FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Extension of the Business Process Execution Language (BPEL) with probabilistic time management of choreographies and time constraints in workflow systems

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Wirtschaftsinformatik

eingereicht von

Christian Österle
Matrikelnummer 0226594

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: o.Univ.Prof. Dipl.Ing. Dr. A Min Tjoa
Mitwirkung: Dipl.Ing. Dr. Amirreza Tahamtan

Wien, 25.01.2011 _____ (Unterschrift Verfasser) _____ (Unterschrift Betreuer)

Eidesstattliche Erklärung

Name: Christian Österle
Adresse: Gernotgasse 9/15, 1150 Wien

"Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe."

Wien, am 25. Januar 2011 .....

(Christian Österle)

Abstract

WS-BPEL (=Web Service Business Process Execution Language) provides a language for describing the behavior of a business process based on interactions between the process and its partners. The interactions of partners occurs through web service interfaces which ensure interoperability between applications by using web standards.
Time management and temporal conformance is an important criteria for business processes. It ensures that activities are performed in a timely manner and that the right information is delivered to the right activities at the right time, such that overall temporal restrictions are satisfied and no deadline is violated. Deadline violations increase the execution time and cost of business processes because they require some type of exception handling.
The contribution of this diploma thesis will be the implementation of a tool that enables users to augment WS-BPEL-processes with temporal information in design time and a run time component for checking the temporal behavior of the processes at run time.
In design time a valid temporal interval for (activities of) each process is calculated with consideration for the structure of each process and the interactions between different processes and then checked if the model is temporally feasible, i.e. there is a solution satisfying all temporal constraints. Calculating the temporal intervals and checking the temporal conformance will be performed for two scenarios. The interval-based scenario, which allows a variable duration of activities within an interval. The stochastic scenario, that allows the definition of different activity durations with different probabilities and also takes the conditional branching behavior into account.
In run time, it is checked if the activities are executed within the valid calculated intervals at design time. Therefore the time points of all processes are mapped to a calendar with current dates and times. Then, the temporal status of each process instance can be monitored.
The prototype of this diploma thesis can be downloaded at
http://www.ifs.tuwien.ac.at/oesterle.

Kurzfassung

WS-BPEL (=Web Service Business Process Execution Language), eine XML-basierte Sprache, dient zur Beschreibung von Geschäftsprozessen, wobei Interaktionen zwischen Prozessen und deren Partnern dargestellt werden können. Die Realisierung dieser Interaktionen erfolgt über Web Services, da somit die Kompatibilität zwischen den einzelnen Applikationen durch Web Standards garantiert werden kann.
Im Prozessmanagement wird vor allem dem Zeitmanagement bzw. zeitlich korrekt ablaufenden Prozessen ein besonders hoher Stellenwert beigemessen.
Die exakte zeitliche Steuerung im Prozessmanagement trägt dafür Sorge, dass die richtige Information zur richtigen Zeit am richtigen Ort verfügbar ist und Deadlines eingehalten werden können. Diese Tatsache ist insofern wichtig, als dass nicht eingehaltene Fristen einerseits dem Image eines Unternehmens großen Schaden zufügen können und andererseits auch hohe Kosten durch den Einsatz von Alternativlösungen entstehen können.
Aufgrund dieser Umstände wurde im Rahmen der vorliegenden Diplomarbeit ein Prototyp entwickelt, der die Modellierung von Zeitaspekten für BPEL-Prozesse ermöglicht. In Design time erfolgt die Überprüfung der Einhaltbarkeit der zugewiesenen zeitlichen Werte einerseits und der gesetzten Deadlines andererseits. Die Modellierung von Zeitaspekten kann sowohl mittels Zeitintervallen erfolgen als auch Zeitdauern und deren Wahrscheinlichkeiten. Letztgenannter Ansatz ermöglicht auch die Definition von Wahrscheinlichkeiten für Verzweigungen. Mit der Prozessinstanziierung werden die Zeitwerte in Kalenderdaten und -zeiten umgewandelt. Anschließend besteht die Möglichkeit die tatsächliche Einhaltung der vorgesehenen zeitlichen Einschränkungen zu überprüfen.
Der Prototyp dieser Diplomarbeit kann unter http://www.ifs.tuwien.ac.at/oesterle heruntergeladen werden.

Acknowledgements

I would like to thank all my friends who supported me during my thesis with patience, knowledge and any other respect. It is a pleasure to thank my parents who always believed in me and made all this possible. I am also grateful to my girlfriend Kathrin for motivating me in any situation and proofreading my thesis again and again. Last but not least, I would like to show my gratitude to my housemates for being patient with me all the time and to my uncle Hubert for his intellectual support.

Christian Österle

Contents

Chapter 1

Introduction

The management of a company has radically changed in the last years. Frederich Winslow Taylor (*1856, †1915) is regarded as the founder of scientific management and predicted: "In the past man was first. In the future the system will be first". His forecast got highly accurate. In the past, the management of a company was defined by leading a company in the economical sense. It was important to make profit; questions like "how to make profit", "to whom is the product directed" or "how to be better than the competitors to satisfy the customer" were negligible.
Today, the idea of management is more complex, based on the different requirements. First of all, companies are more focused on the customer's needs. The economical idea of leading a company is also important, but satisfying the customer's needs and all involved parties that are responsible for a good product is crucial. It is a consequence of the globalization that a product life cycle may depends on other companies which can be far away from each other. Another requirement is to be better and faster than the competition. The increasing competition forces a company to make products free of errors, deliver it fast to the customer and convince the customer of the innovativeness and magnificence of the product. Based on these requirements, a company needs efficient internal structures, which is a challenge of planning-, controlling- and organization systems. At the beginning of the 1990s, business process reengineering and the application of workflow management systems renewed the traditional design of the company management. Consequently, the use of information systems for the process orientated paradigm followed [1].

One approach to make a company more competitive is the planning, controlling and automation of business processes. To make business processes more efficient the idea of workflows and workflow management systems is important. The challenge for workflow

management systems at the moment is the consideration of temporal aspects, in particular with regard to interorganizational workflows. Even thinking about an activity which represents the smallest step in a workflow, it is difficult to forecast its duration because of different aspects. An activity not always consumes the same amount of time. Human errors or failures can cause a delay of an activity as well as unfulfilled required dependencies. Also sickening staff or a loss of machines can lead to a delay of an activity. However, the prediction of an exact activity duration is hard to realize and there is a need to express durations in a useful manner.

As a consequence, the duration of a whole workflow is more difficult to predict than a particular activity, considering that a workflow is a sequence of activities. If an activity of a workflow is delayed, the workflow itself runs the risk of also being delayed.

Interorganizational workflows make the prediction of temporal aspects even more difficult, as they consist of several workflows. Different workflows can have activities in common. From this point of view it is not sufficient to forecast the duration of an activity, a workflow or an interorganizational workflow. If the same activity is used in different workflows, the right temporal position of the activity is important to satisfy the overall deadline of all workflows containing this activity. In other words, for all workflows $W_1...W_i$ with an activity A must be ensured, that A begins and ends at the same date and time, such that the overall deadlines of the workflows $W_1...W_i$ are not violated. Thus, not only the duration of an activity is needed, also its start time and end time is important in order to meet the deadline of all workflows.

Another important temporal aspect in workflows is the modeling of an interval between two not necessary adjacent activities in a workflow. Such intervals can be necessary to express a minimal or maximal temporal distance between two activities.

Beside the temporal problems of interorganizational workflows, there is also still the need of a suitable platform. “Partners of an interorganizational workflow belong to autonomous and organizationally independent and possibly geographically distant entities [2]”. On the one hand, a medium to share information between different companies and workflows is required. On the other hand, a platform independent software has to be used, considering that different companies use different systems. Additionally, processes have to be standardized, that all participants know how to design their processes.

To sum up, a system for the design and monitoring of standardized and interorganizational workflows is necessary, which allows the expression of activity and process durations in a useful manner and facilitates the communication between different companies.

The following example will illustrate this issue.

The starting point for this example is provided by 4 workflows which have activities in common. The 4 workflows are depicted in figure 1.1. Workflow G illustrate in this example the core process. G_1 and G_2 realizes G, i.e. the core process G needs the support of G_1 and G_2 to finish execution. G_1 and G_2 contain an extended subset of the activities of the core process G. In other words G_1 and G_2 partially realize G and share at least one activity with the core process G and also have at least one different activity. Additionally, G_3 realizes G_2.
The arrow between G_1 and G_3 as well as G_1 and G_2 signifies the temporal dependency among each other. Although there is no realization dependency between G_1 and G_3 or G_1 and G_2, they can affect each other temporally by sharing the same activity. To make

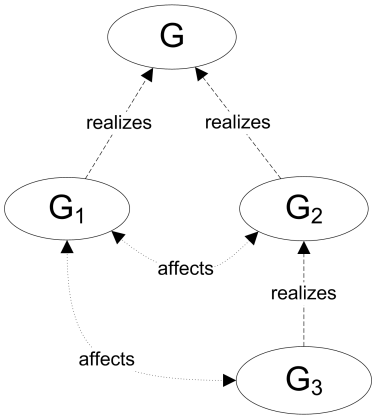

Figure 1.1: Dependencies of workflows

things more clearly, figure 1.2 shows the more detailed processes illustrated in figure 1.1. One can see that all processes (G, G_1, G_2 and G_3) contain the same activity A. G, G_2 and G_3 additionally share the activity B. To plan and calculate the duration of the core process G, all dependent processes have to be considered. Activity A is the first activity of process G and G_2 but the second activity in process G_3 and the last activity in process G_1. For a temporally conformant execution of process G, activity A must have the same start and end time in all related processes. Although activity A could start and finish earlier in process G than in process G_1, A needs a starting and finishing time which satisfy all processes. Otherwise, a deadline violation is likely to occur. If there is an additional need to express an interval between two not necessary adjacent activities in a process, it must be ensured that all other dependent processes consider the effect of this restriction. For instance, process G_3 could have such an interval between the two non-adjacent activity A and B. Let a time interval between A and B be defined with 4 time units. If the restriction causes any changes of starting and finishing times in G_3, all dependent processes have to

consider this change to be temporal conformant.

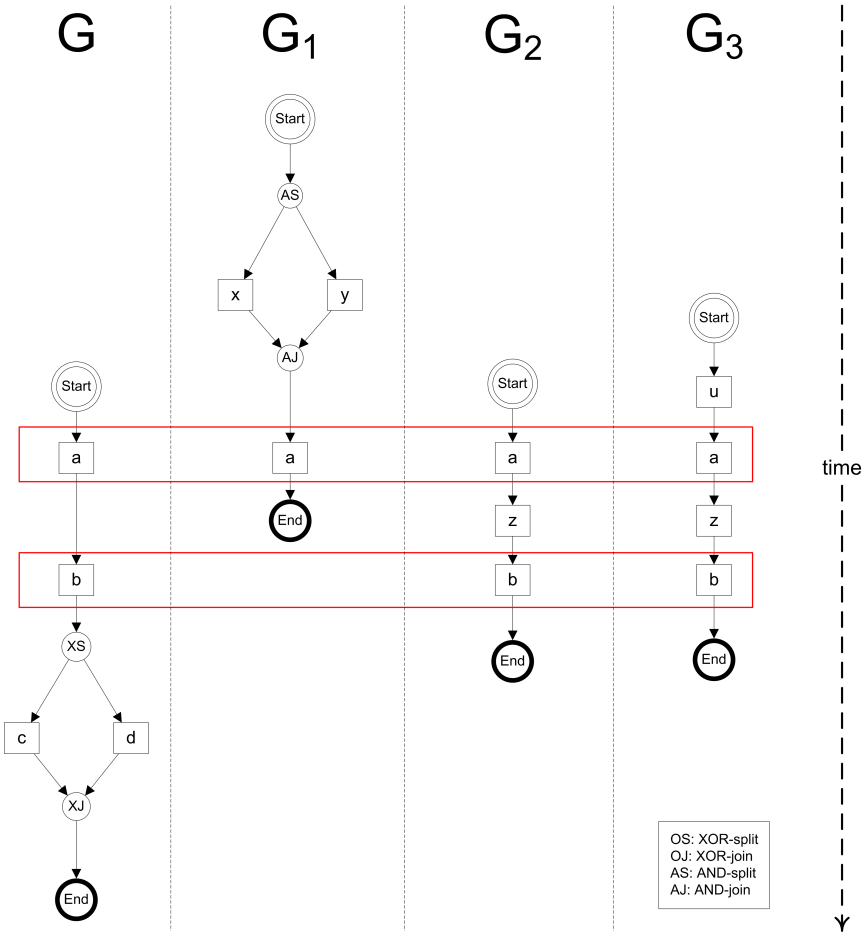

Figure 1.2: Different workflows with activities in common

This thesis provides a solution for the problems discussed previously. A prototype is developed that enables in design time the expression and calculation of temporal aspects of interorganizational workflows, such that all participating processes in an interorganizational workflow are temporal conformant. The theoretical foundation for this prototype was investigated by Tahamtan in [2]. As discussed previously, activity durations are often hard to predict. Tahamtan describes two different ways to simplify the prediction of activity durations. The first one is an interval-based approach, which enables the definition of a maximum duration (upper bound constraint) and a minimum duration (lower bound constraint) for an activity. Upper bound constraints and lower bound constraints can also be used to define an interval between two not necessary adjacent activities. The second approach enables the expression of uncertainty, i.e. every activity has a duration

histogram that contains at least one duration with its belonging probability. This approach also considers the conditional branching behavior. For the calculation of the workflows, Tahamtan provides an algorithm for both approaches, namely *temporalConformanceFederationUbcLbc()* and *temporalConformanceFederation(certainty)*. The former calculates for every activity the earliest possible start (=EPS) and the latest allowed end (=LAE) for both, worst case scenario and the best case scenario. The latter calculates for every activity a time histogram with EPS and LAE values. At process instantiation, the prototype of this thesis checks the precalculated values in design time and monitors the health status of the process.

The workflow language used for this prototype is WS-BPEL (=Web Service Business Process Execution Language). BPEL enables the modeling of executable processes (orchestrations) and abstract processes (choreographies). All functionalities are realized via Web Services, all interactions are performed through Web Service interfaces. BPEL provides a standardized language and uses the Internet as communication medium. This is a prerequisite for interorganizational workflows, considering that participants of interorganizational workflows belong to autonomous and organizationally independent and possibly geographically distant entities.

This thesis is structured as follows:

Chapter 2 gives an overview of related works in time management. The first part describes the mathematical fundamentals of temporal reasoning, the second part discusses different time calculation techniques used in project management.

Chapter 3 presents the concept of Web Services and BPEL. Architecture and functionalities of BPEL are discussed as well as its application area.

Chapter 4 provides an overview of the theoretical basis of time management in BPEL, based on the investigation of Tahamtan in [2].

Chapter 5 describes the prototypical implementation of this thesis. It presents the development environment, used frameworks, data model, most important classes and the validation of the prototype.

Chapter 2

Related works in Time Management

"Management is continually seeking new and better control techniques to cope with the complexities, masses of data, and tight deadlines that are characteristic of highly competitive industries. Managers also want better methods for presenting technical and cost data to customers [3]".

2.1 Temporal reasoning problems

Temporal knowledge and temporal reasoning is important in a wide range of disciplines, e.g. computer science, psychology, philosophy and linguistic. In computer science, it is applicable for information systems, artificial intelligence, program verification and other areas involving process modeling [4]. The form, in which temporal knowledge can be expressed is called temporal representation. Temporal representation should allow significant imprecision and uncertainty of information, because temporal knowledge can not always be presented in precise dates (which may be necessary for computers for temporal computations). "Often, the exact relationship between two times is not known, but some constraints on how they could be related are known [4]".
The area of temporal reasoning problems deals with scheduling and planning of activities and consists of qualitative & quantitative problems. Qualitative problems describe the order of two events. Event A must occur before event B refers to a qualitative problem. Quantitative problems allow the ordering of events with durations. Event A must occur 20 minutes before event B refers to a quantitative problem [5].
Defining durations needs the allocation of time points or time intervals. An example for a time point is *"we will meet us today at exactly 12 noon"*. In contrast to a time point, a time interval could be for instance *"we met us yesterday"*. In this example it is not specified

at which time exactly the meeting occurred yesterday. An interval can be presented by modeling the endpoints of two or more time points. "Assuming a model consisting of a fully ordered set of points of time, an interval is an ordered pair of points with the first point less than the second [4]". Unfortunately, temporal information not always refers to a date system. Temporal relations like *"we met us, while John ate pizza"* do not have durations but the "while" indicates, that the event "we met us" was during the time when John ate pizza. Allen presents in [4] a calculus for temporal reasoning and depicts how to formalize (vague) temporal information. Seven basic relations are proposed for ordering paired objects, namely *before, equal, meets, overlaps, during, starts, finishes* and 13 relations exist by inverting them (except the relation *equal*).
Given the temporal information several requests can be satisfied. Such requests can be for example:

- Does a proposition P holds for a time t_1
- Which possible times hold for a proposition P
- The definition of the possible temporal relationships between two propositions P & Q

With the 13 relationships of Allen, any relationship that can hold between two intervals can be expressed. For instance, the sentence "during dinner, Peter is reading the newspaper" can be formalized in Allen's Interval Algebra [4] as follows:
newspaper {d,s,f} dinner
In this expression, "d" stands for the relation *during*, "s" for *starts* and "f" for *finishes*. Relationships between intervals can be depicted in a network. Nodes represent the individual intervals and the edge indicates a relationship between two intervals. Such a network is called temporal constraint network (=TCN). If a new interval is added to an existing network, all consequences have to be computed by the transitive closure of the temporal relations [4]. For example, if the fact *A is before B* is added and *B is before C*, then it is inferred that A must be before C (and B).

$B \rightarrow C$ (*B is before C*)
$A \rightarrow B$ (Insertion of *A is before B*)
$A \rightarrow B \rightarrow C$ (*A is before B and B is before C*)

The temporal reasoning problems, which are discussed in this chapter are the **S**imple **T**emporal **P**roblem (=STP), the **T**emporal **C**onstraint **S**atisfaction **P**roblem (=TCSP)

and the **D**isjunctive **T**emporal **P**roblem (=DTP). Before the particular problems are discussed, the notion of a constraint has to be clarified. In [6], three types of temporal constraints are presented, the fixed-point constraint, the duration constraint and the interdependent constraint. The last constraint can be seen as a combination of the first two constraints [7]. The fixed-point constraint is given by an absolute time value, like the 15th of March. The duration constraint is a relative value given in time units and refers to another constraint. For instance, activity A should start ten minutes after activity B has been finished, is a duration constraint.

2.1.1 The Simple Temporal Problem

Although the simple temporal problem does not cover a wide area of problems, efficient solving algorithm exist for that problem. A simple temporal problem consists of the following items: Variables depict a time point for an event. A Domain is a set of real numbers (=time instants). Constraints denote an edge between two events and have a weight, that offers the time difference between those two events. A solved STP problem calculates the events by taking into consideration, that all constrains have to be satisfied. The representation of constrains can be done by temporal reference points, which picture some agreed-upon epoch [5]. Figure 2.1 presents on the left hand side a network for a simple

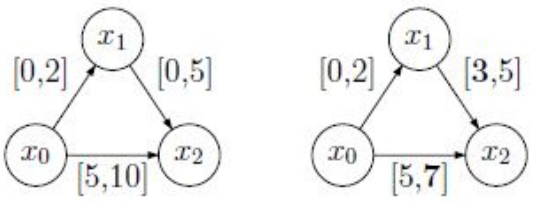

Figure 2.1: A simple temporal problem and its corresponding minimal network[1]

temporal problem. The right hand side provides the solution for the network on the left side, using the method of tightening. Every constraint has been tightened for its maximum, taking into consideration that no solution gets invalid. The result in that case is called minimal network. Note that the schedule of a STP only works, if no cycles with negative total weight exist. Solution techniques for the STP are provided by the following algorithm with given complexity: A minimal network is calculated by using the algorithm of Floyd & Warshall $\mathcal{O}(n^3)$, Johnson $\mathcal{O}(n^2 \log n + m \cdot n)$ and Bliek & Sam-Haroud $\mathcal{O}(n^3)$. The consistency of a STP to check whether a STP does not contain cycles with negative

[1]Image from [5]

total weight can be determined by the following algorithms: Bellmann & Ford $\mathcal{O}(n \cdot m)$ and Dechter et al. $\mathcal{O}(nW \cdot (d)^2)$.

