should not only permit to compare BPMN and CTT languages using change primitives, but also the application of other combinations: CTT change primitive editing compared with change pattern editing and CTT change pattern editing.
3.1. Modeling Requirements

with BPMN change pattern editing.

**Constraint Requirements**  The syntactical structure of CTTs imposes some constraints which must be followed by the modeler. These constraints, for example, define that each node which is connected by a temporal relation edge must have the same parent (see Section 4.2.1 for an overview of these constraints). This adds a requirement to the CTT Editor: The need of defining rules which permit or deny the creation of certain edges or nodes corresponding to the syntax of CTTs. This feature should be an optional feature, i.e., for some experimental configurations it might be useful to disable the constraint checks in order to be able to analyze the skills of the modeling person participating in a Cheetah experiment.

**Auto-Layouting**  To support users during the creation of CTT models layouting support should be provided. The layouting feature should be invoked, if activated in the CTT Editor, automatically after each modeling step of the user. Thus, the modeler should always be provided with a fully layouted CTT.

**Conversion into BPMN**  The need of converting a CTT model into BPMN is especially given by the planned experiments that shall compare the understandability of CTTs with BPMN models. Therefore, a requested requirement is to offer a conversion tool which takes CTTs and creates the equivalent BPMN graph.

**Correctness Checking**  To ensure correct construction of CTT models a correctness checker should be implemented which is based on the previously mentioned constraints. The correctness checker should be invoked before converting a CTT model to BPMN. The correctness check will be necessary only when change primitive modeling is enabled. Change patterns should always provide a correct CTT model.
3.2. User Interface Requirements

Apart from generally being able to model CTTs with the CTT Editor, user interface requirements were given to foster the model-creation for users.

Folding requirement As described in [Kol11] the folding and unfolding of subtrees of a Concurrent Task Tree model presumably fosters the understandability of business process models by inexperienced users. Therefore, this feature should be added to the CTT Editor. This approach has been denoted as Levelled Exploration by [Kol11]. Additionally, the CTT Editor should provide functionality to fold or unfold the whole CTT based on a selectable level.

Zooming functionality A requirement which was added during the implementation phase of the CTT Editor is the need for a zooming feature. Having only limited space on the modeler’s screen it might be useful to have a zooming functionality. This holds especially for bigger models which exceed the screen size. Surely the editor window has also to be scrollable, but for getting a better overview over the whole CTT model this zooming feature is necessary.

3.3. Backend Requirements

User action recording For supporting the analyzing features of the CEP it is necessary that the CTT Editor provides full logging support. The CEP already contains a sophisticated concept for storing modeling histories which allows to replay and analyze all the steps taken by the user during an experimental workflow. This logging concept has to be integrated into the CTT Editor and fed with data.

Configurability As already mentioned above some features of the CTT Editor should be made switchable in order to permit a flexible configuration for different Cheetah experiments. This should be realized using the existing configuration concept of CEP. Expanding the configuration features and adapting
them for the CTT Editor it should be possible to switch on components like the auto-layouter or single change patterns.
Chapter 4.

Implementation

This chapter describes the concepts regarding architecture, design and implementation of the CTT Editor which will enable CTT support for the Cheetah Experimental Platform (CEP).

4.1. Architectural Overview and Integration into Cheetah Platform

The implementation of the CTT Editor is based on the BPMN Modeler of CEP. CEP offers a plug-in concept which will be used for attaching the CTT Editor.
Chapter 4. Implementation

As depicted in Figure 4.1 the basis for the CEP is provided by the Eclipse Rich Client Platform (RCP), the Graphical Editing Framework [MDG⁺04] and other third party libraries. Based on the CEP the BPMN Modeler was created: it provides support for modeling BPMN and an architectural basis for graphical representation of nodes and edges. The CTT Editor uses provided structures from all the underlying architectural layers as well as the BPMN Modeler from which the graph support will be reused.

4.1.1. Technical Integration into Cheetah Experimental Platform Architecture

Existing Architecture  As already mentioned the CTT Editor is included into CEP as a plug-in. The Eclipse RCP offers this plug-in architecture. Since the CTT Editor shall offer similar functionality as the BPMN Modeler to enable the comparison of CTT and BPMN, the CTT Editor was designed according to existing functionality and features of the BPMN Modeler (cf. Figure 4.2 and Figure 4.3 for a comparison of the GUIs of the two editor components). The Graphical Editing Framework (GEF, cf. [MDG⁺04]) is used as a basis for the BPMN Modeler and therefore also for the CTT Editor. GEF provides a Model-View-Controller (MVC) design pattern [GHJ94] which is further used both in the BPMN Modeler feature of CEP as well as in the CTT Editor. According to the MVC design pattern data models can be put in place and changes to the data models can be controlled by the controller classes. The view classes provide a visualization layer for the data model.

The Cheetah Experimental Platform uses GEF for defining node and edge elements which are contained in a graph object. The node elements represent the tasks of a CTT. The edges stand for the hierarchical or temporal relations and the graph contains the sum of nodes and edges. The constellation of these classes is called the data model.
4.1. Architectural Overview and Integration into Cheetah Platform

Extension and re-use of the existing architecture  The integration of the CTT Editor into CEP was done through two different channels. One concept of integration is called extension points. The Eclipse Rich Client Platform offers this way of integrating classes through this XML-based interface. The usage of extension points was the first choice for integrating the CTT Editor into CEP. Figure 4.4 shows which extension points were provided and used in order to integrate the CTT Editor into CEP.

The second way of extending the existing CEP and BPMN Modeler architecture is by expanding the current class structure. The adaptation of the existing CEP architecture results in an extended data model for the CTT Editor described by Figure 4.5.

Figure 4.2.: BPMN Modeler GUI
Chapter 4. Implementation

Figure 4.3.: CTT Editor GUI

**Used Set of Nodes and Edges**  The subset of supported CTT elements by the CTT Editor was chosen based on which BPMN elements are already supported by CEP. This correspondence is important in respect of the conversion functionality since the CTT models shall be convertible into BPMN models. Table 4.1 shows the set of supported elements of both languages.
4.1. Architectural Overview and Integration into Cheetah Platform

Figure 4.4.: Extension Points used to attach to Eclipse RCP and CEP

<table>
<thead>
<tr>
<th>BPMN Elements</th>
<th>CTT Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity</td>
<td>Abstract Task</td>
</tr>
<tr>
<td>XOR-Gateway</td>
<td>Interaction Task</td>
</tr>
<tr>
<td>AND-Gateway</td>
<td>User Task</td>
</tr>
<tr>
<td>Start-Event</td>
<td>Application Task</td>
</tr>
<tr>
<td>End-Event</td>
<td>Cooperation Task</td>
</tr>
<tr>
<td>Sequential Flow</td>
<td>Enabling Relation</td>
</tr>
<tr>
<td>Hierarchical Activity</td>
<td>Concurrent (Interleaving) Relation</td>
</tr>
<tr>
<td></td>
<td>Choice Relation</td>
</tr>
<tr>
<td></td>
<td>Optional Flag</td>
</tr>
<tr>
<td></td>
<td>Iteration Flag</td>
</tr>
</tbody>
</table>

Table 4.1.: Table of supported BPMN- and CTT- Elements
Chapter 4. Implementation

Figure 4.5.: Data Model of the CTT Editor
Class architecture  The classes depicted in Figure 4.5 marked by thicker blue borders are already existing classes provided by the CEP which were extended by CTT Editor classes in order to provide a data model depicting the CTT language. The classes Graph, Node and Edge are the basis for depicting process models using graphical graph editors in CEP. The graph is the central entity containing all the further graphical elements of a process model. A graph contains many nodes which are connected by edges. Each edge connects one source node with a target node. This general concept holds also for CTT graphs. Node elements in the context of CTT depict the tasks. To adapt the general concept of graphs for CTTs a number of extending child classes were introduced. The general abstract class CTTNode holds all the shared attributes and methods for CTT tasks. This class is then further extended by the five different classes depicting the five types of CTT tasks.