2.1.2 Temporal Constraint Satisfaction Problem & Disjunctive Temporal Problem

The problem scope of the simple temporal problem is not very wide. As soon as several alternative ways for performing an action exist or two events occur in an arbitrary order without overlapping, the STP does not provide any solution. Therefore, the formalism for modeling disjunctions is needed. In other words, the union of several temporal intervals has to be considered. The extension of the STP which allows the disjunction formalism was first described as the Temporal Constraint Satisfaction Problem (=TCSP) in [8] and later on expanded as the Disjunctive Temporal Problem (=DTP) in [9]. Figure 2.2 shows a temporal constraint satisfaction problem. For instance, between the time point x_1 and the time point x_2, two intervals [30,40] and [60,∞] exist that describe the temporal consumption between x_1 to x_2. The former needs between 30 and 40 time points, the latter at least 60 time points. In the TCSP, a constraint between two time points can be expressed as a union of the intervals [5]. The same problem can be transformed from a temporal

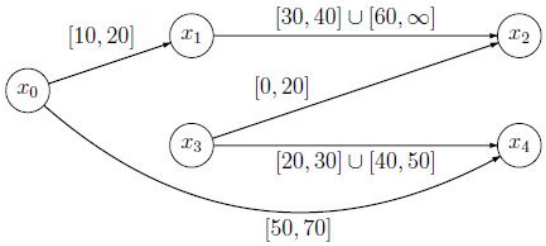

Figure 2.2: A TCSP example[2]

constraint satisfaction problem into a disjunctive temporal problem. "A DTP can be viewed as encoding a collection of alternative STPs [10]". DTP has the advantage, that two disjunctions can have different temporal variables for the same disjunctive constraint [11]. Compared with a TCSP where constraints are binary, every constraint in a DTP is the disjunction of inequalities, where each of them involves two time points. The outcome of a TCSP or a DTP is also the determination of path consistence and the calculation of a minimal network. For solving a TCSP or DTP problem, two categories of algorithm can be used. The former category refers to a constraint satisfaction problem approach where

[2]Image from [5]

all variables are instantiated step by step until a solution has been found or a dead end is detected. The latter category deals with the abstraction of inequalities using the TSAT algorithm. One of the most efficient algorithms at the moment for solving a TCSP or DTP problem is called TSAT++ [5].

2.2 Time calculation techniques in Project Management

Time calculations in workflows refer to the area of network planning. In project management and workflow management, graph-based representation methods of timed workflows offer many advantages. They can be visualized immediately, restrictions and errors can be identified very fast and changes can be performed easily. This section provides models and solutions for calculating network plans. Such techniques are necessary for planning time, costs and capacities. The purpose of time management in workflows is the determination of the shortest overall time i.e. the execution of all activities in a workflow in a minimal time, the finding of the earliest & latest possible begin & end of activities, the detection of buffer times and critical paths. A critical path is the longest sequence of activities which has to be completed on time in terms of meeting the workflow deadline. For calculating time constraints in workflows four common used network scheduling techniques in project management can be used, namely the CPM (=**C**ritical **P**ath **M**ethod), MPM (=**M**etra **P**otential **M**ethod) and PERT (=**P**rogram **E**valuation, **R**eview **T**echnique) or the GERT (= **G**raphical **E**valuation and **R**eview **T**echnique).

2.2.1 Critical Path Method

The critical path method is a deterministic model for calculating the critical path in a workflow and works with the end - start relation. A prerequisite for calculating the critical path in a network plan is an acyclic digraph, which means a directed graph without any loops. The end time of an activity signals the start time of the following activity. Nodes are numbered and an arrow between two nodes represents the activity duration as well as the relation of predecessor and successor activities (see figure 2.3). The determination of the critical path consists of three steps. First step is a forward pass to calculate the earliest occurrence times of activities, i.e. the start time of an activity is calculated by the start time plus the duration of its predecessor. If an activity has more than one predecessor, the predecessor with the minimal value (start time + duration) is selected. In the second step,

the latest occurrence times are calculated by a backward pass of each activity. Finally, the critical path can be identified by selecting the activities, where the earliest occurrence time equals the latest occurrence time (= critical activities). Graphically, the critical path can be determined by spanning the entire network plan from start to finish, passing only the critical activities (see figure 2.3). The total duration is calculated by adding all durations of the critical activities in the critical path. Note, that a network plan can have more than one critical path [12]. "Dijkstra's algorithm computes the shortest paths from a source vertex to every other vertex in a graph, the so-called single source shortest path (SSSP) problem. Dijkstra's algorithm is frequently used because of its time bound $\mathcal{O}(m+n\log n)$, implemented by using a binary heap, remains the best for computing single source shortest path problem with non-negative weighted graphs" [12].

2.2.2 Metra Potential Method

MPM is a network planning method of the type activity on node. The difference between the notion of a CPM and a MPM is illustrated in figure 2.3. Although a huge amount of different notions of CPM and MPM exists, this work references on the differences between CPM and MPM illustrated in [13]. In MPM, every activity with its name is illustrated as a node and the directed edges between two activities represent the weight (duration). In a CPM network, all nodes get numbers with its start and finishing times and the edges between two nodes illustrate the name of the activity with its duration. The metra potential method enables the definition of the earliest possible & latest allowed start by using the forward termination, as well as the earliest possible & latest allowed end of an activity by using the backward termination [14]. Every arrow of a node has to refer to another

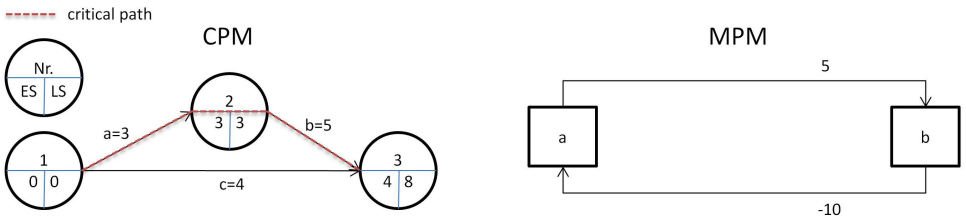

Figure 2.3: CPM vs. MPM

node. The main advantage of the MPM method is the ability to coordinate activities [15]. Therefore, activities can have positive and negative arrows. A positive arrow with the value 5 between an activity A and an activity B means, that B can start 5 time units after A has started. A negative arrow with the value 10 between an activity A and an activity

B defines, that B has to start within 10 time units after A has started. The possibility of positive and negative arrows between two activities effects, that a network plan can contain cycles.
For the calculation of a network plan with the MPM method, two algorithms are presented. The first one is called "Floyd's and Warshall's algorithm" and calculates all pairs shortest paths (=APSP) on a graph with the complexity $\mathcal{O}(n^3)$.
"The algorithm runs a loop of n iterations, for $1 \leq \mathrm{k} \leq n$. In each iteration, the algorithm computes for each pair (i,j) (including $i = j$) the shortest distance from node x_i via x_k to x_j and updates the weight $w_{i \to j}$ if the new value is less than the original value. After n iterations, all $w_{i \to j}$ have been set to their minimal values. The initial value of all $w_{i \to j}$ (the weight of the virtual edge from a node to itself) is taken to be zero; if it is ever to be changed to a negative value, a negative cycle has been detected and inconsistency can be concluded" [5].
The second presented algorithm for calculating the MPM method is called "Bellman's and Ford's algorithm" and works similar to Dijkstra's algorithm. The outcome of this algorithm is the shortest path to all vertices in a distance matrix with the complexity $\mathcal{O}(n \cdot m)$. Unlike Dikstra's algorithm, this algorithm can deal with negative edge weights.
"If the graph contains a negative cycle, the algorithm will detect this in the final for loop. The reason for this is that in a negative cycle, the distance matrix will keep being updated and is never finished" [5].

2.2.3 GANTT

The GANTT chart was developed by the management consultant Henry L. Gantt and is a horizontal bar chart to illustrate the temporal sequence of activities. The duration of activities in a GANTT chart are better depicted than in a network plan, whereas the representation of dependencies between activities are better illustrated in a network plan. The structure of a GANTT chart is illustrated as a matrix which is composed of a vertical axis that lists all activities and a horizontal axis to indicate the duration of an activity. Furthermore, fields like skill level to perform an activity and the name of the person who is responsible for the activity can be also depicted.

2.2.4 Program Evaluation and Review Technique

PERT works quite similar to CPM. The main difference is the handling of the activity times. CPM assumes, that activity times are deterministic. PERT allows the definition of

stochastic activity times [16]. Whereas the activity can have stochastic time values, the structure of the network itself has to be deterministic. A PERT chart can realize parallel executions of tasks, but does not offer a conditional branching behavior. Slack times are usually depicted by a dotted line between the end of the predecessor task and the start of the successor task. The PERT chart is completed if and only if all tasks come together at the end node.

The concept of PERT is to combine the huge number of uncertain durations to three estimated time values for each activity, an optimistic (=DO), a pessimistic (=DP) and a most likely duration (=DM). The aim is to obtain a quantitative prediction based on the probability calculus. Based on the values DO, DP and DM an expected duration (=DE) is determined by a probability density function. Chronometries revealed, that the frequency of the obtained expected durations (DE) approximates to a beta distribution [17]. The formula for the calculation of DE can be expressed as follows:

$$DE = \frac{DO+4DM+DP}{6}$$

A second formula presented in [13] does not include the most likely duration DM because of its uncertainty. DM is the statistical dispersion or the mean square error. Without taking the DM value into account the formula is expressed as follows:

$$DE = (\frac{DP-DO}{6})^2$$

The usage of PERT makes good economic sense, when certain activities have performed for a large time under essentially the same conditions, such that the duration of activities can be interpreted statistically. The outcome of PERT is also the critical path of a network plan. Its requirements are that all activities have to be finished before the project can be finished. Additionally, PERT assumes that all activities with a successor need to be completed before the successor can be performed [18].

The usage of a stochastic model for the calculation of a network plan facilitate a more realistic scenario but also causes consequences. The calculated expected duration (DE) based on a beta distribution do not conform to the most likely duration (DM). Sophisticated worker assigned to an activity or a whole process specify the most likely duration based on their (past) experiences. One disadvantage is, that they often calculate with the most likely duration (DM), even if the expected duration (DE) is already in use for the process computation. Another drawback is the assignment of the optimistic (DO) and pessimistic duration (DP). Past experiences revealed, that the distance between DP and DM is often bigger than the distance between DO and DM [13]. As a consequence, DM tends to be smaller than the DE. As far as the expected duration DE is the standard duration of an

activity and $DM < DE$, the addressed people to that activity tend to consume more time for the activity than they actually would need. On the other side, a very important aspect in process designing is the consideration of buffer times. Even if the employees meet exactly the time scales it is also possible that a third involved party does not fulfill some requirements and the employees can't perform the next activities [13].

GERT provides an extension to the PERT approach. GERT takes the probabilistic branching into account, which allows the modeling of a stochastic network structure. It also allows the looping of activities and enables the possibility of multiple outcomes of a workflow [19]. The branching behavior can be realized by three logical operators (XOR, OR, AND). Nodes of a GERT network are numbered, directed edges between two nodes are indicated with the activity, the possibility and the duration.

Chapter 3

WS-BPEL

This chapter provides an overview of the **W**eb **S**ervices **B**usiness **P**rocess **E**xecution **L**anguage (=WS-BPEL). The first section deals with Web Services in general. The second section describes the development of WS-BPEL. In section three, the BPEL language is presented and the last section describes the structure of a business process in BPEL.

3.1 Web Services

To understand the meaning of a Web Service, the idea of a service itself has to be discussed. A service is a self describing open component with defined interfaces, that enables a fast and economic composition of distributed applications. It follows a wide-spread standard and allows re-usability. Services are offered by a service provider, which is responsible for the implementation and the technical support of the service. A "service consumer" calls a service, using the specified name of the service interface. Every service needs a name, a service interface for the access and a service contract with the information about the responsibility, functionality and the restrictions of the service. The service implementation is the technical realization of the service [20, 21]. The **S**ervice **O**rientated **A**rchitecture (=SOA) is an architecture model which contains the services in consideration of certain design rules [22].
One of the most important concepts of SOA in contrast to other standardized architectures are Web Services and the modeling of executable processes [20]. They transform the basic mechanism of the web (sharing information) into more complex, functional systems. Web Services enable the public access of electronic services, as we know from using the Internet, for getting textual information [20].
The W3C defines a "Web Service" as "a software system designed to support interop-

erable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically Web Services Description Language WSDL). Other systems interact with the Web Service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards [23]". A web Service works in a similar manner to a web site. A client sends a HTTP request to a Web Service that contains a SOAP message using the URL, the name and the parameters of the Web Service. The Web Service receives the request over a listener and passes it to the business interface, which converts the SOAP message into a Web Service readable format. For the processing of the information, the message gets forwarded to the business logic. The result of the data processing is converted back to a SOAP message and is returned to the client [24].
The basic technology behind Web Services is limited to XML for the description of the complex interactions and HTTP for the exchange of the messages. A more complex technology for Web Services is provided by Dustdar in [25]. Dustdar presents a layer model of a Web Service architecture and differs between core layers and higher layers. The core layers include XML, SOAP and HTTP for the transmission of data. The higher layers include BPEL (=**B**usiness **P**rocess **E**xecution **L**anguage) and WSDL (=**W**eb **S**ervices **D**escription **L**anguage).
A Web Service works as follows: The service provider (=supplier) offers the Web Service and registers it at a Service Broker. The standardized Service Broker is called UDDI (=**U**niversal **D**escription, **D**iscovery and **I**ntegration). The Service Broker lists all Web Services in a directory. The service requester (=customer) searches for a service in the directory of the service broker and is able to call it.
SOAP is a protocol for the exchange of XML data over the web. It is the common language of all involved parties (service requester and provider) and contains information about security, routing and additional information. SOAP assumes a communication over a WAN. WSDL describes the interfaces and the functionality of a Web Service and how it can be accessed. A WSDL document contains the following fields: A type, where the data types are defined. The message contains the abstract definition of the exchanged data. An operation describes the supported functionality of a service. A Port Type defines a set of operations, which are supported by one or more network endpoints. The binding specifies the used protocol and the supported operations of a port type. A port is a network endpoint and specifies how a service can be accessed. The service is a collection of related endpoints. A WSDL message can be of the type one-way, request/response, solicit response and notification [20].

3.2 The development of WS-BPEL

In the past, the realization of workflows was often managed manually. BPEL enables the graphical presentation of a workflow and automatically produces the executable code.
WS-BPEL is a specification of Microsoft, IBM, Siebel Systems, BEA and SAP. The BPEL4WS 1.1 specification was presented in 2004 to OASIS (= **O**rganization for the **A**dvancement of **S**tructured **I**nformation **S**tandards is a consortium that drives the development, convergence and adoption of open standards such as Web Services [26]), and after three years of work, the OASIS standard WS-BPEL 2.0 was released on April 12, 2007.
WS-BPEL is a successor of WSFL (IBM) and XLANG (Microsoft) [27][28]. WSFL describes the sequence of business processes within a Web Service. XLANG describes the message exchange between Web Services. WS-BPEL is the combination of WSFL and XLANG [2]. The interface of a Web Service in BPEL is defined in the Web Service description language (=WSDL).
OASIS summarizes some main objectives for the development of BPEL in [29]. The definition of XML based business processes have the ability to interact with external entities through Web Service operations. Web Service orchestrations concepts can be used in common by both the external (abstract) and internal (executable) views of a business process. BPEL should meet the requirements of providing both hierarchical and graph-like control systems, and allow their use to be combined as perfectly as possible. Hierarchical blocks can be nested and are represented in BPEL as structured activities like for instance “If”. BPEL offers functions for simple data manipulations. It provides an identification mechanism at the application message level for process instances and offers the ability for the implicit creation and termination of process instances. BPEL is build on compatible Web Service standards, defines a long running transaction model and uses Web Services as the model for process decomposition and assembly [29].
BPEL is capable to model stateful processes by modeling the behavior and interactions among involved partners. “For example, a process that receives a message, transforms it, sends it to a business partner, and then waits for an asynchronous response is stateful” [30]. All functionalities are realized via Web Services, all interactions are performed through Web Service interfaces.
WS-BPEL enables the modeling of two kinds of business processes:

1. Executable processes (orchestrations): “Refers to an executable business process that may interact with both internal and external Web Services. Orchestration describes how Web Services can interact at the message level, including the business logic and execution order of the interactions. These interactions may span applications and/or

organizations, and result in a long-lived, transactional process. With orchestration, the process is always controlled from the perspective of one of the business parties." [31]

2. Abstract processes (choreographies): "More collaborative in nature, where each party involved in the process describes the part they play in the interaction. Choreography tracks the sequence of messages that may involve multiple parties and multiple sources. It is associated with the public message exchanges that occur between multiple Web Services." [31]

Data associated to an executable process is referred to as "opaque data", to an abstract process "transparent data". There is an implicit relationship between executable processes and abstract processes. "The executed paths and decisions made at choice nodes may depend on exchanged messages in the abstract process [2]."
WS-BPEL extends the Web Services interaction model and makes it possible to support business transactions. The definition of the interoperable integration model of WS-BPEL had the aim to smooth the progress of the expansion of automated process integration [32]. One advantage is that there is no need to reveal the service or the provider's internal process logic. The business know-how is protected but nevertheless it's possible to communicate with other partners. Another advantage is the fact that "as long as an abstract process remains the same, the executable process can be changed and modified with no external effect [2]."

3.3 The BPEL Language

BPEL is a workflow definition language based on XML and allows the description of business processes within and between companies, which are connected via Web Services. BPEL can integrate Web Services into a steady business solution and facilitates the orchestrated interaction by doing so.
"A Business Process using BPEL can compose multiple Web Services, effectively creating a completely new business application with its own public interface to end users (internal or external). BPEL opens a completely new way, or at least enhanced way, for software development for mainstream business applications to allow a programmer to describe a business process that will take place across the Internet [33]." BPEL is a language for the description of the logic to coordinate and control Web Services during a process flow. It builds on XML and Web Services specifications and also extends them [33]. It is an

orchestration language and not a choreography language [34]. The difference is that an orchestration involves message exchanges within different systems by controlling parts of this exchange by the orchestration designer, while choreography is a protocol for peer-to-peer interaction with the aim to guarantee interoperability. However, the protocol provides the possibility for defining choreographies as well and there are different approaches how to adopt BPEL to a choreography language. One approach, presented in [35], is the mapping of choreography parts to abstract BPEL process models.
One of the participating business parties controls the BPEL orchestration engine. The BPEL Engine executes all activities in a process flow, which are compatible to the BPEL standard. It invokes Web Services, maps data content, handles errors, enables transactions and provides security mechanism [36]. BPEL "is comparable to general purpose programming language such as Java, but it is not as powerful as Java. One can say that it is simpler and better suited for business process definition. Therefore BPEL is not a replacement but rather a supplement to modern languages such as Java [37]."

3.4 Definitions of Business Processes in WS-BPEL

The structure of a business process in BPEL consists of *fault handlers*, *compensation handlers*, *event handlers*, *partner links*, *message exchange*, *variables*, *correlation sets* and *activities*. During the execution of a BPEL process faults may occur. Therefore BPEL provides fault handling mechanism that work similar to the one of Java. A catch block within the XML structure allows the fault handling. *Fault handlers* can be referred to activities and specifie the behavior in case of a fault. For instance, a *compensation handler* can be called that allows the functionality of a rollback and thus enables the undoing of a completed faulty process in the reverse order of the normal process execution.
Event handlers can be used to start an activity when certain events or alarms occur.
In WS-BPEL, a *partner* participates in a Web Service transaction and communicates with other partners over defined interfaces. A *partner* can either call another process by a service endpoint reference or can be called by another process. A *partner* is connected to a *partner link*, which models interactions with other services. Partner links describe the range of functions, indicate the role of the process and include information about communication data. Every *partner link* is referenced to a *partner link type*. Partner link types are specified in the Web Service description language file and describe the relationship between two services and their roles.
For saving the information of a message exchange in BPEL, the usage of variables is nec-

essary. BPEL provides the data types "WSDL message types", "XML scheme types" or "XML scheme elements". A variable may have an initializer which assigns a fixed value to it. A validator can proof the conformance of a variable in terms of the XML or WSDL definition.
Correlations are important, when a message is used for several conversations. For example, an invoice number can be important for more than one conversation between partners. If more than one *correlation* exists for a specific purpose, they can be collected to a *correlation set*. Local and global correlation sets exist. Local ones lose visibility outside the associated scope and global ones are valid for the entire business process. A *correlation set* has a name and an ID for the instance of a process.
For creating a BPEL project, the following files are needed: A BPEL file for the description of the process, a WSDL file that offers the interface how to communicate with the Web Service and an optional XML scheme file for the definition of data types.
A BPEL process is initiated after the receipt of a message by a receive or pick activity with the attribute *createInstance = "yes"*. Its termination can happen through the execution of all activities, a modeled exit activity or a fault. A BPEL process can be whether synchronous or asynchronous. The former contains either a request/response activity or an invoke activity with input and output variables and is a blocking process. The latter does not wait for a respond before proceeding further, but can optionally inform the sender by a callback.
Activities are the basic units of work in the course of a business process and they include control flows, invocations of other Web Services, interaction and messaging. An activity can be a task, a complex activity or a (sub-) workflow. BPEL does not provide sophisticated activities for data manipulations or alphanumerical operations, because they assume that data manipulations and computations are basically carried out externally by Web Services [2]. BPEL provides the following activities:

- Invoke: An invoke provides the possibility of invoking exposed services of other Web Services.
- Receive: A receive activity waits until a message of a partner arrives and might be a creator of a new process instance. It is a blocking activity, i.e. the process halts till the arrival of a message.
- Assign: It assigns values to variables with either a XPATH expression or a literal.
- Reply: A reply is a response which can be always sent after a receive activity or a pick activity. It is only used for synchronous interactions. Receive and reply must

have the same partner link, port type, operation and if necessary the same correlation set.

- Throw: A throw is used when faults are expected.
- Wait: The activity wait can be used for time management purposes when delays need to be inserted in a business process. A delay can be specified by giving a duration or a time point.
- Empty: It is a basic activity, does nothing and has no effect. It is useful for debugging and avoids faults when the execution of activities are expected.
- Sequence: It defines in which order a group of activities will be performed. A chain of activities is executed one after another.
- If: An if activity can model the conditional behavior (XOR-split). Every branching needs a condition, specified for example in XPATH.
- While: It is used to model loops and repeat child elements until the condition is false. The condition is at the beginning of the loop.
- RepeatUntil: Child elements of a repeatUntil activity are executed until the condition is true. The condition is at the end of the loop, i.e. the child activities are executed at least once.
- ForEach: Provides the iteration over activities and enables parallel execution of activities.
- Pick: A Pick might instantiate a process and is used for event handlers. It waits for a message, a timeout or an alarm and handles the first arriving event and discards the subsequent events.
- Scope: It "provides the context which influences the execution behavior of its enclosed activities [32]."
- Flow: A flow models parallelism and concurrency of activities. A flow activity finishes after all child elements are finished.
- Exit: It enables the prematurely termination of a process.

For every activity a name can be specified which makes a process more understandable. Additionally, every activity can have the child element *link*. A *link* consists of a source and a target link and thus connects a source activity with a target activity. They are needed for ordering activities. Note that links must not form a cyclic graph. The definition of links can be important for example in a flow activity. As long as no links exist, all activities in a flow are executed parallel. If an activity A has one or more incoming links, a *join condition* is needed. A can only start, if the *join condition* is "true", i.e. the states of all incoming links are set to "true". *SupressJoinFailure* allows a fault mechanism, which is enabled when the *join condition* is set to "false". In that case and if the *supressJoinFailure* is set, it allows the skipping of the activity associated with the false *join condition*.

Chapter 4

Time Management in BPEL

This chapter presents techniques for checking the temporal conformance of federated choreographies. Temporal conformance of federated choreographies in BPEL ensures, that "activities are performed in a timely manner and the right information is delivered to the right activity at the right time such that the overall temporal restrictions are satisfied" [2]. This is an important quality criteria for reducing costs and gaining the highest level of efficiency in federated choreographies. The earlier a time violation can be detected, the less costs are necessary to handle the violation.
This chapter is organized as follows: The first section briefly discusses related works of time management in BPEL. The second section describes the investigation of time management in BPEL by Tahamtan in [2]. His approach consists of two parts, the calculation of temporal constraints in design time and their monitoring in run time. The used algorithms for those calculations, based on [2] are also proposed in this section.

4.1 Related works

In the past, the modeling of temporal features for workflow systems was not a widely investigated area [38]. "Commercial workflow systems (as reviewed, e.g., in [Alonso et al., 1997] are usually limited to the specification of a deadline for each activity or global plan. In some cases more elaborated temporal conditions can be specified, but no reasoning other than run-time evaluation on these conditions is supported [39]". Before different approaches of temporal constraints in workflow systems are presented, a briefly description about workflow models is given.
Van der Aalst proposes in [40] an approach how to map workflow management concepts to Petri Nets. Activities are modeled as transitions, conditions are modeled as places

and cases are modeled as tokens. Directed edges describe dependencies between activities. Conditional branching behavior can be realized by using a place with multiple outgoing edges (OR-split) and a place with multiple incoming edges (OR-join). Parallel processing of activities can be represented by a transition with multiple outgoing edges (AND-split) and a transition with multiple incoming edges (AND-join).
Another concept of a workflow model is called Precedence Graphs. A Precedence Graph is a directed acyclic graph. Activities are represented as nodes and the order of the activities is realized with edges. A directed edge from activity node A to node B means that activity A has to be executed before B can start. If an activity A has more than one incoming edge, all predecessor activities have to be finished that A can start. Precedence Graphs also allow conditional branching behavior and parallel execution of activities. For that purpose, the graph has to be augmented with control nodes that allow split and join structures as well as the parallel processing of activities [39].
In [41], Wodtke and Weikum present a solution how a workflow can be represented as a State Chart. A State chart is principally a finite state machine and consists of a initial state and transitions driven by ECA (=Event Condition Action) rules. A State reflects an activity in a workflow. A Transition represents a relationship between two states (activities) and is annotated with an ECA triple. "A transition from state X to state Y fires if the specified event E occurs and the specified condition C holds. The effect is that state X is left, state Y is entered, and the specified action A is executed [41]". Conditions can be modeled as data item variables.
Another modeling concept how to specify a workflow is using a script language. Eder presents in [42] a workflow definition language (=WDL) which allows the description of a workflow in a textual manner. The WDL language consists of five basic units: the workflow specification part, activities, roles, organization structures and inter-process communication. The structure of a workflow specification reminds of the structure of procedural programs and consists of a header, a declaration part, a body and a set of rule-based languages.

With the fundamentals of different workflow models, several time management approaches based on those models can be discussed.
In [38], an approach is presented which offers a formalism to specify quantitative temporal constraints for an activity and a process as well as a reasoning tool for the workflow management and the workflow enactment service. Regarding the granularity of the temporal values, the special type Temporal Constraint with Granularity (=TCG) is presented which allows the allocation of time points like for example in days, hours, minutes etc.

Algorithms for the consistency, prediction and enactment services are provided. The consistency service ensures, that temporal constraints of a process are possible to satisfy. The prediction service calculates start and end times for an activity or a process, while the enactment service is responsible for the monitoring of instantiated activities. If an activity finishes after its predicted end time, an exception occurs.

Kao and Garcia-Molina present in [43, 44] a strategy, how to deal with tasks in a distributed environment. A distributed task usually consists of several subtasks which can be executed at different locations in a given order. Those subtasks can be executed in a sequential, parallel or serial order. Real time systems often need an overall deadline that indicates the end of a distributed task. Considering that a distributed task consists of several subtasks which also have assigned deadlines, a strategy is presented how to deal with those subtask deadlines.

Adam uses in [45] a block structured process description language for modeling a workflow based on ADEPT. Different types of nodes are used to facilitate for example the modeling of branchings and loops. The possibility, that nodes can be nested reminds of the functionality of BPEL. Dynamic changes like insertions, deletions and shifts (changing the sequence of steps) are supported. Minimal and maximal durations can be specified for each activity in a workflow as well as time dependencies between two not necessary adjacent activities in a workflow. In the prototype of this thesis, such dependencies between two activities are modeled as lower bound and upper bound constraints. At build time, the temporal feasibility of the workflow is checked such that all constraints are satisfied and no deadlines are missed. At run time, these values are compared with the actually durations. If a deadline is likely to be missed, the user gets informed.

In [46, 47] relative and absolute time values are used to model durations. Relative time (three hours, 20 minutes) is used at build time, absolute time (31.12.2009::15:30) at run time. For every activity node, a minimum and maximum duration can be assigned. Additionally, a relative deadline “defines when a task should start/finish relative to the start/finish of another task [46]“. This concept works similar to the approach with lower bound and upper bound constraints which is used in this thesis. Beside the relative deadline, an absolute deadline defines the start or end of an activity during the execution of a workflow.

In [48], common project management tools are used for the time management of a work-

flow. The main problem is, that several concepts of a workflow can not be mapped into project management concepts like the conditional branching, a loop or a recursion.

Pozewaunig in [49] describes a concept for time management in workflow systems using an extension of the network diagram technique PERT. This extension covers the computation of internal activity deadlines and considers the execution of sequential, alternative and concurrent activities. Activities can be assigned with 3 values, namely minimum duration, maximum duration and average duration. Using the β-distribution, execution times of activities as well as the shortest and longest process execution time can be calculated.

An approach for modeling and analyzing time constraints in BPEL is presented in [50] and [51]. This approach uses the transformation of a BPEL process to a Web Service Timed Transition System (=WSTTS), which extends the formalism of a State Transition System (=STS). Web Services Timed Transition Systems allow the modeling of temporal constraints in a BPEL composition and work similar to the formalism of timed automata. The time consumption of an operation is modeled by a time increment in the state. To check whether the transition takes place at the right moment, states and transitions are defined as special clock variables. Using simple modeling constructs and complex Duration Calculus formulas presented in [52] and [53], various time related constructs can be expressed. Model checking techniques enable the verification and computation of timed Web Service compositions. Deadlines can be defined, best case and worst case scenarios can be calculated at build time and the calculated process can be monitored at run time [2].

Another approach, presented in [54] "enables a declarative, separate, and verifiable specification of temporal properties [54]", using the formal language namely XTUS-Automata. A XTUS-Automata combines the functionality of a timed automata (=TA) with the extended time unit system (=XTUS). This approach provides a solution for the specification of temporal constraints with relative and absolute time as well as their monitoring at run time, using the aspect oriented workflow language AO4BPEL based on XML. AO4BPEL is presented in [55] and raises the limitations of the static Web Service model. Temporal constraints can be translated into modular aspect code that listens to activities during the execution of a process. An activity will be only executed, if the temporal constraints are satisfied.

Guermouche, Perrin and Ringeissen in [56] also represent Web Services as an automata using the Roman model [57, 58]. That model differs between internal and external con-

straints. Internal constraints specify a relative time period (= local clock) and an absolute time period (= global clock) and they can be used for the expression of activation and dependency conditions. External constraints are exposed by the client and the provider service and they have to be checked before the initialization of the interaction. They infer from internal constraints and allow the detection of incompatibilities of services.

In [2], different concepts of related works come together and a new approach of calculating time constraints in federated choreographies is presented.
The approach, presented by [2] provides a solution for the calculation of time constraints at design time as well as a solution to check these values at run time. Time constraints are divided into implicit and explicit constraints.
Implicit constraints are execution durations for activities and can be designed at the specification level by experienced specialists or through workflow logs from past executions. They can also be derived implicitly from control dependencies between the start and end time of activities. For example, an activity B can only start if and only if all of its predecessors have finished. Such constraints are also called *structural time constraints*.
Explicit time constraints "are either temporal relations between events or bindings of events to certain sets of calendar dates [59]". They can be based on organizational rules and business policies, laws and regulations [59].

4.2 Calculation in Design time

For calculating federated choreographies in design time, two approaches are presented. The first is called calculation of time constraints and needs a duration for each activity and a deadline for each choreography or orchestration. The second approach is called probabilistic time management and calculates the temporal values based on time histograms with a duration and its belonging probability.

4.2.1 Calculation of time constraints

For every basic activity, a time duration can be defined (see figure 4.1). Apart from the durations of an activity, lbc (= **l**ower **b**ound **c**onstraints) and ubc (=**u**pper **b**ound **c**onstraints) can be defined. A lower bound constraint defines the minimum duration between two events A and B. An upper bound constraint defines the maximum duration between two events A and B. Lbc and ubc can be used for defining the duration of one activity and for setting the duration between two activities. Assuming that one activity A

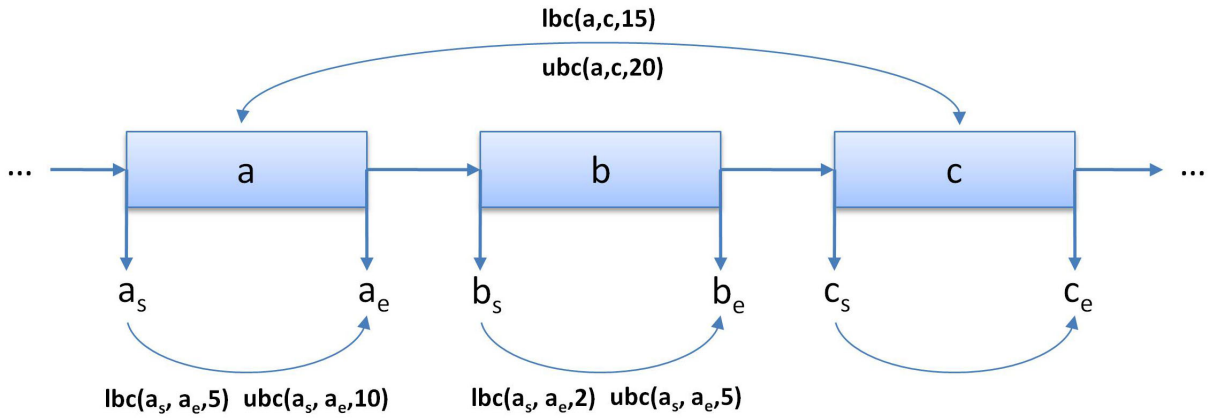

Figure 4.1: Durations of Activities

has a lbc of 5 and an ubc of 10 time points, the minimum duration of the activity would be 5 time points and the maximum duration of the activity 10 time points (see figure 4.1).

Lbc(a,c,15) and upc(a,c,20) between a source activity A and a target activity C means that the duration between activity A and C must have at least 10 time points and must not have more than 15 time points (see figure 4.1). Note that the source and target activity are not necessarily adjacent.
Apart from defining durations for each activity of a BPEL process, a process itself needs a maximum duration, in which the whole process has to be finished.
After defining the maximum duration of a process, the durations of its containing activities and the durations between activities, the whole process can be calculated by setting the following values to each activity (see table 4.1):

- BC EPS (=**B**est **C**ase **E**arliest **P**ossible **S**tart): Defines the earliest point in time for the best case scenario, in which an activity can start execution.
- WC EPS (=**W**orst **C**ase **E**arliest **P**ossible **S**tart): Defines the earliest point in time for the worst case scenario, in which an activity can start execution.
- BC LAE (=**B**est **C**ase **L**atest **A**llowed **E**nd): Defines the latest point in time for the best case scenario, in which an activity can start execution.
- WC LAE (=**W**orst **C**ase **L**atest **A**llowed **E**nd): Defines the latest point in time for the worst case scenario, in which an activity can start execution.

Activity Name	Activity Duration
BC EPS	WC EPS
BC LAE	WC LAE

Table 4.1: Example of a timed activity

Algorithm

For the calculation of the time constraints, the algorithms in [2, S. 157] provides the basis. Due to some calculation errors of those algorithms, a few modifications had to be made. Therefore, the complete algorithms for calculating time constraints in [2] are depicted in this work with its modifications.
The main algorithm is called temporalConformanceFederationUbcLbc() and consists of the following subalgorithms:

- Initialize(G): All EPS and LAE values of activities of a process G are set to 0 and ∞.
- Calculate(G, G.deadline): It is necessary to calculate for each activity A of a process G the EPS and LAE values for best case and worst case scenario. The EPS values are calculated in a forward pass by considering existing EPS values, the newly calculated EPS values and possible lower bound constraints. For activities after "FLOW" or "IF" controls, worst case values can differ from best case values, as there are different branches for the calculation. The LAE values are calculated in a backward pass in consideration of three values: the existing LAE value, the newly calculated LAE value, and the possible lower bound constraint.
- incorporateUbc($G, G.deadline$): This method applies the upper bound constraints into the process G. Before applying the upper bound constraints, it is checked, if the time values of the source activity plus the time value of the ubc are smaller than the time values of the target activity. If so, the EPS values of the source activity are set with the EPS values of the target activity minus the time value of the upper bound constraints. Analog to the EPS values, the LAE values of the source activity are set with the LAE values of the target activity plus the time value of the upper bound constraints. Afterwards, the process is recalculated by the method calculate(G, G.deadline) and the conformance is checked by the method checkConformance(G). If the time values of the target activity have changed, a violation occurs and the method incorporateUbc($G, G.deadline$) stops.

- checkConformance(G): This method checks for all activities in the process G, if the sum of EPS and duration of an activity is less than or equal lae. If this condition is not fulfilled, the method return the value false.

- propagate(G,H): For all activities owned of the process G AND the process H, the following changes are made: If the EPS values of an activity in H are smaller than the EPS values of the same activity in G, the EPS values of the activity in H are set to the EPS values of the activity in G. If the LAE values of an activity in H are bigger than the LAE values of the same activity in G, the LAE values of the activity in H are set to the LAE values of the activity in G. If any changes were made, the method returns the value true, otherwise false.

The algorithm temporalConformanceFederationUbcLbc() consists of two steps. The first step is the initialization and precalculation. The global choreography C_g is initialized and calculated. Upper bound constraints for C_g are incorporated and the conformance is checked. Then, all directly and indirectly supporting choreographies and realizing orchestrations G of C_g are considered. For each directly and indirectly supporting choreography or orchestration G, the following is applied: After the initialization of G, the global choreography C_g is propagated to G, if both have activities in common with EPS (LAE) values of C_g bigger (smaller) than EPS (LAE) values of G. If so, the EPS (LAE) values of G are applied with the EPS (LAE) values of C_g and the deadline of G is newly determined because of its changing values. Furthermore, G is calculated again with its changes. Next, G is propagated back to C_g. If a change occurs, the global choreography C_g is calculated again and its conformance is checked. All incoming and outgoing edges of C_g are marked.

In the second step, dependencies between **all** supported choreographies and supporting choreographies or realizing orchestrations are considered. Note that in the first step only dependencies between the global choreography and all directly and indirectly supporting choreographies and realizing orchestrations are considered. Like in the first step, supported choreographies are propagated to supporting choreographies or realizing orchestrations and vice versa. If any changes occur they are calculated again and their conformance is checked. The algorithm stops, if (i) all dependencies between supported choreographies and supporting choreographies or realizing orchestrations are processed and thus a stable model exists that satisfies all temporal restrictions or (ii) a conformance condition is violated.