The same design concept is applied to the Edge class which is extended by the abstract CTTEdge class providing common attributes and methods for the CTT relations. The six child classes which extend the abstract class model depict the different relations used in CTTs.

The CTTEditor class is connected to the Graph class as it is responsible for depicting the Graph component on the user interface. In order to be able to use the CTT Editor in the CEP workflow the AbstractModelingActivity has to be extended by the CTTModelingActivity to enable CTT modeling as part of a CEP workflow.

4.1.2. Providing Analysis Capabilities for the CTT Editor

Commands  In order to realize the recording of user actions requirement described in Section 3.3 the concept of commands will be applied. It supports gathering of data about the modeling process and modeling languages. As described by [MDG+04] the Graphical Editing Framework provides this command-solution for executing actions in a controlled way. Each user action irrespective of whether it is a node-creation-operation, an edge-deletion or a conversion into BPMN is executed through a command. This command is passed on to the command-stack
Chapter 4. Implementation

provided by the GEF and then executed. An advantage of the command-stack is that, for example, undo-operations can be registered and the changes since the last save are traceable. An advantage is also reflected when it comes to a replay action described in the paragraphs beneath. Full history of each user action is a central concept of CEP. Complex actions which consist of single commands can be mapped to one single step using the command-stack. This is needed, for example, when using change patterns or when layouting the model.

During the implementation of the CTT Editor this command-concept has to be followed for each action that should be registered and logged for further data analysis.

Some commands already provided by CEP could be reused. Table 4.2 shows which commands had to be introduced by the CTT Editor to realize the recording of user actions requirement.

<table>
<thead>
<tr>
<th>Command Name</th>
<th>Functionality</th>
</tr>
</thead>
<tbody>
<tr>
<td>CheckCorrectnessCommand</td>
<td>Triggers a Correctness Check of the CTT</td>
</tr>
<tr>
<td>ConvertToBPMNCommand</td>
<td>Triggers the Conversion of the CTT to Flat BPMN</td>
</tr>
<tr>
<td>ConvertToHierarchicalBPMNCommand</td>
<td>Triggers the Conversion of the CTT to Hierarchical BPMN</td>
</tr>
<tr>
<td>ToggleIterationFlagCommand</td>
<td>Triggers the addition or removal of an iteration flag on a CTT node</td>
</tr>
<tr>
<td>ToggleOptionFlagCommand</td>
<td>Triggers the addition or removal of an optional flag on a CTT node</td>
</tr>
<tr>
<td>TreeCollapseCommand</td>
<td>Triggers the folding or unfolding of CTT nodes</td>
</tr>
<tr>
<td>ChangeNodeTypeCommand</td>
<td>Triggers the change of task type for a selected CTT task</td>
</tr>
<tr>
<td>HierarchyFoldCommand</td>
<td>Triggers the visibility of whole hierarchy levels of a CTT</td>
</tr>
<tr>
<td>StructureCTTCommand</td>
<td>Triggers the layout feature for the actual CTT</td>
</tr>
</tbody>
</table>

Table 4.2: Table of created GEF commands

Logging In order to deduct any conclusions out of the conducted experiments we have to retain all user actions for later analysis. All the above described commands have to be logged permanently in order to foster analysis of the experiments. Every command describing a user action is executed through the command-stack of GEF. The logging itself is done by creating so called AuditTrailEntries which describe the change on the business process model. An Au-
ditTrailEntry may contain a lot of additional information describing the action. Main information always associated with an AuditTrailEntry are a timestamp when the command was executed, which element was affected and the type of action or command which has to be stored. Also a generic list of attributes is associated with each AuditTrailEntry. Usual attributes are the type of edge or node which is created, the duration of a computational task, which is triggered by the command, location information and etc. All information that is needed to analyze the modeling activity and to fully retrace the user actions has to be added to the AuditTrailEntry.

The AuditTrailEntry is then logged and passed on to all LogListeners which are either in charge to store the AuditTrailEntry or are using this notification channel in order to notice a change of the model. The notification channel is, for example, used by the CTT Editor when triggering an automatic layout action. In this case an additional attribute stored in the AuditTrailEntry is the information that the action was triggered by the CTT Editor itself and not by the user.

The logging by the CEP is usually done in two different ways (cf. Figure 4.6). Either a central database is reachable and can be used or the AuditTrailEntries are stored in a local MXML-file [MXM].

Replay In order to be able to review the action the user has taken while modeling a replay-function is available in CEP. This permits to retrace the users modeling process by replaying the single actions the user has taken.

The integration of the replay feature into the CTT Editor was pretty simple. Accessing the underlying architectural layer of CEP, Eclipse RCP and GEF the replay functionality can be enabled by just setting a flag in the configuration class. Additionally, some commands which usually are triggered automatically have to be disabled while replaying an action since the commands are already stored in the command-stack provided by GEF.

Export In a standard configuration the central datastore is a MySQL Database [Kof07]. In future the CEP might be used not only at the University of Innsbruck
and therefore more than the existing database might be put in place. It is recomme ndable to have locally associated databases in order to be sure that experiments can be run even if no Internet connectivity, i.e., a connection to a single central database, is given.

In order to transport data from one database to another the functionality of exporting single processes might be a useful. CEP provides such exporting features providing data as a MXML [MXML] or CSV [Sha05] formatted file. These files can be used to transfer data from one database to another and for using single processes in other applications.
4.2. Assuring Model Correctness

4.2.1. Constraints in Concurrent Task Trees

Concurrent Task Trees follow a certain syntax. These syntactical rules can be used to prevent that a user creates incorrect models which contradict the syntactical specifications of CTTs. This section will point out rules related to edges in CTT models and how they were integrated into the CTT Editor in order to avoid malformed models.

The constraint rules for edge creation are only put in place if the user is using change primitives for editing the CTT graph. During a modeling session, which
uses change patterns, such rules are not required since change patterns imply these rules (cf. Section 4.5).

**Edge Constraint Rules** This paragraph describes the rules applied in the CTT Editor. These rules help the CTT Editor user to model correctly. The following conceptional rules depicted in Table 4.3 could be extracted from the general syntax of CTT and will be applied in the CTT Editor in order to foster correct modeling. The extraction process was done by analyzing syntactical, but also semantical properties of the CTT language. As a basis for gathering the rules the work of [PMM97], [Pat99a], [Kol11], [Agb11] and [NNC06] were used.

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Rule name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Temporal Relation Sourcing</td>
<td>A node is allowed to have only one temporal relation sourcing from it.</td>
</tr>
<tr>
<td>2</td>
<td>Optionality 1</td>
<td>A node marked optional is not allowed to have a sourcing choice edge.</td>
</tr>
<tr>
<td>3</td>
<td>Optionality 2</td>
<td>A node marked optional is not allowed to have a targeting choice edge.</td>
</tr>
<tr>
<td>4</td>
<td>Unique Relation</td>
<td>Two edges are not allowed to have the same source and target node.</td>
</tr>
<tr>
<td>5</td>
<td>Inverted Relation</td>
<td>Edges between the same two nodes are not allowed even if facing the opposite direction.</td>
</tr>
<tr>
<td>6</td>
<td>Unique Hierarchical Relation</td>
<td>A node is allowed to have only one single incoming hierarchical edge.</td>
</tr>
<tr>
<td>7</td>
<td>Hierarchical Dependency</td>
<td>A new hierarchical edge targeting a node which has temporal relations must source from the same parent of the temporal siblings.</td>
</tr>
<tr>
<td>8</td>
<td>Hierarchical Dependency 2</td>
<td>A new temporal edge is only allowed if the targeted node has the same parent as the source node or has no parent yet.</td>
</tr>
<tr>
<td>9</td>
<td>Unique Temporal Relation</td>
<td>The target node of a temporal relation is not allowed to have any other temporal relation edge pointing to it.</td>
</tr>
<tr>
<td>10</td>
<td>Indirect Temporal Relations</td>
<td>Temporal relations are not allowed to create loops.</td>
</tr>
</tbody>
</table>

Table 4.3.: Edge Constraints for Concurrent Task Trees

The implementation of edge constraint rules is based on the GEF functionality called *Edit Policies* which was already adapted by the developers of the CEP.
4.2. Assuring Model Correctness

These edit policies are called each time the user hovers over a node. If the constraints of Table 4.3 are fulfilled, the edit policy permits the edge creation.

In addition, based on rule 7 and 8 of Table 4.3 a feature was implemented which auto-completes missing edges. The modeling process is potentially sped up and the occurrence of modeling errors reduced. Figure 4.8 shows the functionality of the Edge Creation Assistant.

4.2.2. Correctness Checking of CTTs

Many syntactical problems can already be prevented by enforcing the edge constraints described in Section 4.2.1. However, these rules are not sufficient to guarantee syntactical correctness. As a consequence, additional correctness checks are needed ex-post to ensure the correct construction of CTT models. In particular, in addition to the edge constraints further rules as depicted in Table 4.4 have to be enforced.

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>11</td>
<td>A node must have one hierarchical edge pointing to it (except root task).</td>
</tr>
<tr>
<td>12</td>
<td>A CTT shall have only one single root task.</td>
</tr>
<tr>
<td>13</td>
<td>Any node must have only one temporal relation pointing to it.</td>
</tr>
<tr>
<td>14</td>
<td>Only one node among all children of the same parent is allowed not to have a temporal relation targeting him.</td>
</tr>
</tbody>
</table>

Table 4.4.: Correctness Constraints for Concurrent Task Trees

1According to [Pat99b] optional task can not be connected with choice edges
2See footnote 1
Chapter 4. Implementation

Figure 4.8 depicts an example showing an incorrect CTT on the left hand side of the figure. The right hand side shows how it should be when modeled correctly. The incorrectness of the model at the left hand side is caused by missing hierarchical edges, Rule 11 is disrespected. Each time the CTT model is checked the following steps are executed in order to convert the existing CTT structure, which was provided by the user, into an internal representation of a CTT.

**Algorithm**

First of all for each existing CTT node the parent is computed. Rule 11 and 12 are enforced during this step. Figure 4.9 shows a CTT contradicting Rule 11.

![Figure 4.9: Incorrect CTT contradicting Rule 11](image)

If a node does not have a hierarchical relation targeting it, the node represents the root task of the CTT. Figure 4.10 shows an incorrect CTT having more than one root task (contradicting Rule 12).

After this vertical traversing, the horizontal connections are analyzed. Rule 13 and 14 are enforced during this step. If any node has more than one temporal relation pointing on it Rule 13 is violated. Figure 4.11 shows an incorrect model violating this rule.
4.2. Assuring Model Correctness

Figure 4.10.: Incorrect CTT contradicting Rule 12

Figure 4.11.: Model contradicting Rule 13

Figure 4.12. shows a CTT model violating Rule 14. If more than one child of a parent task has no temporal relation pointing on it the model is syntactically incorrect.

Figure 4.12.: Model contradicting Rule 14
Figure 4.13 contains an example showing sequential and hierarchical numerations which will be further used in the layout algorithm in Section 4.4. For each node two numeration-values are determined. The hierarchical level describes the vertical position on the tree, the sequential level the horizontal position.

4.3. Conversion of CTT into BPMN

The conversion of CTT into BPMN is needed to enable experiments comparing the understandability of both modeling languages.

A prerequisite for a conversion of a CTT into a BPMN model are the correctness and consistencies of the model. Only if the model corresponds to the CTT syntax the conversion can be successful. The functionality of checking the correctness of a CTT already described in Section 4.2.2 is used to ensure that the model is correct. If the CTT is incorrect the conversion process is aborted.
Converting CTT models into BPMN requires a mapping for the different CTT elements to corresponding BPMN elements. For this, CTT modeling elements are restricted to the set of elements described in Section 4.1.1 that can be mapped to BPMN. In the following the mapping will be described in detail.

**Flat and Hierarchical BPMN**

The authors of [Kol11] are providing an approach for mapping Concurrent Task Tree models into BPMN. Flat BPMN has no hierarchical information stored in the model, i.e., no support for sub-processes and no activities which can be further refined. The whole process is depicted in one single graph. Hierarchical BPMN supports hierarchical structures in the model. A root BPMN graph describes the most abstract process which consists of BPMN activities and edges describing the control-flows. Some of the activities in the root graph might describe more complex processes and are therefore marked as hierarchical BPMN activities. These hierarchical BPMN activities are linked to other BPMN graphs that refine the hierarchical BPMN activity of the first graph. This graph then, of course, can again contain hierarchical activities linked to further graphs.

Converting a CTT into a *flat* BPMN model imposes that some information contained in the CTT model has to be discarded. The conversion algorithm for the flat BPMN model takes only the leaves of the CTT and interprets them as the activities of the BPMN model. The remaining CTT tasks will not be considered in a *flat* BPMN model.

The conversion algorithm for a *hierarchical* BPMN model considers all intermediate CTT tasks, i.e., tasks having children tasks, and maps them to hierarchical BPMN activities. Only the leave CTT tasks are mapped to regular BPMN activities.

Before discussing the conversion algorithm itself the mapping of CTT elements into corresponding BPMN elements has to be clarified. Table 4.5 shows the corresponding BPMN elements for CTT elements when converting a CTT graph.
Chapter 4. Implementation

Converting tasks  A task in a CTT corresponds to an activity in BPMN, i.e., the different task types existing in CTT and supported by the CTT Editor are always mapped to a BPMN activity. In fact using this conversion strategy a certain amount of information is lost. For now this has to be compensated by the user by choosing meaningful labels for tasks.

Converting edges  Generally, there are two different types of edges in CTTs, hierarchical edges and temporal relations (enabling-, choice- and concurrent-edges). Hierarchical edges can be omitted when converting a CTT into a flat BPMN. Enabling edges are sequential edges, the nodes connected together with that edge are executed sequentially. In fact, the conversion into BPMN of two CTT tasks that are connected with an enabling edge consists in simply linking the two new activities with a BPMN control-flow edge (see Table 4.5). A concurrent edge has to be converted by creating a parallel structure that is started by an AND-Split and finished with an AND-Join. The choice edge corresponds to the concurrent edge with the only difference that this edge type is mapped to XOR-Gateways.