Algorithm 1: temporalConformanceFederationUbcLbc()

```
// initialization and precalculation
conf:=true;
initialize(C_g);
calculate(C_g, C_g.deadline);
incorporateUbc(C_g, C_g.deadline);
conf:=checkConformance(C_g);
for all directly and indirectly supporting choreographies and realizing
orchestrations G of C_g in a topological order do
    initialize(G);
    change:=propagate(C_g, G);
    if change = true then
        G.deadline := G.first.wc.eps + G.d.max;
        calculate(G, G.deadline);
    end
    change:=propagate(G, C_g);
    if change = true then
        calculate(C_g, C_g.deadline);
        conf:=checkConformance(C_g);
        mark all incoming and outgoing edges of C_g;
    end
end
// recalculation and conformance checking
repeat select randomly a marked edge e such that G is the supported
choreography and H the supporting choreography or realizing orchestration
    change:=propagate(G, H);
    if change = true then
        calculate(H, H.deadline);
        conf:=checkConformance(H);
        mark all incoming and outgoing edges of H;
    end
    unmark e;
    change:=propagate(H, G);
    if change = true then
        calculate(G, G.deadline);
        conf:=checkConformance(G);
        mark all incoming and outgoing edges of G;
    end
until all edges are unmarked ∨ conf=false;
```

Algorithm 2: initialize(G)

```
1  for all activities a ∈ G do
2  |  a.wc.eps := 0;
3  |  a.bc.eps := 0;
4  |  a.wc.lae := ∞;
5  |  a.bc.lae := ∞;
6  end
```

Algorithm 3: propagate(G, H)

```
1  change:=false;
2  for all activities {x ∈ H | ∃a ∈ G : x ≡ a} in a topological order do
   |  // Propagation of eps
3  |  if x.wc.eps < a.wc.eps or x.bc.eps < a.bc.eps then
4  |  |  x.bc.eps := a.bc.eps;
5  |  |  x.wc.eps := a.wc.eps;
6  |  |  change:=true;
7  |  end
   |  // Propagation of lae
8  |  if x.wc.lae > a.wc.lae or x.bc.lae > a.bc.lae then
9  |  |  x.bc.lae := a.bc.lae;
10 |  |  x.wc.lae := a.wc.lae;
11 |  |  change:=true;
12 |  end
13 end
14 return change;
```

Algorithm 4: calculate($G, G.deadline$)

```
1  forwardCalculation(G);
2  for all activities a ∈ G with a.pos = end do
3  |  if G.deadline < a.wc.lae then
4  |  |  a.wc.lae := G.deadline;
5  |  end
6  |  if G.deadline < a.bc.lae then
7  |  |  a.bc.lae := G.deadline;
8  |  end
9  end
10 backwardCalculation(G);
```

Algorithm 5: forwardCalculation(G)

```
1  for all activities a ∈ G in a topological order do
       // Worst Case
2      if a is the destination of a lbc(s, a, δ) then
3          a.wc.eps = Max({b.wc.eps + b.d.max | b ∈ a.pred}, a.wc.eps, s.wc.eps + δ);
4      else
5          a.wc.eps = Max({b.wc.eps + b.d.max | b ∈ a.pred}, a.wc.eps);
6      end
       // Best Case
7      if a is the immediate successor of a XOR-Join then
8          if a is the destination of a lbc(s, a, δ) then
9              a.bc.eps = Max(Min{b.bc.eps + b.d.min | b ∈
               a.pred}, a.bc.eps, s.bc.eps + δ);
10         else
11             a.bc.eps = Max(Min{b.bc.eps + b.d.min | b ∈ a.pred}, a.bc.eps);
12         end
13     else
14         if a is the destination of a lbc(s, a, δ) then
15             a.bc.eps = Max({b.bc.eps + b.d.min | b ∈ a.pred}, a.bc.eps, s.bc.eps + δ);
16         else
17             a.bc.eps = Max({b.bc.eps + b.d.min | b ∈ a.pred}, a.bc.eps);
18         end
19     end
20 end
```

Algorithm 6: backwardCalculation(G)

for *all activities* $a \in G$ *with* $a.pos \neq end$ *in a reverse topological order* **do**
  `// Worst Case`
  if *a is the source of a* $lbc(a, d, \delta)$ **then**
    $a.wc.lae = Min(\{c.wc.lae - c.d.max \mid c \in a.succ\}, a.wc.lae, d.wc.lae - \delta)$;
  else
    $a.wc.lae = Min(\{c.wc.lae - c.d.max \mid c \in a.succ\}, a.wc.lae)$;
  end
end
`// Best Case`
if *a is the immediate predecessor of a XOR-Split* **then**
  if *a is the source of a* $lbc(a, d, \delta)$ **then**
    $a.bc.lae = Min(Max\{c.bc.lae - c.d.min \mid c \in a.succ\}, a.bc.lae, d.bc.lae - \delta)$;
  else
    $a.bc.lae = Min(Max\{c.bc.lae - c.d.min \mid c \in a.succ\}, a.bc.lae)$;
  end
else
  if *a is the source of a* $lbc(a, d, \delta)$ **then**
    $a.bc.lae = Min(\{c.bc.lae - c.d.min \mid c \in a.succ\}, a.bc.lae, d.bc.lae - \delta)$;
  else
    $a.bc.lae = Min(\{c.bc.lae - c.d.min \mid c \in a.succ\}, a.bc.lae)$;
  end
end

Algorithm 7: checkConformance(G)

conf:=true;
for *all activities* $a \in G$ *in a reverse topological order* **do**
  if $a.wc.eps + a.d.max > a.wc.lae$ **then**
    conf:=false;
  end
  if $a.bc.eps + a.d.max > a.bc.lae$ **then**
    conf:=false;
  end
end
return conf;

Algorithm 8: incorporateUbc($G, G.deadline$)

```
1  violation:=false;
2  conformance:=false;
3  oldValue:=0;
4  repeat
5      for all ubc(s, d, δ) in G do
           // Worst Case
6          if s.wc.eps + δ < d.wc.eps then
7              s.wc.eps := d.wc.eps − δ;
8              oldValue := d.wc.eps;
9              calculate(G, G.deadline);
10             conf:=checkConformance(G);
11             if d.wc.eps ≠ oldValue then
12                 violation:=true;
13             end
14         end
15         if s.wc.lae + δ < d.wc.lae then
16             d.wc.lae := s.wc.lae + δ;
17             oldValue := s.wc.lae;
18             calculate(G, G.deadline);
19             conf:=checkConformance(G);
20             if s.wc.lae ≠ oldValue then
21                 violation:=true;
22             end
23         end
           // Best Case
24         if s.bc.eps + δ < d.bc.eps then
25             s.bc.eps := d.bc.eps − δ;
26             oldValue := d.bc.eps;
27             calculate(G, G.deadline);
28             conf:=checkConformance(G);
29             if d.bc.eps ≠ oldValue then
30                 violation:=true;
31             end
32         end
33         if s.bc.lae + δ < d.bc.lae then
34             d.bc.lae := s.bc.lae + δ;
35             oldValue := s.bc.lae;
36             calculate(G, G.deadline);
37             conf:=checkConformance(G);
38             if s.bc.lae ≠ oldValue then
39                 violation:=true;
40             end
41         end
42     end
43 until violation=true ∨ conf=false;
```

4.2.2 Probabilistic Time Management

Dealing with Web Services revealed, that the definition of activity durations is difficult to predict. The reason for that is the variable duration of activities in the real world as well as the forecast of branching behavior in case of XOR branches. The proposed approach presents a possibility how to consider stochastic values for the calculation of federated choreographies.
For the probabilistic time management, every basic activity needs a time histogram. A time histogram defines durations with different probability values (see table 4.2). Given a time histogram for each activity, the process including the containing activities also needs a deadline. In contrast to the calculated time constraints approach, probabilistic time management also includes probabilities for XOR branches. The affected BPEL activity in this case is the complex activity "IF". For the calculation of the probabilistic time

Activity Name	**Time Histogram**
a	(0.2,1),(0.5,5),(0.3,10)
b	(0.5,4),(0.5,8)
c	(0.1,8),(0.9,15)

Table 4.2: Example of a time histogram

management approach, "EPS" and "LAE" is calculated for each activity. As an activity has a time histogram containing several durations and probabilities, the Cartesian product is used for calculating two time histograms of two activities.

Algorithm

The algorithms for calculating the probabilistic time management are based on [2, S. 168]. Due to some calculation errors of those algorithms, a few modification had to be made. Therefore, the complete algorithms for calculating the probabilistic time histograms are presented in this work with its modifications. The main algorithm is called temporalConformanceFederation(certainty) and needs the following methods for the calculation:

- initialize(G): For all activities of the process G, the EPS value is set to $\{(1.0,0)\}$ and the LAE value is set to $\{(1.0,\infty)\}$. Additionally, the variables EPS' and LAE' are set to 0. They are used for the propagation of interval restrictions.

- propagate(G, H, certainty): This method propagates time-interval restrictions from process G to process H, if the interval [EPS, LAE] of the propagated process H gets tighter.

- calculate(G, certainty): This method uses the same technique as the method of the calculation of time constraints. EPS histograms are calculated by a forward pass, LAE histograms are calculated by a backward pass. The *Max* operator compares a newly calculated EPS histogram with an existing EPS histogram and applies the maximum histogram under a certain degree for the EPS value. The *Min* operator is responsible for the determination of the minimal LAE value under a certain degree between an existing and a newly calculated LAE histogram. EPS and LAE histograms are merged with EPS' and LAE' histograms, except EPS' or LAE' are set to 0.

- checkConformance(G, certainty): For all activities of process G is checked if the EPS histograms are bigger than the LAS histograms or the EPS histograms are bigger than the EPE histograms. If one of these conditions is true, the method returns the value false.

The algorithm temporalConformanceFederation(certainty) consists of two steps. The first step is the initialization and precalculation, the second step is responsible for the recalculation and conformance checking. Beginning with the first step, the global choreography is initialized and calculated. After a conformance check of the global choreography C_g, all directly and indirectly supporting choreographies and realizing orchestrations G are included. In a topological order, they are first initialized. Then, the propagation is applied which means that if common activities between C_g and G exist and the EPS (LAE) histograms of C_g are with a certain probability bigger (smaller) than the EPS (LAE) histograms of G, the EPS' (LAE') values of C_g are filled with the EPS (LAE) histograms of G. If a propagation occurred, the deadline of G is newly determined and G has to be calculated again because of its changing activity histograms. Afterwards, G is propagated back to C_g, if the interval [EPS, LAE] of G gets tighter. If the propagation effect any changes, C_g is calculated, its conformance is checked and all incoming and outgoing of C_g are marked. Note that in the first step only changes between the global choreography and all directly and indirectly supporting choreographies and realizing orchestrations happen.

In the second step, the recalculation and conformance checking, dependencies between **all** supported choreographies and supporting choreographies and orchestrations are considered. If possible, the outcome of the algorithm is a temporal conformant model for all choreographies and orchestrations. Otherwise the algorithm stops after a violated conformance checking.

Algorithm 9: temporalConformanceFederation(certainty)

```
// initialization and precalculation
conf:=true;
initialize(C_g);
calculate(C_g, C_g.deadline, certainty);
conf:=checkConformance(C_g, certainty);
for all directly and indirectly supporting choreographies and realizing
orchestrations G of C_g in a topological order do
    initialize(G);
    change:=propagate(C_g, G, certainty);
    if change = true then
        G.deadline := Max(G.first.eps, G.first.eps') + G.d.max;
        calculate(G, G.deadline, certainty);
    end
    change:=propagate(G, C_g, certainty);
    if change = true then
        calculate(C_g, C_g.deadline, certainty);
        conf:=checkConformance(C_g, certainty);
        mark all incoming and outgoing edges of C_g;
    end
end
// recalculation and conformance checking
repeat select randomly a marked edge e such that G is the supported
choreography and H the supporting choreography or realizing orchestration
    change:=propagate(G, H, certainty);
    if change = true then
        calculate(H, H.deadline, certainty);
        conf:=checkConformance(H, certainty);
        mark all incoming and outgoing edges of H;
    end
    unmark e;
    change:=propagate(H, G, certainty);
    if change = true then
        calculate(G, G.deadline, certainty);
        conf:=checkConformance(G, certainty);
        mark all incoming and outgoing edges of G;
    end
until all edges are unmarked ∨ conf=false;
```

Algorithm 10: initialize(G)

```
1  for all activities a ∈ G do
2  |  a.eps := (1.0, 0);
3  |  a.lae := (1.0, ∞);
4  |  a.eps′ := ∅;
5  |  a.lae′ := ∅;
6  end
```

Algorithm 11: checkConformance($G, certainty$)

```
1  conf:=true;
2  for all activities a ∈ G in a reverse topological order do
3  |  if a.eps >_certainty a.las then
4  |  |  conf:=false;
5  |  end
6  |  if a.eps >_certainty a.epe then
7  |  |  conf:=false;
8  |  end
9  end
10 return conf;
```

Algorithm 12: propagate($G, H, certainty$)

```
1  change:=false;
2  for all activities {x ∈ H | ∃a ∈ G : x ≡ a} in a topological order do
   |  // Propagation of eps
3  |  if x.eps <_certainty a.eps then
4  |  |  x.eps′ := a.eps;
5  |  |  change:=true;
6  |  end
   |  // Propagation of lae
7  |  if x.lae >_certainty a.lae then
8  |  |  x.lae′ := a.lae;
9  |  |  change:=true;
10 |  end
11 end
12 return change;
```

Algorithm 13: calculate($G, G.deadline, certainty$)

for *all activities* $a \in G$ *in a topological order* **do**
  if *a is the immediate successor of a XOR-Join* **then**
    $a.eps = Max(\{\vee\{b.epe * p_b\} \mid \forall b \in a.pred\}, a.eps)$;
  else if *a is the immediate successor of a AND-Join* **then**
    $a.eps = Max(\{\wedge_{max}\{b.epe\} \mid \forall b \in a.pred\}, a.eps)$;
  else
    $a.eps = Max(\{b.epe \mid b \in a.pred\}, a.eps)$;
  end
  if $a.eps' \neq \emptyset$ **then**
    $a.eps := a.eps \wedge_{max} a.eps'$;
    $a.eps' := \emptyset$;
  end
  $a.epe := a.eps + d$;
end
for *all activities* $a \in G$ *with* $a.pos = end$ **do**
  if $G.deadline <_{certainty} a.lae$ **then**
    $a.lae := G.deadline$;
  end
end
for *all activities* $a \in G$ *with* $a.pos \neq end$ *in a reverse topological order* **do**
  if *a is the immediate predecessor of a XOR-Split* **then**
    $a.lae = Min(\{\vee\{b.las * p_b\} \mid \forall b \in a.succ\}, a.lae)$;
  else if *a is the immediate predecessor of a AND-Split* **then**
    $a.lae = Min(\{\wedge_{min}\{b.las\} \mid \forall b \in a.succ\}, a.lae)$;
  else
    $a.lae = Min(\{b.las \mid b \in a.succ\}, a.lae)$;
  end
  if $a.lae' \neq \emptyset$ **then**
    $a.lae := a.lae \wedge_{min} a.lae'$;
    $a.lae' := \emptyset$;
  end
  $a.las := a.lae - d$;
end

For the calculation of the temporal values, the algorithms above need histogram operations. Those operations, presented in [2], are briefly discussed below.

- Histogram addition: Is used for calculating the EPE values and generates the Cartesian product of the tuples of two histograms. Probabilities are multiplied and durations are added:
  $\{(0.25, 3), (0.75, 5)\}+\{(0.5, 3), (0.5, 5)\} = \{(0.125, 6), (0.125, 8), (0.375, 8), (0.375, 10)\}$.
  Resulting tuples with equal durations are merged by summing up their probabilities (=aggregation): $\{(0.125, 6), (0.5, 8), (0.375, 10)\}$.

- Histogram substraction: Is used for calculating the LAS values and generates the Cartesian product of the tuples of two histograms. Probabilities are multiplied and durations are substracted:
  $\{(0.5, 10), (0.5, 15)\}-\{(0.3, 3), (0.7, 7)\} = \{(0.15, 7), (0.35, 3), (0.15, 12), (0.35, 8)\}$. As all resulting tuples have different durations, no aggregations is needed.

- Histogram conjunction (max-conjunction): Is used for the calculation of the EPS values after an AND-join. The outcome of this operation is a Cartesian product. Probabilities are multiplied and the maximum duration of each tuple-combination determines the duration of the resulting tuple: $\{(0.25, 3), (0.75, 5)\}\wedge_{max}\{(0.5, 3), (0.5, 5)\} = \{(0.125, 3), (0.125, 5), (0.375, 5), (0.375, 5)\}$. After aggregating the resulting tuples, the result is: $\{(0.125, 3), (0.875, 5)\}$.

- Histogram conjunction (min-conjunction): Is used for the calculation of the LAE values before an AND-Split. Again, the Cartesian product is generated, probabilities are multiplied and the minimum duration of each tuple-combination is selected for the resulting tuple:
  $\{(0.25, 3), (0.75, 5)\}\wedge_{min}\{(0.5, 3), (0.5, 5)\} = \{(0.125, 3), (0.125, 3), (0.375, 3), (0.375, 5)\}$.
  The aggregation of those tuples yields to: $\{(0.625, 3), (0.375, 5)\}$.

- Weight-operation and Histogram disjunction: Is used for the calculation of the EPS (LAE) values after (before) a XOR-Join (XOR-Split). The weight-operation multiplies all probabilities in a histogram with a given probability:
  $\{(0.25, 3), (0.75, 5)\} * 0.25 = \{(0.0625, 3), (0.1875, 5)\}$ and
  $\{(0.5, 3), (0.5, 5)\} * 0.75 = \{(0.375, 3), (0.375, 5)\}$. As the weight operation produces an invalid histogram with the sum of all probabilities less than 1.0, it always appears in combination with the histogram disjunction, which merges two weighted histograms: $\{(0.0625, 3), (0.1875, 5)\} \vee \{(0.375, 3), (0.375, 5)\} =$

$\{(0.0.625, 3), (0.1875, 5), (0.375, 3), (0.375, 5)\}$ and after aggregation $\{(0.4375, 3), (0.5625, 5)\}$.

- Histogram comparison: Is used for comparing two histograms with each other. As durations always come with its possibilities, they can not be compared without taking their possibilities into account. Thus a comparison of the form $h_1 <_{certainty} h_2$ is needed to express that h_1 is smaller than h_2 under a specific certainty. In table 4.3, two histogram are compared. The label p in the table stands for probability, t for its time (=duration) and rel for the relation between two times and their belonging probability. Based on the two histograms h_1 and h_2 in table 4.3, the following statements can be made: "up to a degree of 0.545, h_1 is greater than h_2 and up to a degree of 0.35, h_1 is equal to h_2. Thus, for instance, the following expressions are true: $h_1 <_{0.05} h_2$, $h_1 >_{0.25} h_2$, $h_1 >_{0.545} h_2$ and the following are false: $h_1 >_{0.7} h_2$, $h_1 \geq_{0.9} h_2$ [2]".

p_1	t_1
0.15	10
0.50	15
0.35	20

Histogram h_1

p_2	t_2
0.30	9
0.70	15

Histogram h_2

$p_1 * p_2$	t_1	rel	t_2
0.045	10	$>$	9
0.150	15	$>$	9
0.105	20	$>$	9
0.105	10	$<$	15
0.350	15	$=$	15
0.245	20	$>$	15

calculated results

rel	p
$t_1 > t_2$	54.5%
$t_1 = t_2$	35.0%
$t_1 < t_2$	10.5%
$t_1 \leq t_2$	45.5%
$t_1 \geq t_2$	89.5%

histogram comparison

Table 4.3: Calculating the values for histogram comparison[1]

4.3 Calculation in Run time

During the runtime of a BPEL process, the calculated constraints in design time are checked. The following two sections describe the runtime calculations of time constraints and of probabilistic time values.

[1] Example from [2]

4.3.1 Calculation of time constraints

"At process instantiation time, an actual calendar is used in order to transform all time information which was computed relative to the start of the workflow to absolute time points [59]".
For every instantiated activity, the calendar value EPS is compared with the start date of the instantiated activity. Analog to the EPS value, the mapped calendar value LAE is also compared with the end date of the instantiated activity. In addition to the EPS and the LAE values, the values LAS (=**L**atest **A**llowed **S**tart) and EPE (=**E**arliest **P**ossible **E**nd) are calculated in run time. The LAS of an activity A is calculated by the formula: $a.LAS = a.LAE - a.duration$. The EPE value of an activity A can be calculated by the formula: $a.EPE = a.EPS + a.duration$. The monitoring of a timed process during the execution of a BPEL process instance is realized by the traffic light approach, presented in [60] (See table 4.4).

Color	**Explanation**	**Formula**
Green	The process can be finished in time without any delays	$now \leq BC\ LAS$
Yellow	Yellow indicates, that a delay occurred in the past, but the process still can be finished within the specified maximum duration of the process. Future delays should be avoided, as the process is already delayed	$now \leq WC\ LAS$
Red	Red means, that the probability to miss the deadline is high.	$now \geq WC\ LAS$

Table 4.4: Monitoring the process health enabled by the traffic light system

4.3.2 Probabilistic Time Management

The probabilistic time management approach during run time can be also monitored by the traffic light model. Eder and Pichler describe this model in [61]. A prerequisite for this approach is the representation of a table with different duration values and the cumulated probabilities like presented in table 4.5. In other words, probabilities with the same duration values are summed up. At the instantiation of a process in run time, two thresholds can be defined. "The first determines the workflows state-change from green to yellow (warn) and the second determines the state-change from yellow to red (alarm) [61]". The defined thresholds are compared with the cumulated probabilities c_i like depicted in

table 4.5. For instance, the first threshold is set to 90% and the second one to 50%. If the activity, presented in table 4.5, of a process instance requires 22 time points till it is finished, the status has to be switched from green to yellow, because the 90% threshold has not been reached. If the calculated end time value e_i does not match with the actual end time like in the example above, the nearest higher value must be chosen. In the example, it would be 96%. The threshold values can be chosen freely, the higher the values the

c_i	p_i	d_i	s_i	e_i
0,048	0,048	24	24	29
0,480	0,432	27	21	26
0,528	0,048	29	19	24
0,960	0,432	32	16	21
0,964	0,004	38	10	15
1,000	0,036	41	7	12

Table 4.5: Duration histogram of an activity with the cumulated probability c_i, the probability p_i, the duration d_i, the start time s_i and the end time e_i[3]

riskier the adjustment. It is also possible to define more than two thresholds for an exacter prediction of executed activities.

[3]Based on the example from [61]

Chapter 5

Prototypical Implementation

The main task of this thesis is the extension of the business process execution language with probabilistic time management and time constraints of choreographies. For that purpose the selection of an appropriate BPEL-Designer and BPEL-Engine was required. Several solutions exist for modeling and executing BPEL-Processes, like ActiveVOS, Oracle BPEL Process Manager, NetBeans Enterprise Pack etc. For this prototypical implementation the following open source software, written in Java, was used.

1. BPEL-Designer: Eclipse BPEL-Designer, because it supports the current BPEL 2.0 specification and is stable.

2. BPEL-Engine: Apache ODE (**O**rchestration **D**irector **E**ngine), because it can be integrated in the Eclipse BPEL-Designer and also supports the BPEL 2.0 specification and has a stable release.

This chapter is organized as follows: Section 1 describes the architecture of the used software, a guide how to compile and install the chosen software from its sources as well as possible problems during this process. In Section 2, the modification of the selected software will be presented and section 3 provides an overview of the validation of the modified software.

5.1 Development Environment

As BPEL is a language in which activities are implemented through Web Services, the software environment for modeling and running the BPEL processes is quite complex. Especially for the development of BPEL extensions, the building of a stable development environment can be difficult due to insufficient documentations.

5.1.1 Requirements

For extending the Eclipse BPEL-Designer and Apache ODE a working **J**ava **D**evelopment **K**it (=JDK) is obligatory. According to the Eclipse BPEL-Team, every JDK version bigger than 1.5 works fine for the Eclipse BPEL-Designer and Apache ODE. Java is a platform independent language, therefore the software mentioned above can be developed and used in several operations systems.