Another CTT construct which has to be mapped into BPMN are the iteration and option flags that can be associated with each CTT task. If a CTT task is marked as iterative it will be included into two BPMN-XOR-Gateways which are connected with each other by a backward control-flow (see Table 4.5). Optional CTT tasks will also be mapped as BPMN activity in the middle of two XOR-Gateways, the only difference is that the control-flow is pointing in a forward direction (see Table 4.5). In BPMN 2.0 [Obj11] there are different constructs for associating an activity to a loop or marking it as optional activity. Still the BPMN set offered by the CEP is a reduced one and therefore this semantical meaning has to be depicted by using XOR-Gateways.
### 4.3. Conversion of CTT into BPMN

<table>
<thead>
<tr>
<th>CTT Element</th>
<th>BPMN Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract Task</td>
<td>Activity</td>
</tr>
<tr>
<td>Iteration Task</td>
<td></td>
</tr>
<tr>
<td>Application Task</td>
<td></td>
</tr>
<tr>
<td>User Task</td>
<td></td>
</tr>
<tr>
<td>Cooperation Task</td>
<td></td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Enabling Edge</td>
<td>Control-Sequence</td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Choice Edge</td>
<td>Two XOR-Gateways surrounding the two connected tasks</td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Concurrent Edge</td>
<td>Two AND-Gateways surrounding the two connected tasks</td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Hierarchical Edge</td>
<td>Irrelevant for conversion</td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Optional Flagged Task</td>
<td>Two XOR-Gateways surrounding the task with a control-flow bypassing the task</td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
<tr>
<td>Iterative Flagged Task</td>
<td>Two XOR-Gateways surrounding the task with a control-flow bypassing the task in opposite direction</td>
</tr>
<tr>
<td></td>
<td><a href="#">Diagram</a></td>
</tr>
</tbody>
</table>

Table 4.5.: Mapping Table for CTT to BPMN Conversion
Chapter 4. Implementation

When converting edges into BPMN structures the *Ambiguity Problem* has to be considered. If an ambiguous model is supplied, the conversion algorithm has to know the priority rules described in Section 2.1.

Applying the BPMN flat conversion algorithm (described beneath) to the model shown in Figure 2.4 will result in a BPMN model depicted in Figure 4.14.

4.3.1. Flat Conversion Algorithm

The flat conversion algorithm for a flat BPMN graph consists of the following steps:

- Determination and conversion of the root node
- ConvertSubTree (Recursive)
- Creation of Node-Group-Nodes
- Convert CTT tasks of a hierarchy
- Convert Node-Group-Nodes
- Recursion of ConvertSubTree for children

**Determination and conversion of the root node** A prerequisite for starting the conversion of a CTT model is to know the root node of the CTT since the conversion traverses the CTT from the top. The root node is determined by the correctness-check which is a prerequisite for the conversion (see Section 4.2.2). This node is the only node in a correct CTT, which does not have a hierarchical edge pointing on it. In a next step a BPMN-Start-Event-Node is created which defines the starting point of the BPMN process. As soon as the root node is known it will be converted according to the mapping table (Table 4.5). This is done by passing the root node to the *convertSubTree*-method. At the very end of the *convertSubTree*-method a BPMN-End-Event-Node is added.

**ConvertSubTree** If the passed node has no children or is folded the *convertSubTree*-method converts the passed node into a BPMN activity according to Table 4.5. If the passed node has children, they have to be analyzed (creation...
4.3. Conversion of CTT into BPMN

Creation of Node-Group-Nodes If the node passed to the convertSubTree-method has children abstract Node-Group are created. This is necessary if a child task is marked either optional or iterative and is done because at this point it is not known whether they will be part of a more complex construct. In short Node-Groups are placeholders in CTT for CTT tasks or a group of them, which will need further attention during the conversion.

At this point the priority of the edges according to the ambiguity-problem-solution (cf. Section 1.1) has to be considered when adding children to Node-Groups which are connected by choice- or concurrent- edges. The replacement of CTT tasks connected with either choice- or concurrent- edges works the following way: All tasks that are connected by the same type of edge without interruption are added to a new Node-Group element. This includes the source and the target of the edge. The type of the edge is stored in the Node-Group element and the tasks are stored in a sorted list according to their sequence level (cf. Section 4.2.2). The Node-Group element replaces all the CTT tasks that are stored in it. The edges which were pointing to the first and the last

Figure 4.14.: Flat Conversion of CTT in Figure 2.4

of Node-Group-Nodes described beneath) and then further converted (recursive call of convertSubTree-method).
node which were replaced by the Node-Group are now targeting and sourcing the Node-Group.

**Convert all CTT tasks of a hierarchy** After all children have gone through the replacement process the current hierarchy level of the CTT is converted into BPMN. Starting with the most left hand node, i.e., the node which is not targeted by a temporal relation, the conversion is started. If the node is a CTT task and not a Node-Group the conversion works like the `convertSubTree`-method when applied to a node without children. If the next node is a Node-Group the group itself has to be converted and then connected to the previous node.

**Convert Node-Group-Nodes** The conversion of a Node-Group is split up in four parts, depending on which type of Node-Group is currently processed. Four types of Node-Groups were introduced and have to be distinguished in order to guarantee a correct conversion:

- Choice-Edge-Node-Groups
- Concurrent-Edge-Node-Groups
- Option-Flag-Node-Groups
- Iteration-Flag-Node-Groups

Choice-Edge-Node-Groups are converted by creating a XOR-Split which will be stored as start-element of the Node-Group. A XOR-Join element represents the end-element of the Node-Group. All CTT tasks which were associated to this Node-Group are then connected to the XOR-Split and to the XOR-Join.

Concurrent-Edge-Node-Groups are converted the same way as Choice-Edge-Node-Groups. The only difference is that instead of XOR-Gateways AND-Gateways are used.

Option-Flag-Node-Groups and Iteration-Flag-Node-Groups are converted apply-
4.3. Conversion of CTT into BPMN

Recursion of ConvertSubTree for children

Whenever a CTT task has children the convertSubTree-method is called recursively. The result of the convertSubTree-method will be integrated into the new BPMN model.

4.3.2. Hierarchical Conversion Algorithm

The hierarchical conversion algorithm for hierarchical BPMN graphs is quite similar to the algorithm for flat conversion. Nevertheless there are some differences which have to be considered:

Each time a task has children it has not to be translated into a standard BPMN activity but has to be mapped to a hierarchical BPMN activity. The children will have to be converted into an independent BPMN graph which will only be connected by the hierarchical activity. In fact for each sub-tree contained in the CTT model an own BPMN graph is created and further linked to the root-activity. This permits to create BPMN models that do not lose the hierarchical information of the CTT model.

When applying the hierarchical conversion algorithm the conversion of a CTT model results in a much higher number of nodes than a flat conversion would result in.

Layouting

When all tasks and Node-Groups are converted the newly created BPMN graph has to be layouted. CEP already offers some functionality which supports layouting of a BPMN model. Behind this feature the graph is submitted to a web-service offered IBM (cf. [IBM]).

This web-service takes the newly created BPMN graph which currently is not having any structure at all and moves the elements in order to return a layouted BPMN model. A main disadvantage of this feature is that it needs a working
Internet connection. If this service is not reachable the converted BPMN model will not be readable by the user since all elements and edges are positioned on the same spot.

Visualization

To visualize the result of the conversion a view is added to the Eclipse perspective. This particular view is provided by the CEP framework simply expecting a BPMN graph structure. Depending on this structure it visualizes the hierarchical BPMN structure. Editing of the BPMN is not allowed in this view. This choice was taken in order to keep consistency between the two models, i.e., the only editing domain is related to the CTT-model.

Future Work

In a further step a bidirectional conversion could be developed allowing the user to convert also BPMN models into CTT. This would also permit to edit the BPMN graph and then propagate the changes into the CTT model.

4.4. Layouting of CTTs

When change primitive editing is enabled, the CTT Editor allows dragging nodes on the editor canvas and creating of edges with bendpoints as needed. To support the user to lay out the CTT as tree, layouting features have been implemented which can be manually invoked by the user. In addition automatic layout functionality is provided. This feature triggers a re-layout action automatically after each editing step. The user is then always provided with a layouted tree.

Node-Link Layout

Generally, there are many different ways to describe and layout trees [ZMC05]. The most common visualization style for a tree is the node-link diagram [RJT81]. This standard is also used to visualize Concurrent
4.4. Layouting of CTTs

Task Trees since it is one of the most understandable and natural designs for inexperienced users. Nevertheless, it is indisputable that this layout has some disadvantages. As mentioned in [ZMC05] the node-link diagram visualization fails to scale large models (cf. Section 4.6.1).

**Layouting algorithm**

For now the CTT Editor supports only one specific layout algorithm described below. The layout of a CTT in the CTT Editor bases on a symmetric tree layout which is applied to the node-link diagram.

In order to know the coordinates of the nodes on the canvas according to a tree layout the algorithm needs two values for each node. The first one describes how many leaves are connected to the node through hierarchical edges. This value is called *own*. The second variable which has to be determined is how many leaves are located on the left hand side of each node, including leaves from other hierarchical levels. This value is called *left*. In order to compute these values for each node the tree has to be traversed two times.

For the calculation of the absolute coordinates three additional constants and one variable are needed. The maximum width (*maxNwidth*) of a graphical task representation in the CTT Editor is needed to determine the total width (*width*) of the layouted CTT in combination with the configured node horizontal distance (*nHdist*) and the variable containing the number of leaves (*leafnum*). The third constant is the vertical node distance (*nodeVdist*). By applying the following formula the total width of the layout can be computed:

\[
\text{width} = \text{leafnum} \times \text{maxNwidth} + (\text{leafnum} + 1) \times \text{nHdist} \tag{4.1}
\]

The height of the layout depends on the maximum height of the nodes. This is a variable value since the height of each node depends on the length of the describing label, i.e., the number of lines the label causes. Therefore, for each hierarchy level the node with the maximum height is determined and all the other nodes of
this level are aligned centrally during the absolute coordinates calculation. Starting from the highest hierarchy level for each node now the absolute coordinates are computed. Using the score of the left hand nodes (left) and the own score (own) and relating them to the number of leaves of the CTT (leafnum) and the total width of the layout (width) calculated by the first formula (4.1) the x-coordinate (xcoord) of each node can be calculated in formula (4.2):

\[
xcoord = \frac{\text{left}}{\text{leafnum}} \cdot \text{width} + \frac{\text{own}}{\text{leafnum}} \cdot \text{width} - \frac{\text{maxNwidth}}{2}
\] (4.2)

The vertical position of each node is easier computable. It is calculated using the vertical node distance, the cumulated sum of all other heights of the levels above (cumHeight), the maximum height of all nodes on this level (maxNHeight) and the actual height of the node itself (nHeight). The root node’s vertical distance from the border of the editor canvas is defined by the vertical node distance and therefore the initial value of cumHeight is defined as cumHeight = nodeVdist.

\[
ycoord = \text{cumHeight} + \frac{\text{maxNHeight} - \text{nHeight}}{2}
\] (4.3)

For each further iteration the new value of cumHeight is defined by:

\[
\text{cumHeight} = \text{cumHeight} + \text{maxNHeight} + \text{nVdist}
\] (4.4)

The iteration of absolute coordinates calculation is done top-down because of the cumulative growth of the variable cumHeight.

Figure 4.15 shows the calculation based on the graph already depicted on figure 4.13 for three selected CTT tasks.

**Further improvements** As already mentioned above this layouting algorithm computes a symmetric tree which expands space requirements of the tree considerably. Especially trees with broad branches need a lot of space. An improvement which saves space would be the implementation of a balanced tree
4.5. Change Patterns Support

This section describes how change pattern support for CTT has been implemented. [WRRM08] describes the change patterns approach for process modeling languages. CTTs, however, are organized as hierarchical structure and therefore the change patterns described in [WRRM08] cannot be directly applied, but have to be modified in some cases.

In [Agb11] some of the proposed patterns from [WRRM08] have already been adapted to Concurrent Task Trees. The CTT Editor bases its change patterns on this work and will adapt and expand the concepts where necessary.

In the following lines the proposed change patterns from [WRRM08] will be discussed in respect to how they can be applied and which adaptations are needed.
in order to support them in a CTT environment.

**Insert Process Fragment** The insertion of a process fragment in traditional modeling languages can be performed in a threefold way. Either as serial insertion, as parallel insertion or as conditional insertion. This does not hold for hierarchical models as CTTs since a serial, parallel or conditional constraint is defined by the type of edge connecting the task.

In the context of CTTs the insert process fragment pattern is enriched by the additional dimension of hierarchisation. A new process fragment can be added either on the horizontal axis, i.e., as sibling or on the vertical axis. Adding a process fragment vertically offers either the possibility to add it as parent in relation to the actually selected node or as a child (cf. Table 4.6, Table 4.7, Table 4.8).
### Upward Hierarchical Insert of Process Fragment

<table>
<thead>
<tr>
<th>Description</th>
<th>Insertion of a parent node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>The actually existing task is already modeled in a too detailed manner. A new parent task will allow to add a further abstraction layer.</td>
</tr>
<tr>
<td>Selection criterion</td>
<td>All selected items have to be nodes. The selected nodes have to be the full set of child nodes of an existing node.</td>
</tr>
<tr>
<td>Example</td>
<td><img src="image.png" alt="Image of example" /></td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The result of this change pattern is not fully compliant with the CTT syntax since the newly created parent has no sibling. The user is required to add a sibling using other change patterns. The node type of the new node has to be adapted.</td>
</tr>
<tr>
<td>Implementation</td>
<td>This pattern is implemented by the following steps: Creation of the new node (Abstract Task), reconnection of the existing hierarchical edges and creation of a new hierarchical edge to the newly created node.</td>
</tr>
</tbody>
</table>

Table 4.6.: Upward Hierarchical Insert Process Fragment


Chapter 4. Implementation

**Downward Hierarchical Insert of Process Fragment**

<table>
<thead>
<tr>
<th>Description</th>
<th>Insertion of a child node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>The selected task shall be modeled in more detail.</td>
</tr>
<tr>
<td>Selection criterion</td>
<td>Only one node has to be selected. This node is not allowed to have any children (i.e., a leaf).</td>
</tr>
<tr>
<td>Example</td>
<td></td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The node type of the newly created node has to be adapted.</td>
</tr>
<tr>
<td>Implementation</td>
<td>This pattern is implemented by the following steps: Creation of the new node (Abstract Task) creation of a new hierarchical edge to the newly created node.</td>
</tr>
</tbody>
</table>

Table 4.7.: Downward Hierarchical Insert Process Fragment
4.5. Change Patterns Support

**Insert Sibling Process Fragment**

<table>
<thead>
<tr>
<th>Description</th>
<th>Insertion of a sibling node.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Problem</td>
<td>The actually existing model is missing a task.</td>
</tr>
<tr>
<td>Selection criterion</td>
<td>(a) One selected node: This node has to be either a sequence starter or a sequence end. This node is not allowed to be the root node.</td>
</tr>
<tr>
<td></td>
<td>(b) Two selected nodes. These two nodes have to be connected by a temporal edge.</td>
</tr>
<tr>
<td>Example</td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The node- and edge type have to be adapted according to the process the model is describing.</td>
</tr>
<tr>
<td>Implementation</td>
<td>This pattern is implemented by the following steps: Creation of the new node (Abstract Task) creation/reconnection of a new enabling edge to the newly created node.</td>
</tr>
</tbody>
</table>

Table 4.8.: Insert Sibling Process Fragment
Delete process fragment. The deletion of a part of the CTT model does not need any further distinction like the insert patterns. Deleting a fragment of the CTT is straightforward. Still there are some considerations to make which selection criterion is valid. Table 4.9 describes the possible selection criteria for the Delete Process Fragment Pattern.

| Description | Deletion of nodes and connected edges. |
| Problem     | A task has to be skipped or deleted. |
| Selection criterion | The root node might not be selected. |
| (a) One selected node. The selected node has to be either a sequence starter or a sequence end and is not allowed to have any children. |
| (b) The selected nodes have to be a full subset of leaf-children of a task. |
| (c) The selected nodes have to be a complete subtree of one single parent. |

Example

![Diagram of Delete Process Fragment]

Table 4.9.: Delete Process Fragment

Replace Process Fragment. The replace process fragment pattern is not yet supported in CTTs. Still, the two patterns change node type (see Table 4.10) and change edge type (see Table 4.11) are an initial approach to the replace process fragment pattern. It has to be mentioned that in CTTs the change of an edge type is a major change for the process.
4.5. Change Patterns Support

<table>
<thead>
<tr>
<th>Change Node Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Problem</td>
</tr>
<tr>
<td>Selection criterion</td>
</tr>
</tbody>
</table>

Example

**Implementation**
Internally this pattern is implemented by removing the old node and creating a new one with the corresponding type. This pattern constitutes a variant of the pattern *Replace Process Fragment* [WRRM08].