5.1.2 Eclipse BPEL-Designer

The Eclipse BPEL-Designer is an open source project under the Eclipse Technology Project. This software provides a GEF based Designer, in which the BPEL processes can be created, opened, viewed, graphically modeled by drag and drop and saved as ".bpel" files. It also allows the modification directly in the ".bpel" source files. An EMF model represents the BPEL 2.0 specification. The BPEL Designer contains a validator that checks the EMF model for a BPEL conformed language based on the specification. For executing the BPEL processes in the Engine, a runtime framework allows their deployment and execution.

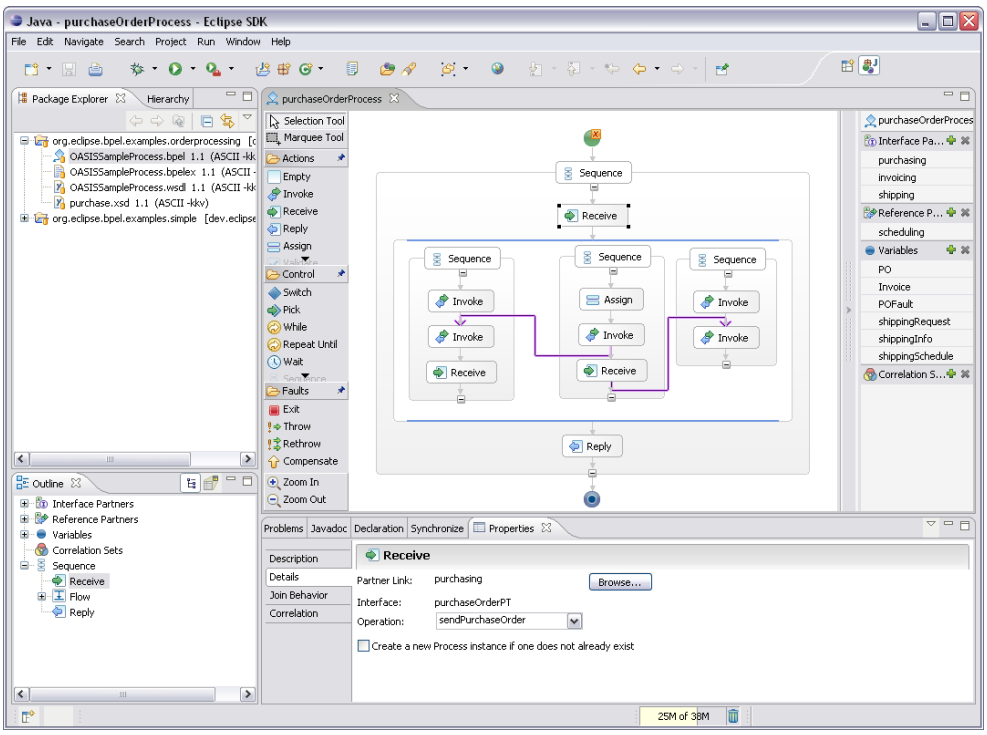

Figure 5.1: Eclipse BPEL Designer[1]

[1]Source: http://www.eclipse.org/bpel/images/screenshot.png

EMF & GEF

EMF (=**E**clipse **M**odeling **F**ramework) and GEF (=**G**raphical **E**diting **F**ramework) are the frameworks, on which the BPEL-Designer is based. The Eclipse Modeling Framework allows the design and implementation of a structured model. The model can be defined graphically (see figure 5.2) and its Java code generated automatically. That helps to keep the focus on the model itself and prevents the user from making errors while implementing the model. The generated code can be extended after the code generation. While using Java Annotations the extensions of generated code are not discarded after a next code regeneration.
The model can be serialized over one of the interfaces provided by the EMF Persistence framework (Resource, ResourceSet, Resource.Factory and URIConverter). The Eclipse BPEL-Designer serializes the model in XML files.

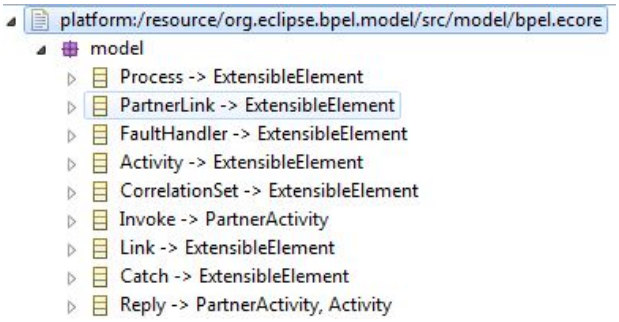

Figure 5.2: Eclipse Modeling Framework

The Graphical Editing Framework allows the building of graphical user interfaces that correspond with existing models. The graphical part is done by the Draw2D framework. GEF is based on a MVC (=**M**odel **V**iew **C**ontroller) architecture (see figure 5.3). MVC handles the connection between the model and the graphical representation.
A user interacts with the graphical representation (View). In case the user makes some changes in the graphical editor, the view passes a request to the controller. Using a command, GEF updates the model with the changes made by the user. The model fires an event to the view and changes in the graphical editor are visible to the user.
GEF provides the functionality that a change in the graphical editor effects in the first place only a change in the model without persisting that change in the XML files. To persist the changes, the file has to be saved by CTRL+S or over the menu File/Save. An example for a graphical Editor is presented in figure 5.1.

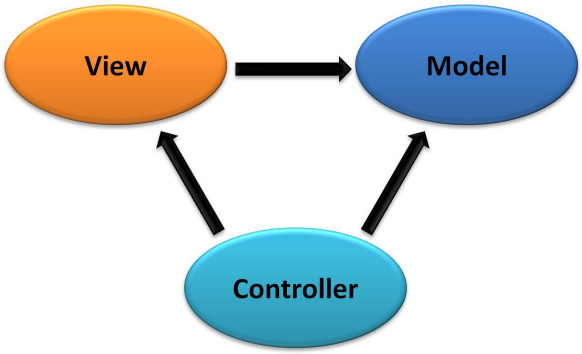

Figure 5.3: Model View Controller

Installing the sources

For the development of the Eclipse BPEL-Designer, a working Java runtime environment (at least JDK 1.5) and Eclipse (at least 3.3) is required. To be sure that the Eclipse BPEL-Designer can be installed properly it is recommended to use exactly the Eclipse version 3.3, because the BPEL-Designer was tested in this release. The prototype of this thesis was developed in the newest version of Eclipse (Eclipse HELIOS), due to a lot of comfortable usability and performance updates from the old version to the current version. During the development of the prototype, only a handful errors occurred in the new version of Eclipse. After installing Eclipse, the BPEL-Designer can be installed as a plug-in over the menu Help/Install new software. Therefore, a new remote update site with the URL "http://download.eclipse.org/technology/bpel/update-site/" has to be created. Then the Eclipse BPEL-Designer can be installed. Required frameworks like EMF, GEF, JEM and WST will automatically be installed, if they have not been installed already.
The original source code can be downloaded over a CVS repository:

- Host: dev.eclipse.org
- Repository path: /cvsroot/technology
- User: anonymous
- Password: anonymous
- Connection type: pserver

The prototype of this thesis is packed in an archive file and can be imported over the menu "File/import/General". A new window "Import" appears. "General/Existing Projects into

Workspace" has to be clicked after clicking "next". The zipped prototype can be selected and the projects imported. Running the BPEL-Designer from its sources is facilitated over the menu "Run/Run as/Eclipse Application".

Problems

In case of compile errors after building the project from its sources, the following instructions should help.

- Under menu "Window/Preferences/Java/Compiler/Errors/Warnings", the section "Deprecated and restricted API" can be expanded. The forbidden references (access rules) should be switched to "Warning" instead of "Error".
- In case of the error message "The container Plug-in Dependencies references non existing library /ECLIPSE_HOME/plugins/javax.wsdl_1.5.1.v200806030408.jar", the file "javax.wsdl_1.5.1" can be download in the Internet and has to be moved in the /ECLIPSE_HOME/plugins/ path.
- If the exception "java.lang.OutOfMemoryError: PermGen space" is thrown, the Java Virtual Machine has not enough memory. To change this, the dialog "Run/Run Configurations" has to be opened. In the tab "arguments", "-XX: PermSize=64m -XX: MaxPermSize=128m" should be raised. Those settings can be also changed in the Eclipse configuration file "eclipse.ini".

In this section, an overview of the architecture of the BPEL-Designer was given as well as an instruction how to build the designer from its sources. The next section describes how to install the BPEL-Engine Apache ODE and how to integrate it into the Eclipse BPEL-Designer.

5.1.3 BPEL Engine - Apache ODE

In Apache ODE, BPEL conform business processes can be executed. Apache ODE talks to Web Services, sends and receives messages and handles data manipulation and error recovery as described by the process definition [62].

Installing the sources

For the development of the prototype, Apache ODE 1.3.4 was used because it is the latest stable release. After downloading of the source distribution of Apache ODE there are two

ways how to build in Eclipse. The first way is to use "Buildr".
Apache Buildr is a system that enables the building of java-based applications. Under windows 7 x64, it was not possible to build Apache ODE from its sources. Buildr under Linux works better, but still has problems with building Apache ODE properly. The second and recommended way to build the sources is to use Maven 2.
Apache Maven is a software project management tool, that runs the whole life cycle of a software (building, tests, generating of documentation and reports etc.). Each project of a software project contains a configuration file "pom.xml", where "pom" stands for **P**roject **O**bject **M**odel. Maven makes the building of the process easy because it automatically downloads dependencies like Java libraries or Maven plug-ins.
After downloading and installing Maven 2 the following configurations have to be done:

- First step is adding the bin directory of Maven to the PATH
- Step two is setting the JAVA_HOME variable to the location of the JDK

After downloading the Apache ODE sources and installing & configuration of Maven 2, the following command has to be executed in the folder of the Apache ODE sources in order to build an Eclipse project:

- *mvn eclipse:eclipse*

In the next step, the Apache ODE sources can be imported in Eclipse under "File/Import/Existing Projects into Workspace". The root directory of the Apache ODE sources has to be selected, afterwards the project can be imported.
The execution of Apache ODE is described in the following section.

Preparing Apache ODE

Since Apache ODE is a BPEL-Engine that listens and executes Web Services, it has to be run in a web server. For this prototype, the web server Apache Tomcat was used. To integrate Apache ODE in the web server, the sources have to be compiled to a ".war" file (=**W**eb **A**rchive). This can be done in the root directory of the Apache ODE sources by the command

- *mvn install -DskipTests*

The compiled ".war" file is placed in "/distro/target/apache-ode-war-1.3.4.zip" of the Apache ODE root directory. In the zipped file "apache-ode-war-1.3.4.zip", the file "ode.war" has

to be copied into the Tomcat's webapp directory "PathToApacheTomcat/webapps". Afterwards, the Apache Tomcat server has to be started to configure the "ode.war" file properly in the "PathToApacheTomcat/webapps/ode" directory. For that purpose, the directory has to be changed to "PathToApacheTomcat/bin". The execution of the following command starts and configure Apache Tomcat:

- *catalina start*

The next step is the integration of Apache ODE in the Eclipse BPEL-Designer, to deploy the created BPEL-processes in Apache ODE.

5.1.4 Integrating Apache ODE in Eclipse BPEL-Designer

For integrating the Apache ODE as a server in Eclipse, the "server view" has to be opened by the menu bar "Window/Show View/Other". Then, the item "Servers" under the folder "Server" has to be opened. In the server view, a right-click opens a dialog, where a new server has to be selected. The configuration of the appearing window should be similar to figure 5.4. After this step, a new Apache ODE server appears in the "Server view". With

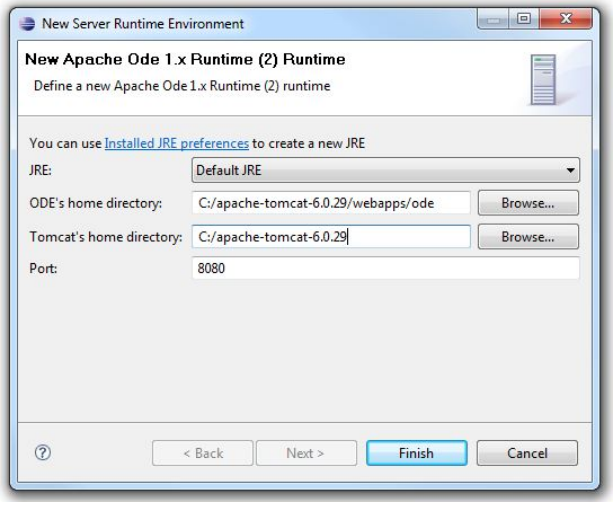

Figure 5.4: Configuration of Apache ODE in Eclipse

a right-click on the server, the context menu entry "Add and Remove Projects" can be selected. In the appearing window, BPEL processes can be added or removed. In case of adding a BPEL process to the server, the process has to be published. This can be done over a right-click on the server and the selection of the context menu entry "Publish". The system is now prepared to execute BPEL processes.

Problems

In case of errors, the following instructions should help:

- In the Eclipse Build Path, the classpath variable "M2_REPO" has to be set to the Maven 2 repository. This can be done from the menu bar by selecting "Window/Preferences". A preferences dialog appears. Under "Java/Build Path/Classpath Variables", a new variable should be created with the name "M2_REPO". That variable has to point to the Maven 2 repository, which is normally in the "/home/.m2" directory.
- Maven 2 can be integrated as a plug-in in Eclipse. It is highly recommended not to install this Plugin because the building process of Apache ODE does not work properly with that plugin. The Maven 2 integration of Eclipse always removes the target directory, where the generated byte codes of XMLBeans and OpenJPA are saved.
- In case of a System.OutOfMemoryException, a environment variable "MAVEN_OPTS" should be created with the value "-Xmx512M".
- If there are still building errors after considering the instructions above, the Apache ODE sources should be rebuilt with the command "*mvn eclipse:eclipse*".
- Alternatively, the link *http://eclipse.org/bpel/users/pdf/HelloWorld-BPELDesignerAndODE.pdf* gives also hints about installing & configurating the Eclipse BPEL-Designer

5.2 Modifications

In this section, the modified prototypes of the Eclipse BPEL-Designer and the Apache ODE BPEL-Engine are described. Starting with the Eclipse BPEL-Designer, the modifications of the graphical user interface are depicted as well as the used algorithms for the calculations of time constraints and probabilistic time values. Later the most important Java classes are described, in which the modifications were made and how exceptions are handled there. Next, the modifications of Apache ODE are described. An overview of the most important Java classes is given. It is also explained, how the time calculations of the BPEL designer are checked in Apache ODE. After that, the section Validation provides information on how the whole system was tested.

5.2.1 Eclipse BPEL Designer

The implemented prototype calculates constraints in two ways. On the one hand, every activity gets time values like a minimum duration, a maximum duration and durations between two activities. On the other hand, every activity gets durations in combination with probabilities. The first way of calculation is called calculation of time constraints, the second one probabilistic time management. Both types of calculations consider the approach of federated choreographies. That means, that not only one BPEL process is calculated, but several BPEL processes that depend on each other.

Graphical User Interface

In this section, the modified graphical user interface of the Eclipse BPEL-Designer is presented. The modified BPEL Designer allows to define activities with time values. Consider that only basic activities like *Empty, Invoke, Receive, Reply, OpaqueActivity, Assign, Validate and Wait* can have time values. Based on figure 5.5, the 3 most important changes are explained. The modification of the GUI mostly concerns the property view, which is marked in figure 5.5. In general, the Eclipse BPEL-Designer manages all properties of activities, controls etc. in that property view. A property view is divided in several sections. On the left side of the property view are 3 sections in red rectangles. Those rectangles show the added sections of the prototype.

The first section is called choreography. In this section dependencies of choreographies and orchestrations can be defined (see figure 5.6). Under “Select Process”, a process can be chosen in a combo box which should be marked as a supported process. In the underlying combo box, another process can be selected which should be marked as the supporting process. After pushing the “Add” button, a XML file “dependencies.xml” is written in which the dependencies are visible. The button “remove” removes an already existing dependence between a supported and a supporting choreography. In figure 5.6, the choreography G will be supported by S1.

The next property section is called constraints. This section provides all the necessary dialogs for calculating the time constraints (see figure 5.7). The necessary fields for the calculation of the time constraints are the duration and the process deadline. The optional fields are the lower bound constraints and the upper bound constraints. For adding a lbc or ubc, the button “Add” has to be pressed. Lower bound constraints and upper bound constraints can be set as often as necessary. They can be removed separately over the

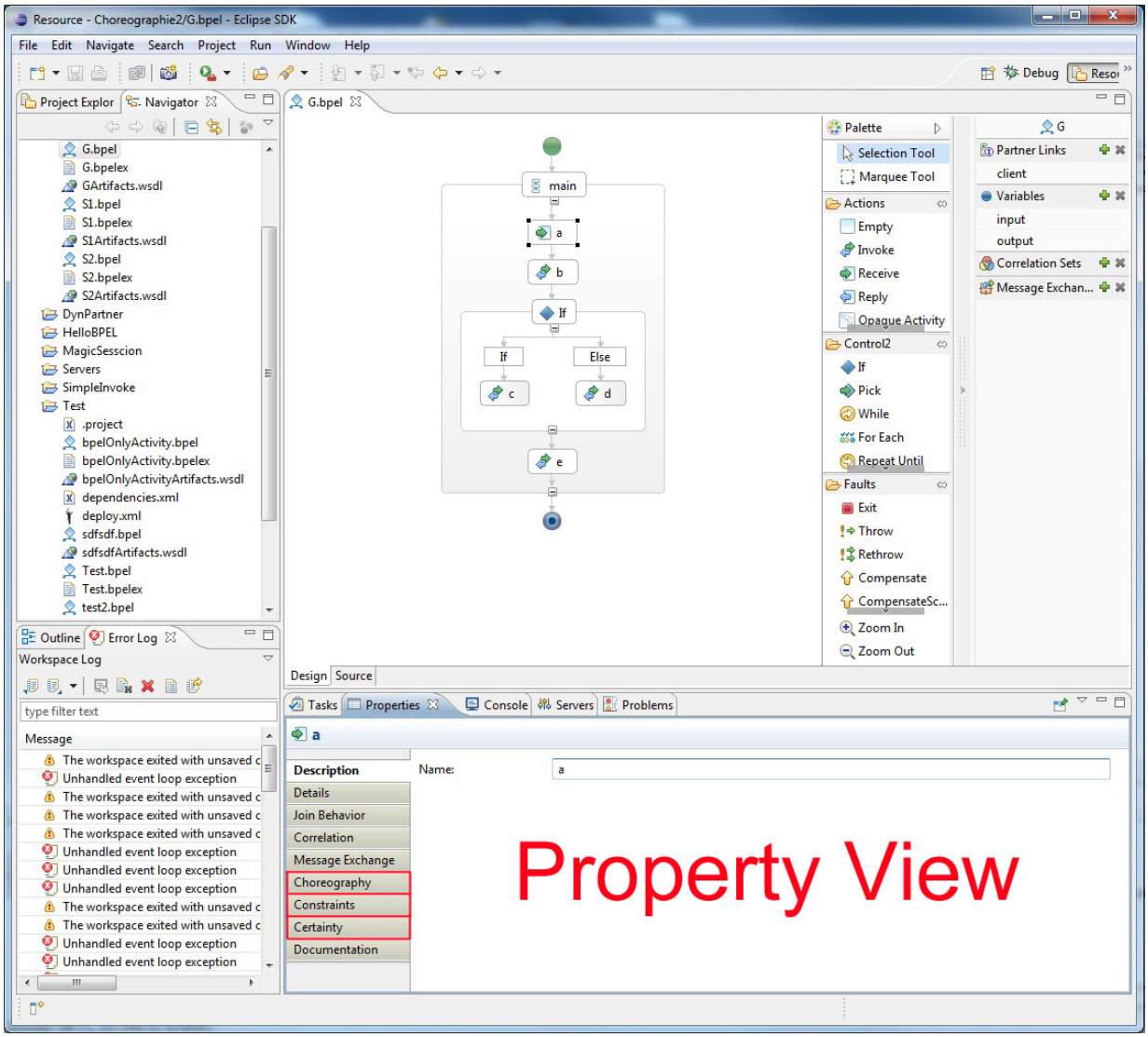

Figure 5.5: Changes in the GUI BPEL-Designer

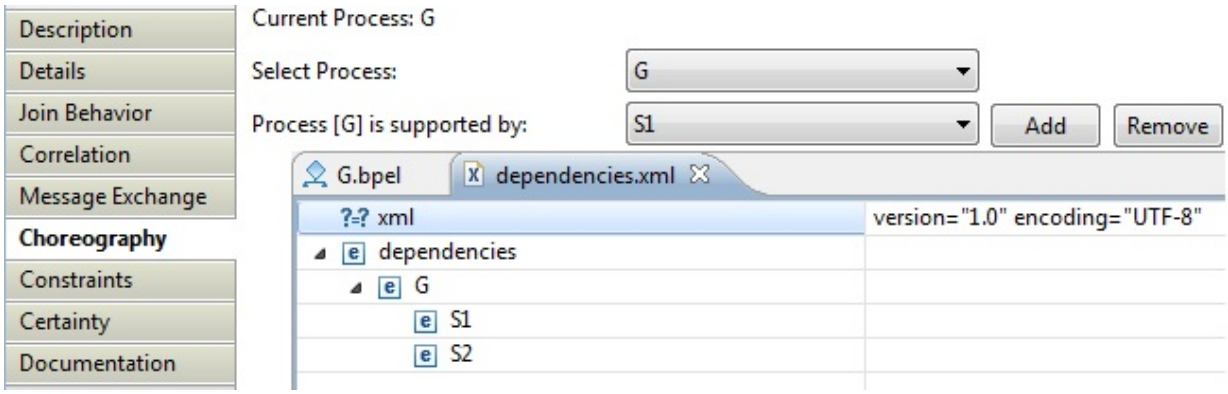

Figure 5.6: Managing dependencies of choreographies

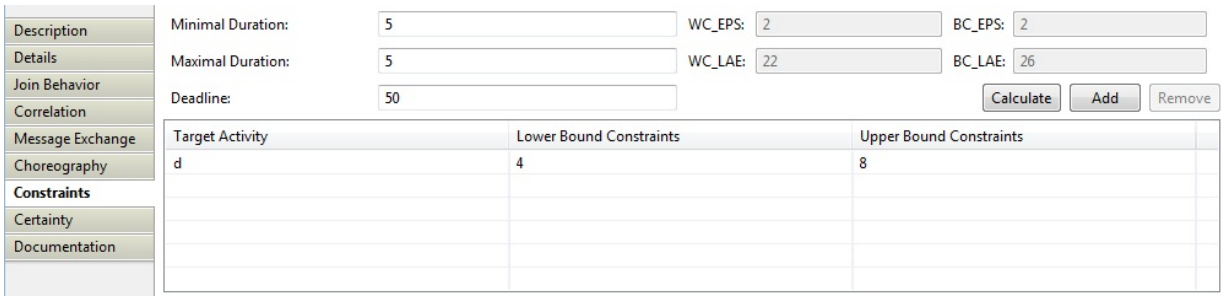

Figure 5.7: Calculating time constraints

"remove" button. For creating a lbc or an ubc, a target activity has to be chosen. Then a lbc or (and) a ubc can be defined. After the creation of a lbc or a ubc an arrow connects the source activity with its target activity (see figure 5.8). The fields "WC_EPS, BC_EPS, WC_LAE and BC_LAE" are not changeable because they only inform the user about the calculated values after a performed calculation of a process.

The third new section in the property view of the Eclipse BPEL-Designer is called "Certainty". This section is required to calculate a process using the probabilistic time management approach. For every process, a deadline have to be set and for every activity of a process, at least one duration with a given probability has to be defined. The so called time histogram for each activity can be created, extended or removed by the two buttons "Add" and "Remove". Furthermore, probability values have to be set for every XOR split, which concerns the control activity "IF". Figure 5.9 shows the GUI where the probabilistic time values for activities can be set. To add or change values for XOR splits, the user has to click on a branch like "If", "Else" or "ElseIf". A new field with the name "branching probability" appears. Note, that XOR split nodes can not have durations. Because of this,

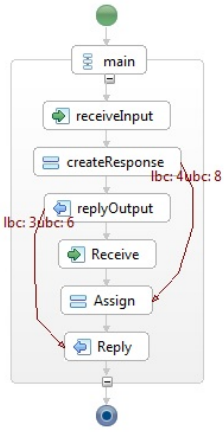

Figure 5.8: Representation of lbc and ubc in the graphical editor

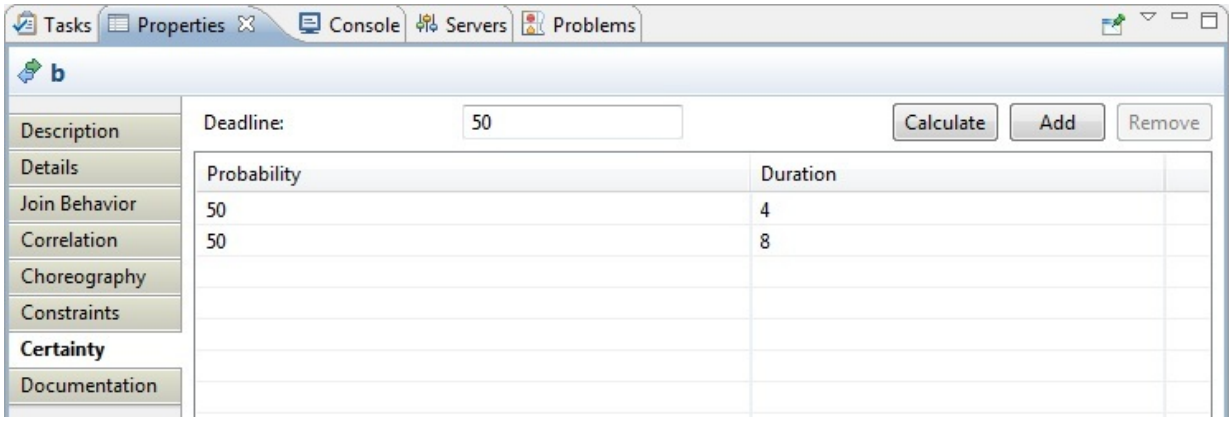

Figure 5.9: Representation of the graphical interface for probabilistic time management

the button for adding or removing time histograms is disabled in this view. In contrast to the calculation of the time constraints approach, the calculated values of the probabilistic time management are stored in an external file, which is called “certaintyValues.xml”.

Data Model

In this section, the extended data model for the implemented prototype is presented. In the eclipse modeling framework, the root object of every model is called “EObject”. In case of the BPEL-Designer, the “ExtensibleElement” extends the “EObject”. Every activity (basic activities and control activities) of the BPEL-Designer extends the “ExtensibleElement”. The data model of the prototype also extends the “ExtensibleElement”. That enables every activity to access the data model of the presented prototype. Figure 5.10 shows the class diagram of the data model. The class “Constraint” and “Probability” extends the “Exten-

sibleElement". The "ExtensibleElement" and thus every "Activity" has access to the class "Constraint" and "Probability" over the methods getConstraint(), setConstraint(), getProbability() and setProbability(). Every "Constraint" has getter and setter methods for the attributes "BC_EPS", "BC_LAE", "WC_EPS", "WC_LAE", "minDuration" and "maxDuration". Additionally, a method getConstraintSet() with the return value "Constraints" exists. "Constraints" contains the attributes "lbc", "ubc" and "targetActivity". As far as an activity can have several lower bound constraints and upper bound constraints, one "Constraint" refers to many "Constraints". Every "Activity" refers to exactly one "Constraint" or one "Probability".

Analog to the class "Constraint", the class "Probability" also refers to many "Probabilities" by the method getProbabilitySet(). GetProbabilitySet() returns the class "Probabilities" with the attributes duration and certainty. The class "Probability" itself does not contain any attributes.

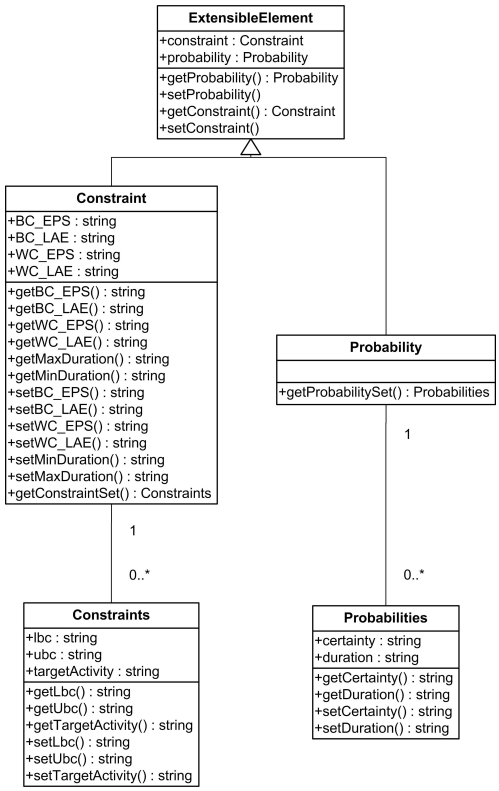

Figure 5.10: Class diagram of the data model

Description of the created Classes

This section contains a short overview of the most important classes, used for the modification of the graphical user interface and the created algorithms for the calculations (see table 5.1).

Package: org.eclipse.bpel.ui.properties

This package contains all the classes, which are responsible for the modification of the property view in the Eclipse BPEL-Designer.

Class	**Function**
ChoreographySection.java	Creates the section "Choreographie" in the property view, where dependencies of choreographies and orchestrations can be managed
ConstraintsSection.java	Responsible for the property view section "Constraints", where a timed graph can be defined and calculated afterwards
CertaintySection.java	This section is used for calculating a process with the probabilistic time management approach

Table 5.1: Created classes for the GUI of the BPEL-Designer

Package: org.eclipse.bpel.ui.util

This package contains the classes for the calculations of constraint and probabilistic time values.

Class	**Function**
BPELConstraintsCalculator.java	Calculation of the time constraints approach
BPELCertaintyCalculator.java	Calculation of the probabilistic time management approach
CalculatorUTIL.java	Helper Class for several common used methods by the time constraints and probabilistic time management approach

Table 5.2: Created classes for the Calculation in the BPEL-Designer

Exception Handling

The exception handling is important for a calculation free from errors. Before starting a calculation with the time constraints approach, several checks have to be performed. A process needs a set deadline and for all activities, the definition of the duration is obligatory. If lower bound constraints or upper bound constraints are set, also a target activity has to be set. A target activity can't be defined without setting either a lower bound constraint or an upper bound constraint. Furthermore, a lbc has to be smaller than an ubc as well as the minimum duration has to be smaller than the maximum duration. In case of an "IF" control or an "AND" control, no lbc or ubc constraints can be set between the branches.
For the calculation with the probabilistic time management approach, a deadline must be set. For every activity, the sum of the probability values has to be 100. Using "IF" control activities, all branches need a probability value and the sum of all branches also has to be 100.
For every exception, a dialog appears with the specific problem message and the name of the activity, in which the problem exists.
A process has to contain unique activities, otherwise calculation errors can occur due to redundancy.

5.2.2 Apache ODE

In the BPEL-Engine Apache ODE, the time values calculated in the BPEL-Designer are controlled. After the instantiation of the first activity, all temporal time values calculated by the Eclipse BPEL-Designer are mapped to fixed date values. At this time, a dialog pops up which contains the traffic light.
In the following section, the main classes are described in which the monitoring of the calculated BPEL processes is implemented.

Description of the created Classes

The most important class, where all the information about start-time, end-time, name etc. of activities comes together is called "ACTIVITYGUARD.java" and can be found in the package org.apache.ode.bpel.runtime. With the 2 helper classes BpelREADER.java and BpelWRITER.java, the calculated time constraints of the activities can be loaded. In the helper classes, the mapping of the calculated time points in design time to the calendar dates occur.

5.3 Validation

The validation of the prototype is divided into JUnit-Tests an human GUI-Tests. JUnit was used for testing the correct input and output of time constraint values and probabilistic values.
A JUnit test is composed of the three methods “setUp()”, “runTest()” and “tearDown()”. The “setUp()” method prepares the test environment for the underlying tests. “RunTest()” executes the particular tests. Finally, the “tearDown” method resets all the changes, which were made during the “setUp()” and the test methods.
As far as BPEL files are serialized and deserialized to XML, unit tests were used for checking weather values are written correctly to XML or read correctly from XML. The calculations were also tested with unit tests. For that purpose, a BPEL file without any time constraints or probabilistic values is read in the “setUp()” method. After that, constraint values or probabilistic values are set. In the actual test, several calculation methods of the prototype are executed. The calculated results are compared with the hard coded, expected values. After finishing the unit tests, all the changes in the BPEL files are reverted to the original state.
GUI tests were used to validate changes in the graphical user interface. After defining lower bound constraints or upper bound constraints, it was tested if the source and the target activity are connected by an arrow. Another test scenario is the correct representation of calculated values in the property view.

Chapter 6

Conclusion and Outlook

The contribution of this thesis is the implementation of a tool that enable users to augment WS-BPEL-processes with temporal information in design time and the development of a run time component for checking the temporal behavior of the processes at run time. In design time, temporal information can be expressed in two ways. On the one hand, valid temporal intervals for (activities of) each process can be defined. On the other hand, a stochastic approach can be used to express activity durations with different probabilities that also takes the conditional branching behavior into account. For both approaches can be checked, if a temporally feasible model exists, that satisfies all temporal constraints with consideration for the structure of each process and the interactions between different processes. At run time, the precalculated time values in design time are checked and the health of a process instance can be monitored. The developed prototype consists of two parts:

- A BPEL-Designer for the expression and calculation of temporal information, based on the Eclipse BPEL-Designer by IBM.
- A runtime component for checking the precalculated temporal information in design time, based on Apache ODE by the Apache Software Foundation.

Both software products are Open Source and written in JAVA. A prerequisite for the selection of the software was the Open Source criterion. Unfortunately, this restriction limited the amount of available software products of BPEL. After testing several Open Source BPEL solutions, the Netbeans BPEL-Designer and the Eclipse BPEL-Designer were shortlisted. Finally, the Eclipse BPEL-Designer by IBM was used because of the available documentations and tutorials for this product. Apache ODE was selected, because the Eclipse BPEL-Designer facilitates an integration of Apache ODE. However, both products

possess a high potential for improvement, concerning usability, stability and documentation. To make those products more interesting for companies, they should be realized as standalone platforms. As far as BPEL is a Web Service Business Execution Language, it is obvious that a BPEL-Designer and a BPEL-Engine should be realized by a Web 2.0 based application. The BPEL solution Orchestra [63] of the company BULL offers such a product, which is fully Open Source. Unfortunately, their Web 2.0 based product was not completed when the development of the prototype of this master's thesis started.

Appendix A

Installation

This section provides a step by step guide to install the Eclipse BPEL-Designer and the Apache ODE BPEL-Engine, as well as a guide to model a test choreography and execute that choreography in the BPEL-Engine. Download

- the sources of this master thesis' prototype can be downloaded at http://www.ifs.tuwien.ac.at/oesterle.
- the Eclipse IDE for Java EE Developers Galileo (3.5) Package at http://www.eclipse.org/downloads/, unpack it and name it to Eclipse-BPEL-Designer.
- the Eclipse Ganymede enterprise project bundle at https://www.ibm.com/developerworks/eclipse/downloads/ganymede/, unpack it and name it to Eclipse-BPEL-Engine
- the Java Development Kit JDK 6 update 21 at http://www.oracle.com/technetwork/java/javase/downloads/index.html and install it.
- JDOM 1.1 at http://www.oracle.com/technetwork/java/javase/downloads/index.html, unpack it and copy the file "PathToJdom/build/jdom.jar" into the directory "PathToJDK/jre/lib/ext/".
- JAXEN 1.1.3 at http://jaxen.codehaus.org/releases.html, unpack it and copy the file "PathToJaxen/jaxen-1.1.3.jar" into the directory "JDK_HOME/jre/lib/ext/".

- Apache Tomcat 6.0.29 at
  http://tomcat.apache.org/download-60.cgi and unpack it. Set the JAVA_HOME environment variable and point it to the directory, where the JDK is installed.

A.1 Configuratin of Apache ODE

Go to the directory "PathToApacheTomcat/webapps/ and copy the ode.war, which was modified in this thesis, into that directory. Then, open a command prompt and point it to "PathToApacheTomcat/bin" directory. Execute the command "catalina run". The installation of Apache ODE starts. Wait until the message appears "INFO: Server startup...". Then, hit CTRL+C to terminate the installation process. In the directory "PathToApacheTomcat/webapps", a subdirectory "ode" should be created.

A.2 Configuration of the Eclipse BPEL-Designer

Start the Eclipse IDE for Java EE Developers from the directory, where it was installed. In Eclipse, click in the menu bar on "Help/Install new Software...". In the appearing window, click on "Add...". In the field name, type in "BPEL" and point the location to "http://download.eclipse.org/technology/bpel/update-site/" (see figure A.1).

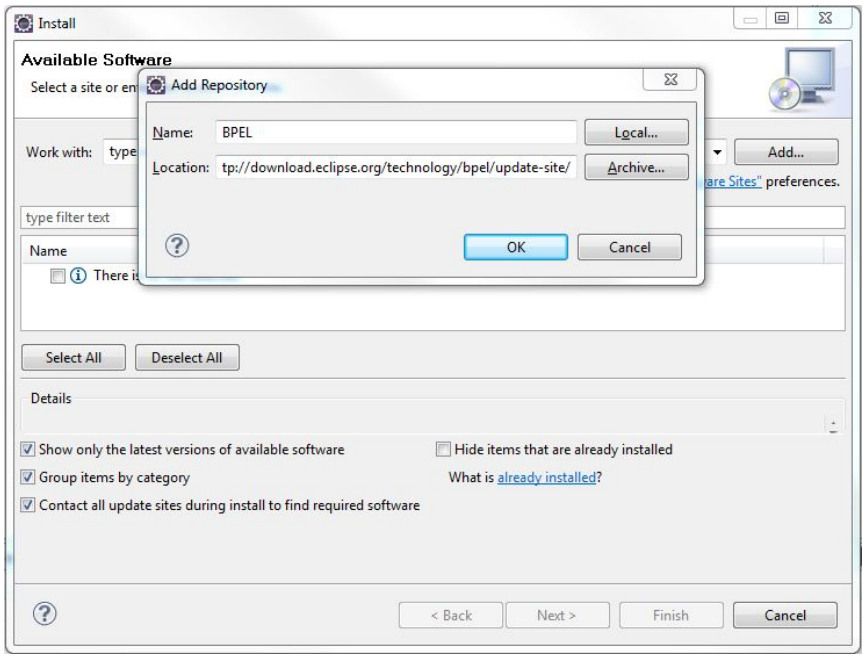

Figure A.1: Eclipse - Install new Software

Select the Eclipse BPEL Designer Nightly Build Update Site as shown in figure A.2 and follow the upcoming dialogs. After finishing the installation, a restart of eclipse is necessary.

Figure A.2: Eclipse - Install the BPEL-Designer

In the next step, the sources of the prototype of this thesis have to be installed. In Eclipse, click on the menu bar "File/Import", select "General/Existing Projects into Workspace" in the appearing window and click on next. Click on "Select archive file" and then on the button "browse". Chose the zipped prototype and click on open. To finish the import, click on "finish" (see figure A.3).

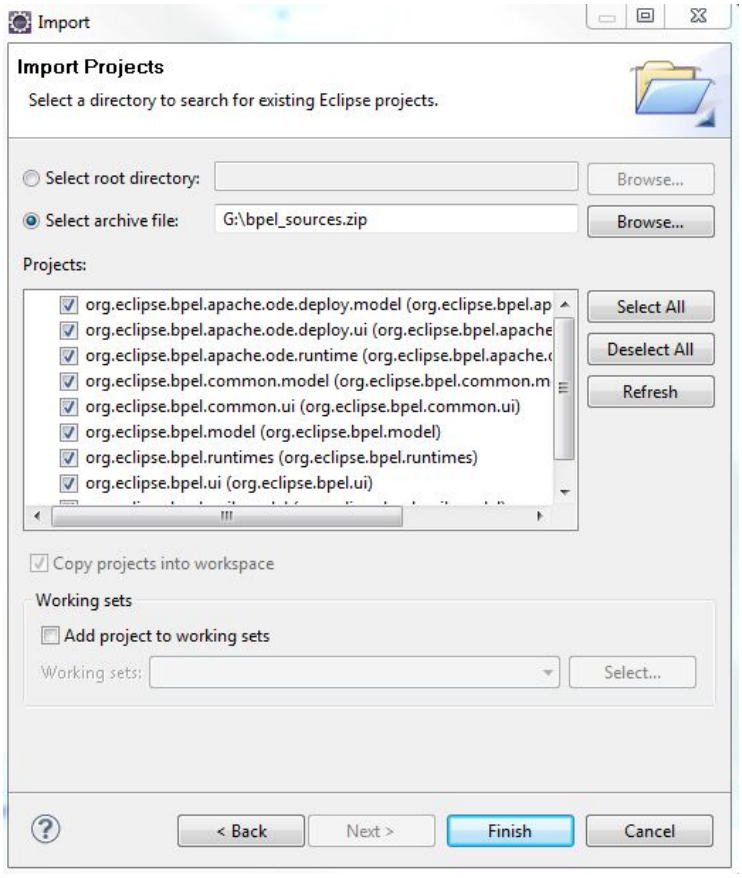

Figure A.3: Importing the BPEL prototype sources

To run the modified Eclipse BPEL-Designer, click on the menu bar “Run/Run configurations...”. With a right mouse click on “Eclipse Application”, a new Eclipse instance can be created (see figure A.4). Select a workspace location by clicking on “File System...” and choosing a directory, where the BPEL - processes will be saved. Before clicking on run, the tab “Arguments” have to be opened and the memory for the Java Virtual Machine has to be raised. Type into the field “VM arguments” the value as seen in the red rectangle in figure A.5.