Table 4.10.: Change Node Type

<table>
<thead>
<tr>
<th>Change Edge Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
</tr>
<tr>
<td>Problem</td>
</tr>
<tr>
<td>Selection criterion</td>
</tr>
</tbody>
</table>

Example

**Implementation**
The implementation of this pattern removes the edge between the two nodes and creates a new one having the requested edge type.

Table 4.11.: Change Edge Type

**Embed process fragment in loop**
While in traditional process modeling languages this pattern is implemented by embedding an activity into a loop structure, implementing this pattern in CTT only requires adding an iteration flag to
the selected task. The embed process fragment in loop pattern is called “Toggle Iteration Flag” when applied to CTT.

**Parallelize process fragments** The parallelization of process fragments in traditional modeling languages is done by adding AND-Gateways. In CTT this is reached by changing the temporal relation between the nodes that should be executed in parallel to a concurrent edge. For this, the CTT Editor offers a pattern called “Edge Type” which is described in Table 4.11.

**Embed process fragment in conditional branch** The conditionality in traditional modeling languages like BPMN is achieved differently than in CTTs. The syntax of CTT offers two different types of conditionality. One way of depicting a conditional relation between two tasks is a choice edge. This case is covered by the “Edge Type” pattern (cf. Table 4.11). The second possibility is to mark a task as optional. This means that the task might then be executed optionally,
i.e., the choice has not to be made between two tasks, but only whether to execute the task or not. The marking of a task as optional is done by surrounding its label with two square roots. For this reason in CTT a \textit{Toggle Option Flag} pattern exists (cf. Figure 4.16). The selection criterion for the \textit{Toggle Option Flag} pattern requires a single node to be selected. This node is not allowed to be connected to a choice edge and must not be marked as iterative task.

\textbf{Rename Task} \hspace{1em} In order to support all editing options that are provided by the CTT Editor when editing using change patterns some operations have to be included in the change pattern palette. Apart from the toggle option and iteration flag primitives already mentioned above a \textit{Rename Task} action has to be provided.

\textbf{Comments on Change Pattern and Future Work} \hspace{1em} Actually the change patterns in the CTT Editor are not considering task types. In some cases the change patterns could be further refined including considerations about this issue. A further consideration which should be part of future work is that a father task in a CTT must have either the type of its child nodes, if all the child nodes are from the same type, or it has to be an abstract task if the children are not of the same type. Currently the compliance with this constraint has to be assured by the user.

\textbf{Interplay of Change Patterns with other Features} \hspace{1em} Having a layout functionality as described in Section 4.4 is crucial when using change patterns. After each execution of a pattern the layout algorithm is invoked.

\textbf{Further issues regarding change patterns} \hspace{1em} Currently the CTT Editor automatically creates either enabling edges or abstract tasks if a new element is needed. The user then has to adapt the type of the element to his needs for the
model. This keeps the number of available change patterns straightforward but in some cases does not guarantee for correctness by construction. The pattern assures a syntactically correct CTT but does not ensure that the user adapts the edge type according to the reality he is depicting with the model.

For creating a new task that should be part of a conditional structure, three change patterns have to be executed. Firstly, the new task has to be added probably using the insert sibling pattern. Then the type of the task will have to be changed if the task is not an abstract task. Thirdly, the edge type pattern has to be called in order to create the optional edge.

4.6. Additional Features

4.6.1. Folding and Unfolding of CTT

The requirement of supporting a visualization concept, that permits the user to analyze and view a complex CTT, has to be satisfied. In a first step the fold/unfold feature was implemented.

This functionality fosters the understandability of the CTT model since it does not appear that complex when some of the nodes are hidden. [Kol11] describes such visualization concepts. Figure 4.17 and Figure 4.18 show how the folding works by clicking on the “+” icon of the parent Application Task.
In order to provide a more sophisticated and automated way of exploring CTT using folding and unfolding of tasks a slider has been installed. By moving this slider by one position a whole hierarchy level of the CTT will be folded or unfolded. By changing the sliders position the abstraction of the model is either increased or decreased.

A further, not yet implemented, improvement could be to create the possibility to extract a single, specific level of the CTT model. All child and parent tasks which are not part of this level could be hidden, showing only tasks of one granularity level.

Another feature that might be useful to be implemented is the adaptation of the layout algorithm described in Section 4.4. Currently, this layout algorithm always uses the full CTT to compute the positions of the nodes. The behaviour of this feature could be improved by taking the folded tree as basis for the layout instead of the whole one.

4.6.2. Zooming Feature

Since big Concurrent Task Trees might not be easily visualizable especially on small resolution displays, a zooming functionality was added to the CTT Editor. This feature allows the user to scale the CTT model in order to get a better overview. Changing and editing the model is only possible while the CTT model is not scaled, i.e., the zoom level is at 100%. This limitation might be removed in a future release of the CTT Editor.
Chapter 5.

Evaluation and Running Example

To assure quality of the CTT Editor it has been extensively tested as described in Section 5.1. In addition, it has been applied to a running example as described in Section 5.2. Section 5.3, in turn, describes the setup of the CEP experiment for comparing CTT and BPMN. Finally this chapter contains also an evaluation of the CTT Editor based on the Technology Acceptance Model (TAM, cf. [Dav86], see Section 5.4).

5.1. Testing

5.1.1. Unit-Test

In order to assure good quality of the implementation of the CTT Editor tests were implemented. Complex parts, like the layout algorithm or the conversion functionality, were developed from scratch. For these parts unit tests have been created during and after the implementation of the algorithms itself. JUnit [Bec04] was used as a framework for testing since it is already included in the actual Eclipse distributions. Currently there are about 100 test-cases defined in the ambit of the CTT Editor in order to test the layout algorithm and the conversion into both hierarchical and flat BPMN. [Kol11] published some concepts on how the conversion from CTT to BPMN can be conducted. There are also some other sources like [NNC06] which already converted CTT into other modeling languages like UML. The conversion algorithms incorporated in the CTT
Editor correspond with the concepts presented by other authors.

5.1.2. Evaluation by Domain Experts

A further testing phase was conducted by submitting the CTT Editor to be reviewed by domain experts. The authors of [Kol11] and [Agb11] conducted a user-test and found the CTT Editor to be valid. A main focus during the evaluation by domain expert was put on the conversion algorithm.

5.2. Running Example

To evaluate the proper functioning of the CTT Editor it has been applied to a running example. An adapted version of the example described in Section 4 of [Agb11] has been used. This example was chosen because it covers most of the implemented concepts and therefore provides a good basis for testing the CTT Editor. The example has a suitable degree of complexity for applying the concepts and represents more than a toy example.

Description of the example

The example depicted in Figure 5.1 and Figure 5.2 shows the process of the preparation of a chess-tournament. It describes some preparation steps which are executed in sequence. After these steps the tournament and the gastronomic activities are executed contemporaneously. The tournament activities are further refined since the inscription is done using a computer system. After the tournament has been conducted and evaluated the award ceremony follows.