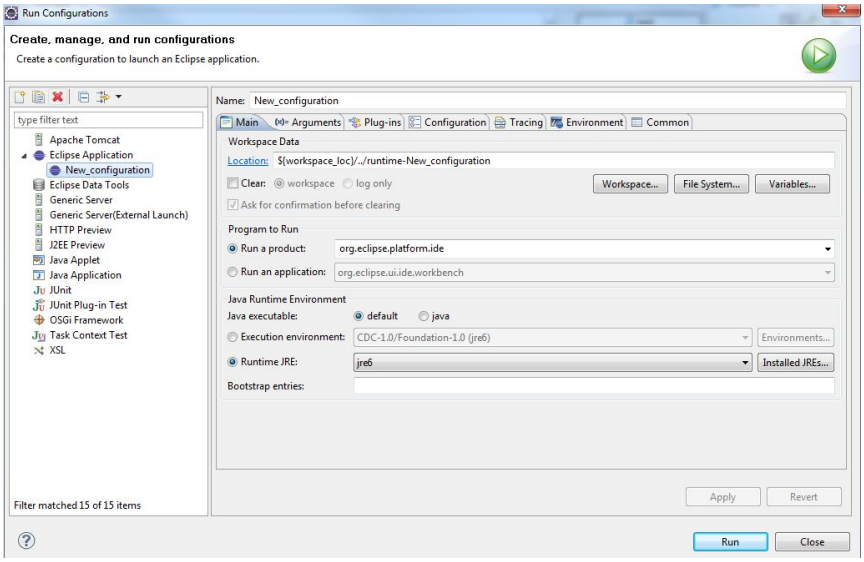

Figure A.4: Run the modified Eclipse BPEL-Designer

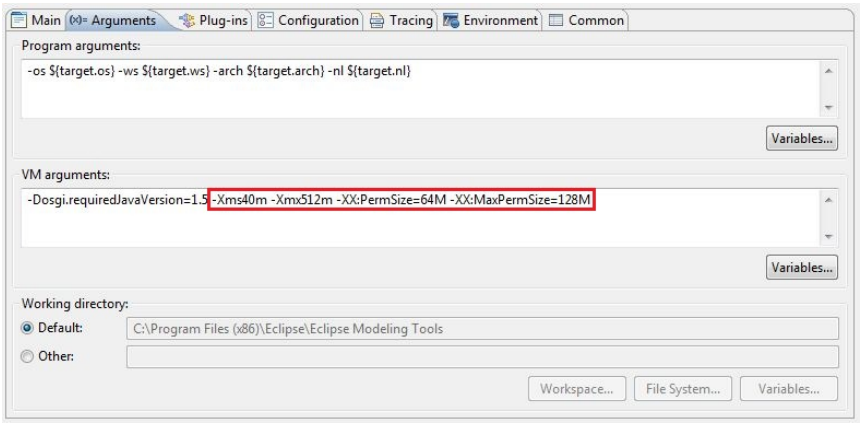

Figure A.5: Raise memory of the JVM

Now the “run” button can be pressed to launch the BPEL-Designer prototype of this thesis. A new Eclipse instance appears. Go to menu bar “Window/Show View/Other...”

and select in the folder “General” the “Properties” entry. Click again on the menu bar “Window/Show View/Other...” and select in the folder “Server” the “Servers” entry.

A.3 Creating a BPEL Choreography with time constraints

First of all, a new BPEL project has to be created. Click on menu bar “File/New/Other...” and expand the folder “BPEL 2.0” in the list. Select the entry “BPEL Project” and click on next. Give the project the name “Choreography” and select Apache ODE in “Target runtime” as shown in figure A.6. After clicking on “Finish”, a new project appears in the project explorer of Eclipse.

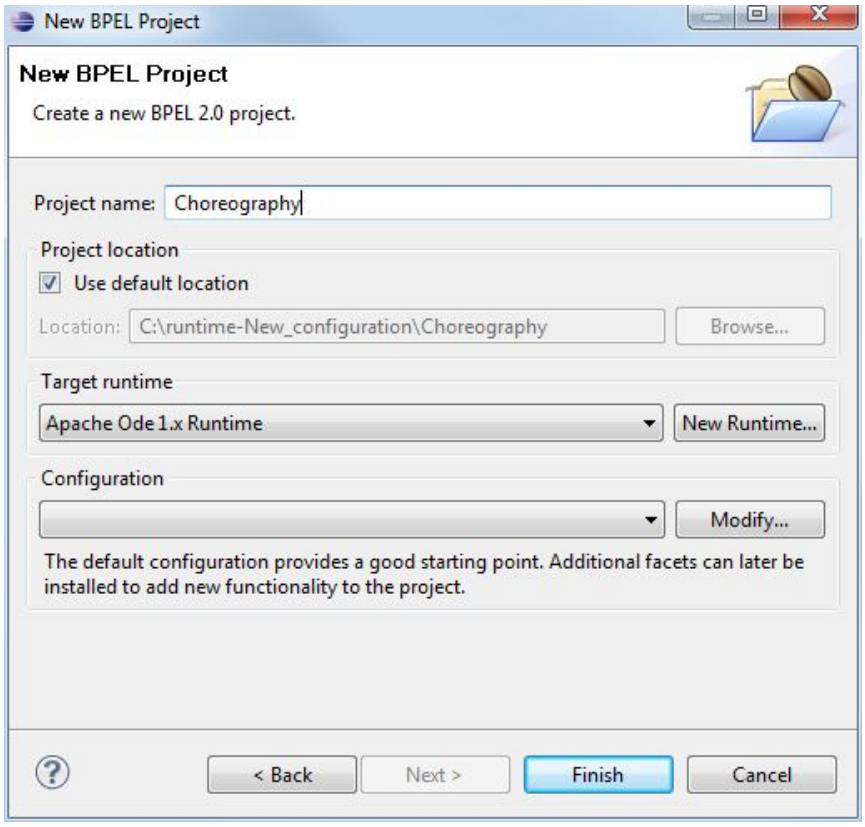

Figure A.6: Creating a new BPEL project

A BPEL process have to be added to the project. Therefore, perform a click on the created project and select “New/Other...”. In the appearing window, expand the folder “BPEL 2.0”, select “New BPEL Process File” and click on next. Fill out the fields as

shown in figure A.7. After a click on “Finish”, a new BPEL process file has been added to BPEL project “Choreography”. With a double click on the created “G.bpel” file, the process will be displayed in the graphical BPEL Editor.

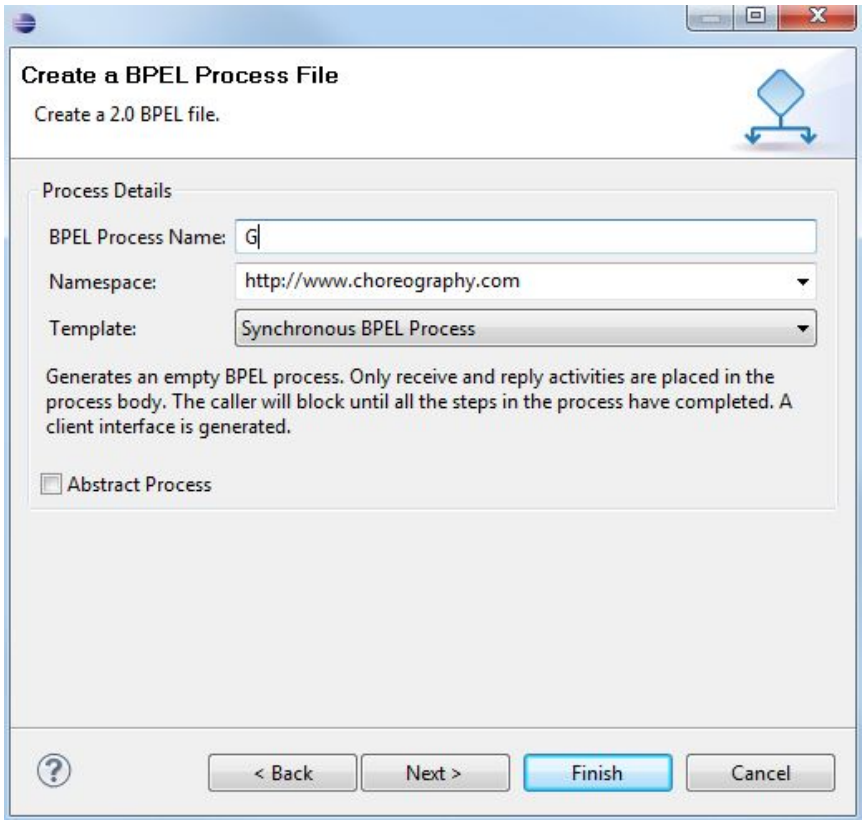

Figure A.7: Creating a new BPEL process

To start modeling the process, the palette with the activities has to be rolled out. Therefore, perform a click on the arrow on the right top of the BPEL editor (see the red rectangle in figure A.8).

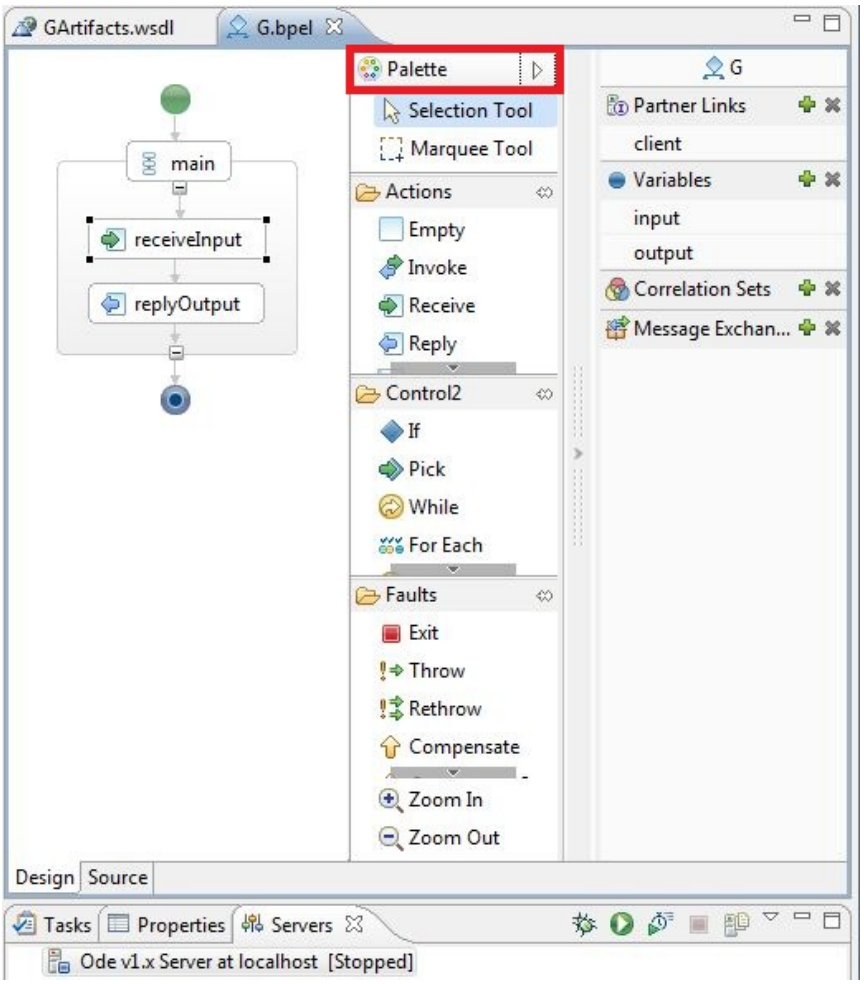

Figure A.8: Rolling out the BPEL palette

The created process “G.bpel” already contains the activities “receiveInput” and “replyOutput”. Between those two activities, create a new “Assign” activity by selecting the activity “Assign” of the palette and moving it in between the activities “receiveInput” and “replyOutput”. Perform a right click on the new “Assign” activity and select “Show in Properties”. In the section “Description”, type in the name “createResponse”. Click on the section “Details” and click on “New”. Assign the variable “input/payload/input” to the variable “output/payload/result” as shown in figure A.9. A initializer popup appears and asks whether a new initializer variable should be generated. Click on Yes. Till now, we have 3 activities in the process “G”. A “Receive” activity, which receive a message of a Web Service, an “Assign” activity, which assigns the received message to the variable input and forwards the message to the “Reply” activity.

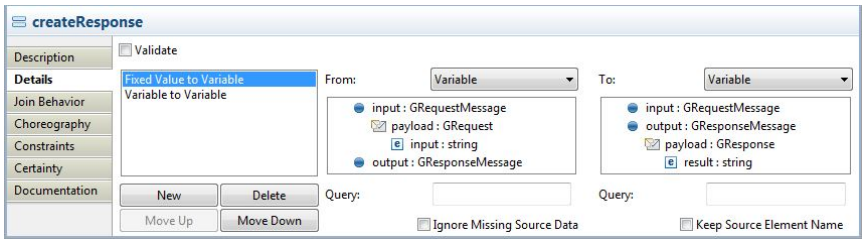

Figure A.9: Assigning variables

The next step is adding a "Flow" activity into the process after the "replyOutput" activity. Into this flow activity, add two "Wait" activities with the names "Wait5s" and "Wait10s". Furthermore, augment the process with an "If" activity. Perform a right click on the "If" activity and click on "Add ElseIf". Add a "Wait" activity after the "If" branch and name it to "Wait2s". Add also a "Wait" activity after the "ElseIf" branch and name it to "Wait4s". To create conditions for the If branches, first click on the "If" branch. Click on the section "Details" and press the button "Create New Condition". Type in *$output.payload/tns:result='ger'* as shown in figure A.10. Next, click on the "ElseIf" branch, go to the section "Details" and press the button "Create New Condition". Type in *$output.payload/tns:result='eng'*. The conditions for the "If" activity are set. If the "receiveInput" activity gets the message "ger", the activity "Wait2s" will be executed and for the message "eng", the activity "Wait4s" will be executed. For all other messages, no activity in "If" will be executed.

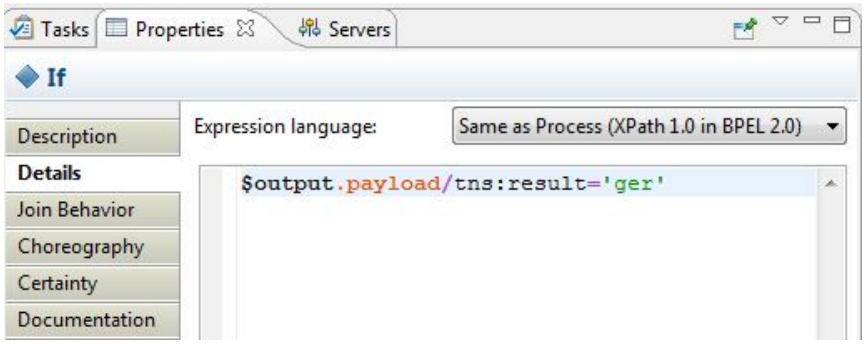

Figure A.10: Creating a condition in an "If" activity

Add a "While" activity to the process. Click on the section "Details" and press the button "Create a New Condition". The condition has the value "true()", which causes a infinity loop. Put a "Wait" activity into the "While" activity and name it to "Wait3s". In the next step, a duration for each "Wait" activity has to be defined. Therefore, select a

wait activity, click on the section “Details” and press the button “Create New Condition”. Set the duration to the amount of seconds which are defined in the name of each “Wait” activity. Repeat this step, until all “Wait” activities have a duration. The created process should look like the figure A.11 without the tables next to the activities.

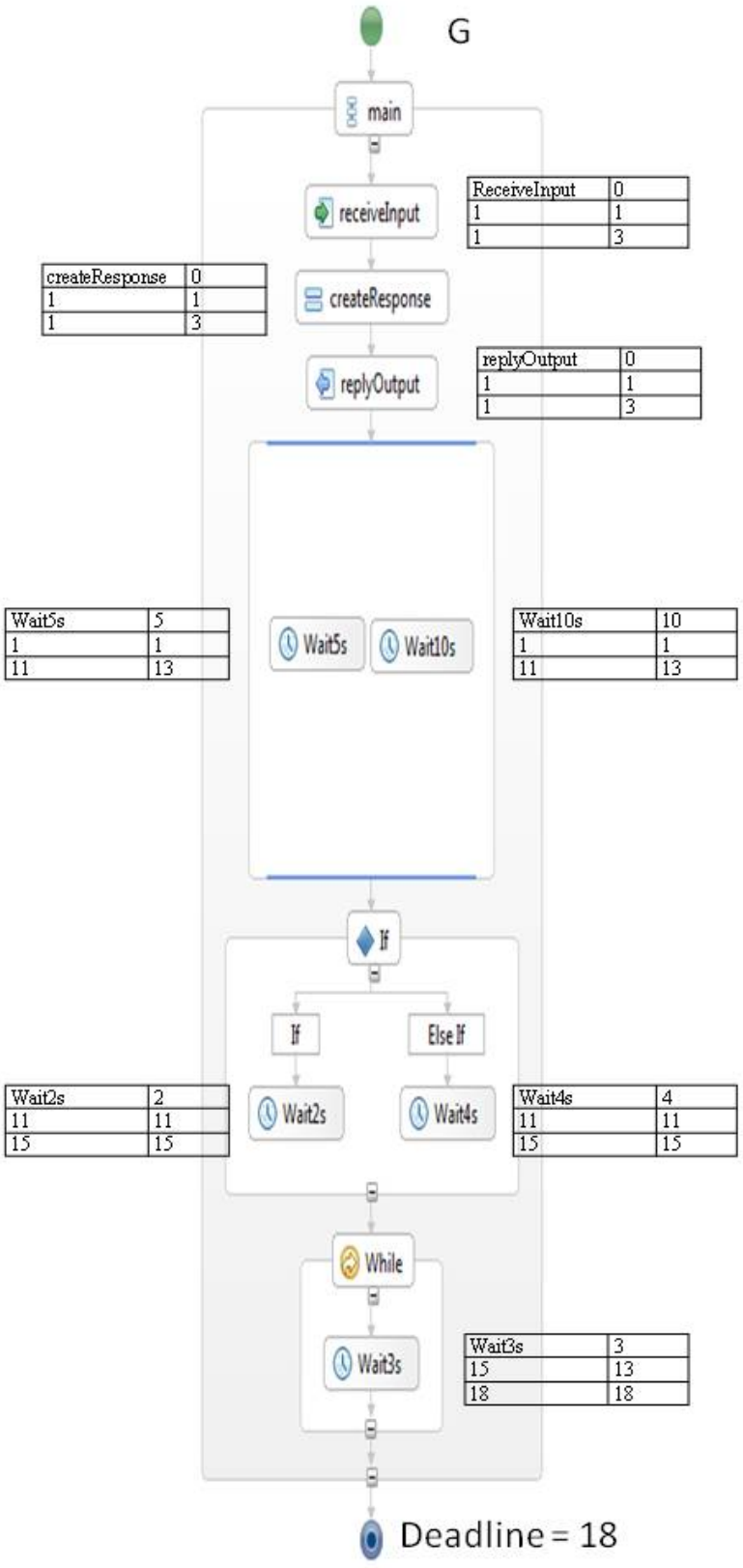

Figure A.11: Process G with time durations

The next step is to define the time values of each activity of the process “G.bpel”. Therefore, select an activity, click on the section “Constraints” and set the duration to the value according to the table beside the selected activity in figure A.11. For the whole process, a deadline has to be specified. Therefore, select an activity, click on section “Constraints” and set the deadline according to the deadline value in figure A.11.
Before the process is ready for executing, the Web Service description file “GArtifacts.wsdl” has to be prepared. Perform a double click on this file. In the appearing window, perform a right click in the blank area. Click on “Add Service” and give it the name “GBPELService”. Again, perform a right click in the blank area and click on “Add Binding”. Right-click the appearing element and select “Show properties”. Name it to GBinding. Select in the ComboBox “PortType” the entry “Browse...”. Select the port “G” in the appearing window and press OK. Finally, press the button “Generate Binding Content...”, select “SOAP” as protocol as shown in figure A.12 and click on “Finish”.