When using CTT the model’s root task is an abstract task without description since it has children of different task types. Five of its six children are leaf nodes, they do not have any further children. All six child tasks are connected with enabling edges which infers that the tasks have to be accomplished sequentially. The task types of this hierarchy depend on either how the tasks are accomplished or on the fact that they have other child tasks. All tasks but two are user tasks. They are executed by a user without interaction with an information system.
The second task, named *Create Calendar Appointment*, is an interaction task. It describes the creation of a calendar appointment. The creation is apparently done by interacting with a computer system. The fifth task is an abstract task which is further refined by its children tasks. Its children are connected with a concurrent edge, i.e., both tasks can be conducted simultaneously. The left hand task has again two children but is a user task since both children are user tasks. The right hand task is abstract as it has again children of different types. The *Set-up boards* relate concurrently with the interaction task having two children *Computer-Installation* and *Inscription* being executed sequentially. The other two tasks *Tournament* and *Tournament Evaluation* are executed again sequentially.

![Chess Tournament Example from [Agb11]](image1)

Figure 5.1.: Chess Tournament Example from [Agb11]

![CTT Editor Chess Tournament Model](image2)

Figure 5.2.: CTT Editor Chess Tournament Model
Reproduction of the Example and Compliance with Requirements

The example can be reproduced applying change primitive editing mode, but also using change patterns. All features of the CTT Editor were tested during the reproduction of this example in different runs and combinations. If any error was determined it has been solved and the test runs were repeated. Table 5.1 shows the features which were tested and if they were found to be compliant with the requirements (cf. Chapter 3).

The Backend Requirements described in Section 3.3 are tested implicitly. The Configurability Requirement was tested by putting in place the three different configuration options for the different testing scenarios.

The User Action Recording Requirement was tested separately by replaying the modeling steps of all three testing scenarios. The replay worked out without errors.

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Satisfactory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modeling of CTT</td>
<td>Yes</td>
</tr>
<tr>
<td>Change Pattern Support</td>
<td>Yes</td>
</tr>
<tr>
<td>Conversion into BPMN</td>
<td>Yes</td>
</tr>
<tr>
<td>Constraint Requirements</td>
<td>Yes</td>
</tr>
<tr>
<td>Correctness Checking</td>
<td>Yes</td>
</tr>
<tr>
<td>Auto-Layouting</td>
<td>Yes</td>
</tr>
<tr>
<td>Automatic Completion of CTT models</td>
<td>Yes</td>
</tr>
<tr>
<td>Folding Requirement</td>
<td>Yes</td>
</tr>
<tr>
<td>Zooming Requirements</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 5.1.: Requirement Compliance
5.3. Evaluation of the CTT Editor

The practical use for the CTT Editor plug-in will be tested by conducting a controlled experiment. It was part of the assignment for this work to realize an experimental workflow aiming at the comparison of CTT with BPMN. The comparison is done in respect to modeling ease and comprehensibility of the two modeling languages. In the following the experimental setup is outlined in detail.

The experimental setup, which will be used to compare BPMN was provided by Jens Kolb and follows Figure 5.4. At the beginning of the experiment each user has to enter a code which associates him to a certain group. The user is then guided through the whole experimental process and the tasks he has to complete. Before the modeling tasks are called the user is shown a welcome message, thanking for the participation and explaining how to conclude the single workflow steps. Then, depending on the code the user entered, the user is provided with either a BPMN Modeling Task or a CTT Modeling Task.

![Figure 5.3: Flat-Hierarchy BPMN Result](image)

![Figure 5.4: Experimental Workflow](image)
Both tasks require the user to model a process corresponding to a model description (see Appendix A) from scratch. The same model description is provided to both groups, the difference lies in the modeling language. The first group has to model using BPMN, the second one using CTT.

The following six change tasks have to be accomplished using the same modeling language as for the modeling task. For each change task the same model is provided as basis. Figure 5.5 and Figure 5.6 show the two given models for all six change tasks. The change tasks are listed in Table 5.2.

Figure 5.5: BPMN Change Model
5.3. Evaluation of the CTT Editor

Figure 5.6.: CTT Change Model

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Change Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Insert node &quot;X&quot; parallel to node &quot;H&quot;</td>
</tr>
<tr>
<td>2</td>
<td>Insert node &quot;X&quot; as choice to the nodes &quot;D&quot;, &quot;B&quot;, &quot;C&quot;, &quot;E&quot;, &quot;F&quot;, &quot;G&quot;, &quot;H&quot;, &quot;I&quot;, &quot;J&quot;, &quot;K&quot;, &quot;L&quot;, and &quot;M&quot;</td>
</tr>
<tr>
<td>3</td>
<td>Insert node &quot;X&quot; parallel to node &quot;B&quot;</td>
</tr>
<tr>
<td>4</td>
<td>Delete node &quot;L&quot;</td>
</tr>
<tr>
<td>5</td>
<td>Move choice-subtree with the nodes &quot;K&quot;, &quot;L&quot;, and &quot;M&quot; after &quot;C&quot;</td>
</tr>
<tr>
<td>6</td>
<td>Parallelize the nodes &quot;I&quot; and &quot;J&quot;</td>
</tr>
</tbody>
</table>

Table 5.2.: Change Tasks

In a following step in the experimental workflow the users are guided through ten comprehension tasks. Depending on the group the users were assigned to the same model as for the change tasks will be the basis for the comprehension task. Based upon that model the users have to answer the questions shown in table 5.3. After each modeling or comprehension task the user is asked to categorize the mental effort experienced.

At the end of the experimental workflow both user groups are asked for feedback regarding the experiment in order to determine possible improvements for future experiments.

Main response variables for this experiment will be the amount of correct modeling actions as well as correct answers to the comprehension questions. A further response variable will be the time which the users needed to perform the single
## Chapter 5. Evaluation and Running Example

<table>
<thead>
<tr>
<th>Nr.</th>
<th>Comprehension Question</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Which activities follow on node &quot;C&quot;?</td>
</tr>
<tr>
<td>2</td>
<td>Which activities are in parallel with node &quot;F&quot;?</td>
</tr>
<tr>
<td>3</td>
<td>Which activities are alternatively to node &quot;L&quot;?</td>
</tr>
<tr>
<td>4</td>
<td>Which activities follow on node &quot;J&quot;?</td>
</tr>
<tr>
<td>5</td>
<td>Which activities come before node &quot;G&quot;?</td>
</tr>
<tr>
<td>6</td>
<td>Is the following process log valid? &lt;A,L,M,N&gt;</td>
</tr>
<tr>
<td>7</td>
<td>Is the following process log valid? &lt;A,D,E,F,H,G,N&gt;</td>
</tr>
<tr>
<td>8</td>
<td>Is the following process log valid? &lt;A,E,B,F,C,N&gt;</td>
</tr>
<tr>
<td>9</td>
<td>Is the following process log valid? &lt;A,K,L,N&gt;</td>
</tr>
<tr>
<td>10</td>
<td>Is the following process log valid? &lt;A,B,H,G,E,C,F,N&gt;</td>
</tr>
</tbody>
</table>

Table 5.3.: Comprehension Questions

assignments. As an additional response variable mental effort will be considered. This will also provide subjective data regarding understandability and ease of use of both modeling languages.

### 5.4. Evaluation Applying the Technology Acceptance Model

Usability of the CTT Editor is a precondition for the successful execution of the experiment described in Section 5.3. In order to evaluate the CTT Editor in terms of its ease of use a small set of users was asked to accomplish the tasks described in Section 5.3 using the CTT Editor and to assess its ease of use. For this the Technology Acceptance Model (TAM, [Da86]) was applied. The TAM focuses on two factors describing the perception of users regarding an application:

- **Perceived Usefulness** (USEF) is a factor that describes to which extend users believe that the application will help them to perform a better job [Da89].

- **Perceived Ease of Use** (EOU) describes how hard the application is to use and if the performance benefits of the usage are worth the effort using the application [Da89].
5.4 Evaluation Applying the Technology Acceptance Model

The evaluation of the CTT Editor will focus on the Perceived Ease of Use since this factor describes the usability of the CTT Editor more generally and some of the process modelers are not confronted with applying the CTT Editor in their daily business. Therefore, the Perceived Usefulness will not be evaluated.

In order to gather data about the Perceived Ease of Use the experimental setup (cf. Section 5.3) was expanded by adding an additional questionnaire at the end of the experiment. Table 5.4 shows which questions were used. The selectable responses correspond with a 7-point Likert scale [Lik32] using values ranging from 1 'Extremely Likely' to 7 'Extremely Unlikely'.

The modelers were provided with the extended experiment, all persons had to run through only the CTT part of the experimental workflow, since the TAM data is needed for the CTT Editor part only.

Table 5.4 shows which results the TAM questionnaire showed. The responses regarding the Ease of Use statements pointed out a total average of 2.09. This shows that the CTT Editor is perceived to be easy to use at the grade “quite likely”.

<table>
<thead>
<tr>
<th>Statement</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Learning to operate CTT would be easy for me.</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I would find it easy to get CTT to do what I want it to do.</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>My interaction with CTT would be clear and understandable.</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I would find CTT to be flexible to interact with.</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>It would be easy for me to become skillful at using CTT.</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>I would find CTT easy to use.</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.4.: Perceived Ease of Use Statements
Chapter 6.

Summary and Outlook

There is a thoroughly notable range of different modeling languages in the field of Business Process Management [LK06]. Still there are diversities and singularities within modeling languages which were defined for various application fields. In order to determine which group of users, depending on domain origin, domain knowledge, business process expertise and other factors will be more comfortable with a specific modeling language a way to compare the modeling languages has to be found. Which modeling language and which user group fit best has to be evaluated empirically. The CTT Editor represents a step in this direction by enabling the comparison between BPMN and CTT using the Cheetah Experimental Platform [PZW10].

The creation of a plug-in for the CEP enabling modeling of CTT models according to requirements from the environment as defined by the IS research framework of Hevner et. al. [HIMPR04] has led to the provision of an additional powerful feature for the CEP. The CTT Editor provides functionalities comparable to the existing BPMN modeler feature of the CEP. Special features as the conversion algorithm into BPMN as well as the correctness checker of the CTT model were added and will permit the users to design CTT models more easily. To foster usability especially in respect to End-User-Modelling the folding feature (see Section 4.6.1) and zooming feature (see Section 4.6.2) were put in place.
The CTT Editor was subject to a running example (see Section 5.2) and a small experiment (see Section 5.3) in order to determine its ease of use. The result shows that the CTT Editor is a usable tool and will be a good basis for comparing the two modeling notations CTT and BPMN.

As already suggested in the implementation Chapter 4 future additional, new features will help to propagate the usage of CEP in ample research fields. Future expansions of the CTT Editor comprise the inverse conversion from BPMN to CTT or additional automatic layouting features.
Appendix A.

Model-Description for Experimental Setup

The credit application starts with the creation of a written application/request by the customer. After making an application the customer’s data will be captured by the clerk in three steps. First the customer informs about his domicile, about his financial and about his marital situation. For the capturing of the financial situation, the customer has to state his earnings and expenses. Next, the marital situation indicates whether the customer is married or unmarried. If he is married, it has to be declared whether he has children or not and in case of existing children’s their names are documented. Lastly the credit institute has to check the creditworthiness of the customer. Therefore two results exists: whether the creditworthiness is positive or negative. If the result is positive, the credit institute is allowed to create an credit agreement, which is afterwards signed by the customer. If the result is negative, the credit institute rejects the application or offers another credit application.  

1This Model-Description was provided by the group in charge of the empirical experiment at the DBIS institute of the University of Ulm.
# List of Figures

1.1. IS Research Framework from [HMPR04] .......................... 9

2.1. CTT Task Types ...................................................... 14
2.2. Simple CTT Example showing Iteration and Option Flags ........ 16
2.3. Simple CTT Example .................................................. 17
2.4. Ambiguity Problem .................................................. 18
2.5. Disambiguation ....................................................... 20

4.1. Integration of the CTT Editor with CEP and BPMN Modeler ..... 27
4.2. BPMN Modeler GUI .................................................. 29
4.3. CTT Editor GUI ....................................................... 30
4.4. Extension Points used to attach to Eclipse RCP and CEP ........ 31
4.5. Data Model of the CTT Editor ..................................... 32
4.6. Logging of Modeling Steps .......................................... 36
4.7. User Interface with BPMN Conversion ............................ 37
4.8. Edge Creation Assistant Functionality ............................. 39
4.9. Incorrect CTT contradicting Rule 11 .............................. 40
4.10. Incorrect CTT contradicting Rule 12 .............................. 41
4.11. Model contradicting Rule 13 ...................................... 41
4.12. Model contradicting Rule 14 ...................................... 41
4.13. Task Numeration ..................................................... 42
4.14. Flat Conversion of CTT in Figure 2.4 ............................ 47
4.15. Layout Calculation based on Figure 4.13 for three selected CTT
     Tasks ................................................................. 53
4.16. Change Pattern Dialog ............................................. 60
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.17</td>
<td>Unfolded Simple CTT Model</td>
<td>62</td>
</tr>
<tr>
<td>4.18</td>
<td>Folded Simple CTT Model of Fig. 4.17</td>
<td>62</td>
</tr>
<tr>
<td>5.1</td>
<td>Chess Tournament Example from [Agb11]</td>
<td>67</td>
</tr>
<tr>
<td>5.2</td>
<td>CTT Editor Chess Tournament Model</td>
<td>67</td>
</tr>
<tr>
<td>5.3</td>
<td>Flat-Hierarchy BPMN Result</td>
<td>69</td>
</tr>
<tr>
<td>5.4</td>
<td>Experimental Workflow</td>
<td>69</td>
</tr>
<tr>
<td>5.5</td>
<td>BPMN Change Model</td>
<td>70</td>
</tr>
<tr>
<td>5.6</td>
<td>CTT Change Model</td>
<td>71</td>
</tr>
</tbody>
</table>
List of Tables

4.1. Table of supported BPMN- and CTT- Elements ................. 31
4.2. Table of created GEF commands ................................. 34
4.3. Edge Constraints for Concurrent Task Trees ................. 38
4.4. Correctness Constraints for Concurrent Task Trees ........... 39
4.5. Mapping Table for CTT to BPMN Conversion .................. 45
4.6. Upward Hierarchical Insert Process Fragment ................. 55
4.7. Downward Hierarchical Insert Process Fragment ............... 56
4.8. Insert Sibling Process Fragment ................................. 57
4.9. Delete Process Fragment ........................................ 58
4.10. Change Node Type ............................................. 59
4.11. Change Edge Type ............................................. 59

5.1. Requirement Compliance ........................................ 68
5.2. Change Tasks .................................................. 71
5.3. Comprehension Questions ...................................... 72
5.4. Perceived Ease of Use Statements ............................... 73
Bibliography


Bibliography

1993.


Bibliography


Bibliography


Declaration of Authorship

I hereby declare that this Master’s thesis has been written only by the under-signed and without any assistance from third parties. I confirm that no sources have been used in the preparation of this thesis other than those indicated in the thesis itself.
This Master’s thesis has heretofore not been submitted or published elsewhere, neither in its present form, nor in a similar version.

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Masterarbeit selbständig angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.
Die Arbeit wurde bisher weder in gleicher noch in ähnlicher Form einer anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

________________________________________  ________________________________
Date                                      Signature (Thomas Gress)