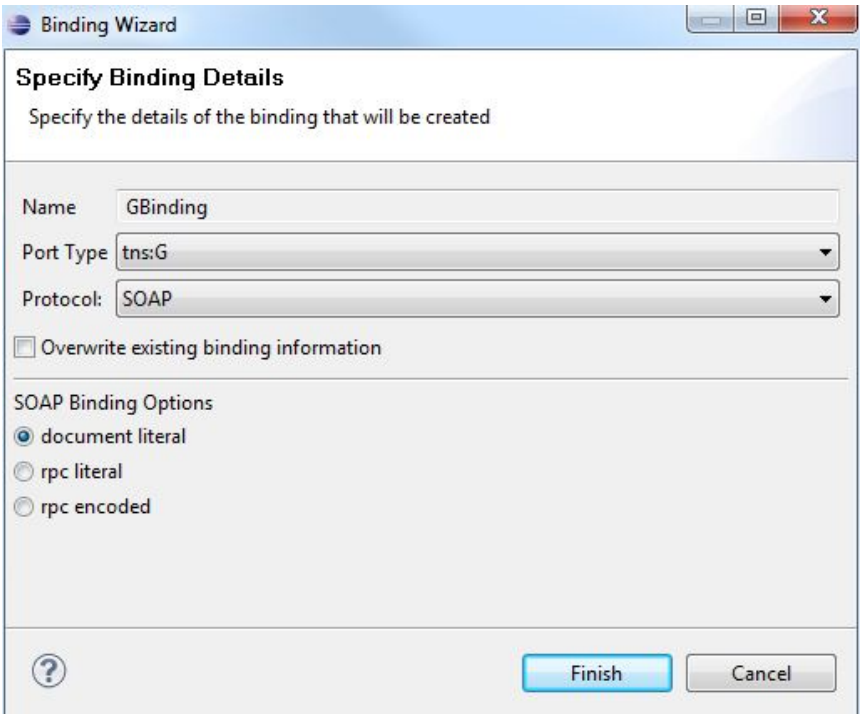

Figure A.12: Create the binding

Perform a right click on the automatically created Port of the “GBPELService” and select “Show Properties”. Name it to GPort and select the entry “GBinding” in the “Binding combo box”. In the address field, add the URL
“http://localhost:8080/ode/processes/GBPELService”. The process “G.bpel” can later be

instantiated over that URL. Finally “GArtifacts.wsdl” should look like figure A.13 Save the performed changes by pressing CTRL+S.

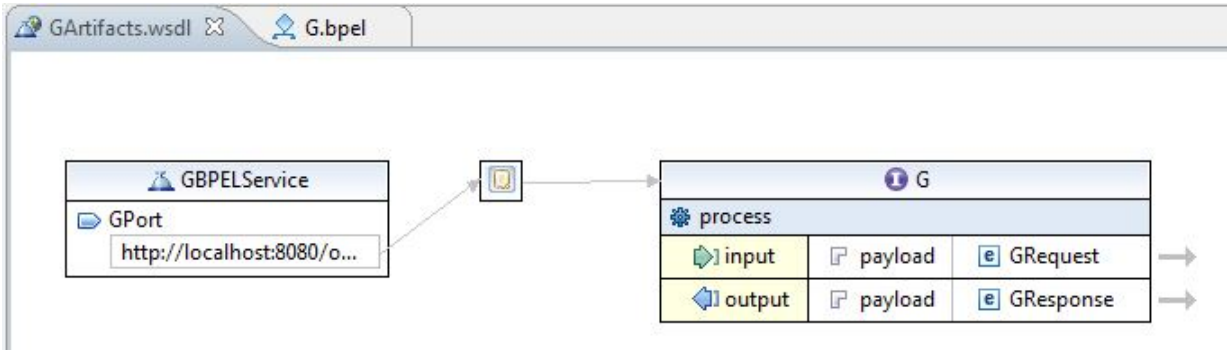

Figure A.13: Setting up the Web Service description file

Add a new process to the project. Name it S1 and set the namespace to “http://www.choreography.com”. Again, choose a synchronous BPEL Process as template. Model the process as shown in figure A.14. Add the activities to the process, assign the variables for the activity “createResponse”, set the time durations for all “Wait” activities and define the time constraints according to the tables beside the activities in figure A.14. Set the deadline for the entire process S1 to the value, specified in figure A.14. Analog to the preparation of the Web Service description file of process G, the Web Service description file “S1Artifacts.wsdl” has to be prepared for process S1. Add a Service, name it to “S1BPELService”. Rename the automatically generated port to “S1Port”, add the address http://localhost:8080/ode/processes/S1PELService and choose the SOAP protocol. Add a new binding, name it to S1Binding, select the PortType “S1” and generate the binding content.

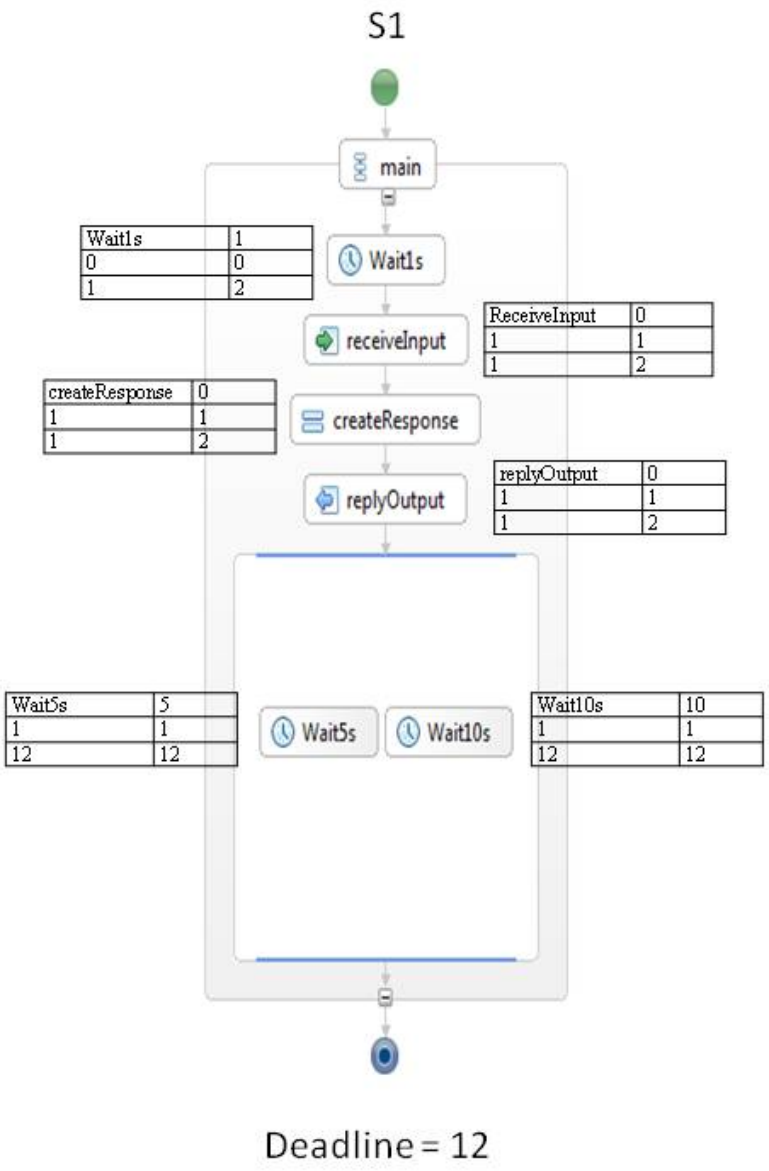

Figure A.14: Process S1 with time durations

Add a new process to the project. Name it S2 and set the namespace to "http://www.choreography.com". Again, choose a synchronous BPEL Process as template. Model the process as shown in figure A.15. Add the activities to the process, assign the variables for the activity "createResponse", set the time durations for all "Wait" activities and define the time constraints according to the tables beside the activities in figure A.15. Set the deadline for the entire process S1 to the value, specified in figure A.15. For the "If & While" activity, use the same conditions like used in process G. Analog to the preparation of the Web Service description file of process G, the Web Service description file "S2Artifacts.wsdl" has to be prepared for process S2. Add a Service, name it to "S2BPELService". Rename the automatically generated port to "S2Port", add the address http://localhost:8080/ode/processes/S2PELService and choose the SOAP protocol. Add a new binding, name it to S2Binding, select the PortType "S2" and generate the binding content.

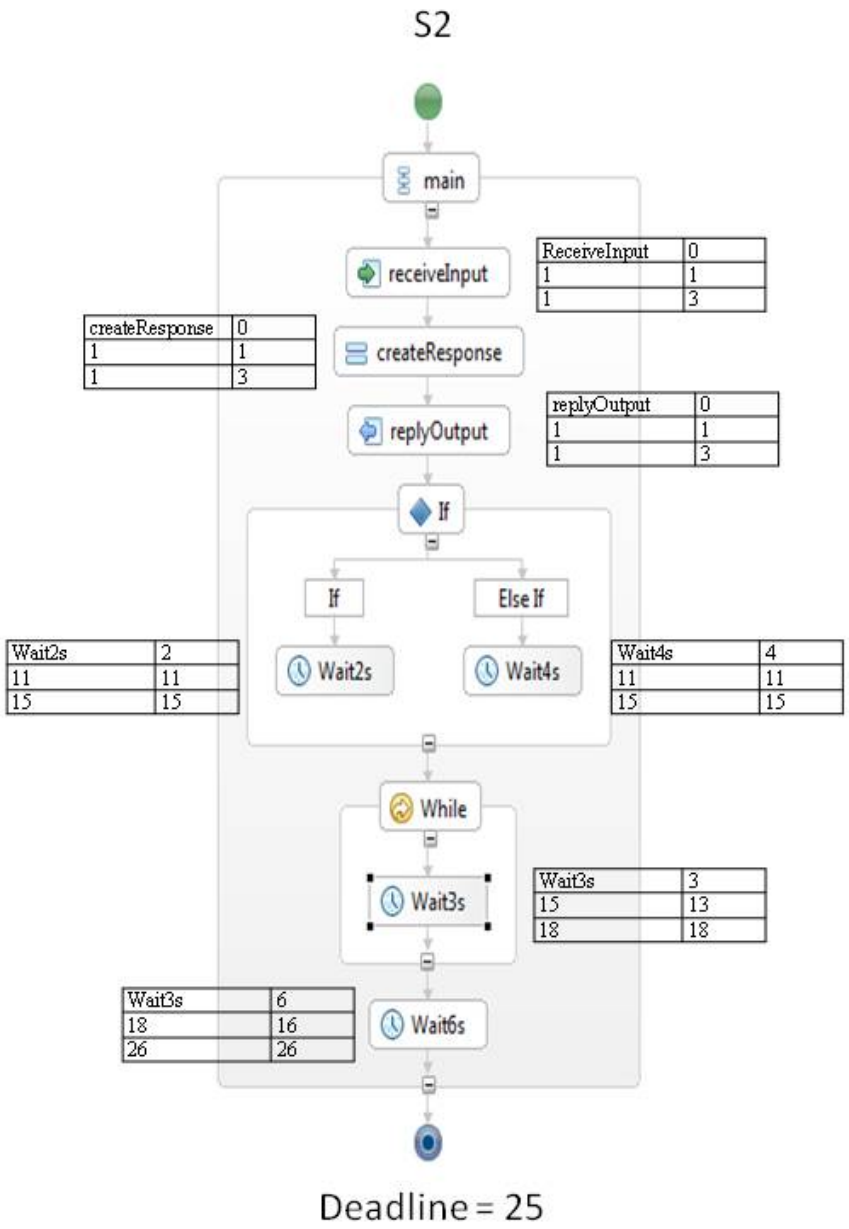

Figure A.15: Process S2 with time durations

A.3.1 Definition of the dependencies of the choreography

The next step is defining the dependencies of the choreography. Process “G” is in that example the supported choreography, “S1” and “S2” are the supporting choreographies. Therefore, select the section “Choreography” in the Eclipse BPEL-Designer. Select “G” as process. In the underlying row, the message “Process[G] is supported by” appears. Select the process “S1” and hit the button “Add”. Select “S2” and hit “Add” again (see figure A.16). The dependencies for the choreography are defined now and after refreshing the project “Choreography”, a new file “dependencies.xml” appears.

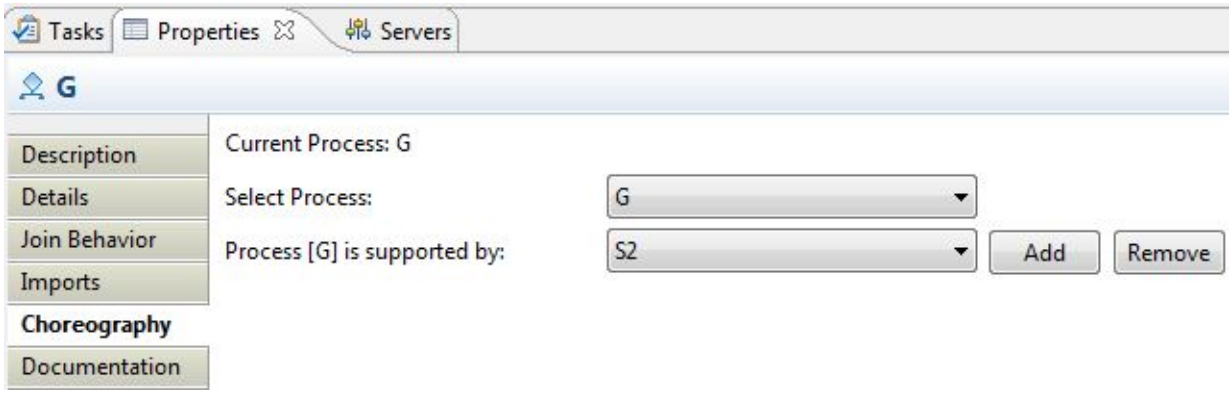

Figure A.16: Defining the dependencies of the choreography

A.3.2 Calculating the time constraints of the choreography

After the definition of the process dependencies the time constraint values can be calculated. Therefore perform a click on the “Constraints” section in the Properties (see figure A.17) and click on the button “calculate”. The “EPS” and “LAE” values for best case and worst case scenario are calculated in consideration of the process dependencies of the choreography.

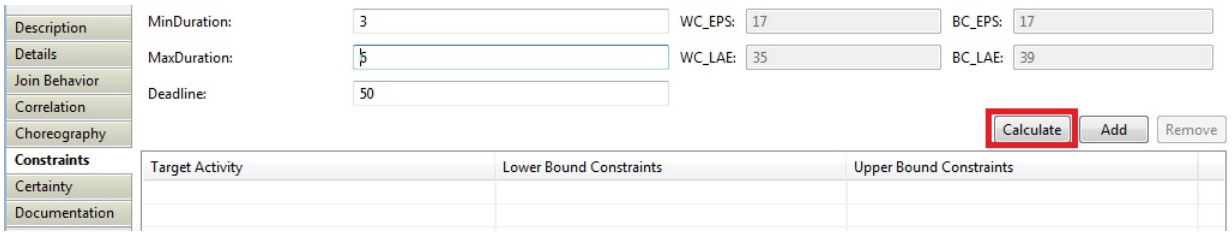

Figure A.17: Calculation of the Choreography

A.3.3 Preparing the choreography for the execution

The next step is the deployment of the generated processes. Perform a right click on the project “Choreography”, select “New/Other...”. Expand the folder “BPEL 2.0”, select “Apache ODE Deployment Descriptor”, click on next and then on finish. A new window appears in the BPEL-Designer. In the section “Inbound Interfaces (Services)”, choose the Associated Port “GPort”, which was specified above in the Web Service description file of process “G”. Figure A.18 shows the correct deployment of process G. In the left bottom of figure A.18, a red rectangle shows the 3 specified processes “G, S1 and S2”. Click on the other processes and perform the deployment as described above. Note, that for “G” the “GPort”, for “S1” the “S1Port” and for “S2” the “S2Port” have to be selected.

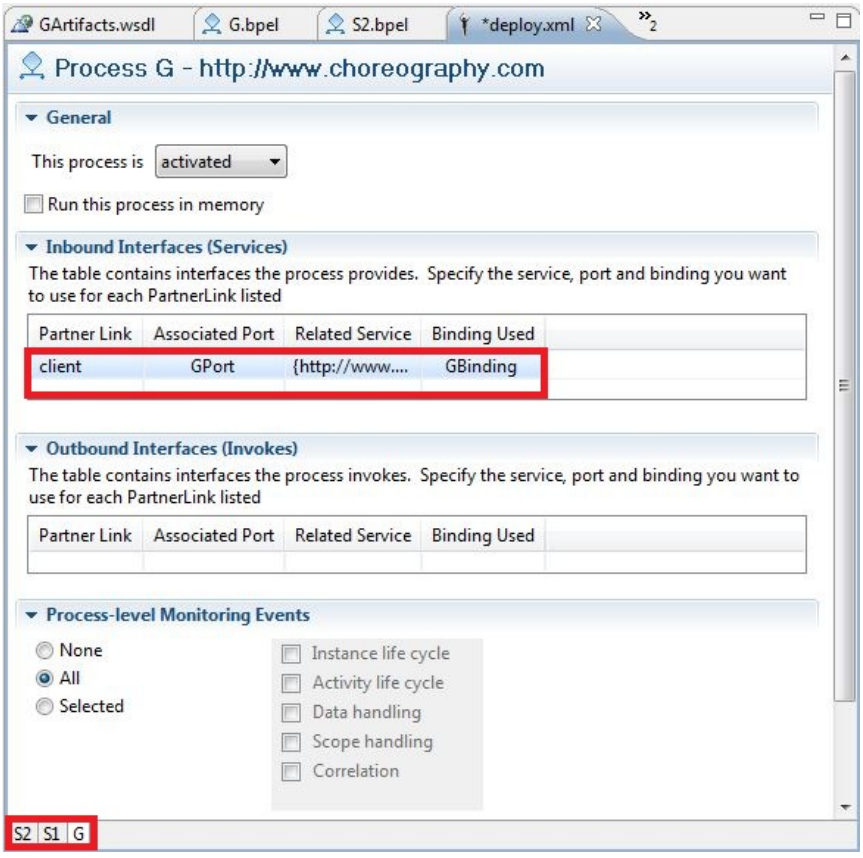

Figure A.18: Deployment of the processes

Finally the processes have to be published in order to start the execution of the processes. Therefore, start the other unpacked Eclipse with the name Eclipse-BPEL-Engine. Point the workspace to the directory, where the BPEL-process was created. Go to the menu bar “Help/Software Updates...”, select the tab “Available Software”, and press the

button “Add Site...”. Copy the link “http://download.eclipse.org/technology/bpel/update-site/” as the location of the site and click on “OK”. Select the “Eclipse BPEL Designer Nightly Build Update Site” as shown in figure A.2. Add the Server view by clicking on the menu bar “Window/Show View/Other...” and select in the folder “Server” the “Servers” entry.

A.4 Execution of the choreography in Apache ODE

Start the Eclipse with the name Eclipse-BPEL-Engine. Point the workspace to the directory, where the BPEL-process was created. In the bottom of the Eclipse, the view “Servers” appear. Click on “Servers” and perform a right click in the “Servers” view. Select “New/Servers”. In the list, expand the “Apache” folder, select “Ode v1.x Server” and go to next. In the appearing dialog, the path to the ODE’s home directory and to Tomcat’s home directory have to be set. Point the ODE home directory to “PathToApacheTomcat/webapps/ode” and the Tomcat’s home directory to “PathToApacheTomcat” as shown in figure A.19.

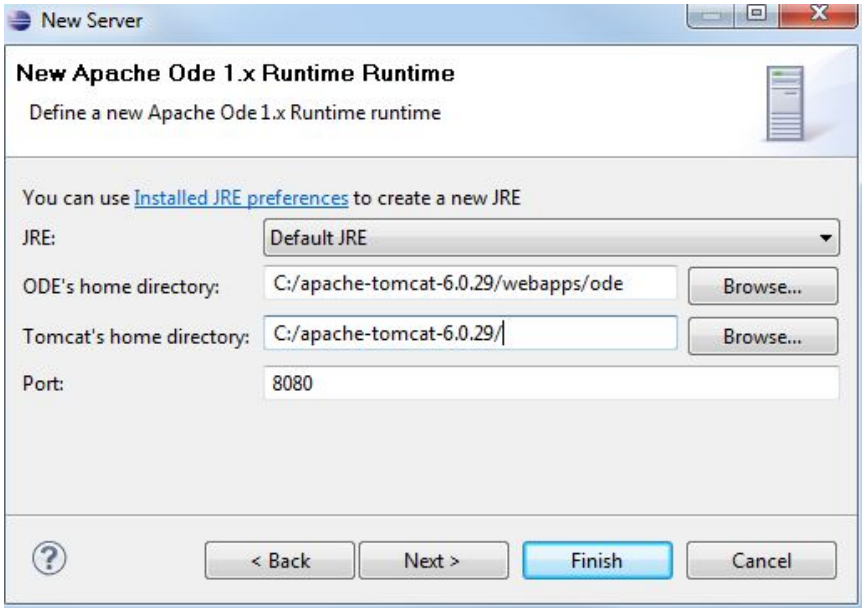

Figure A.19: Integrating the Apache ODE Server in Eclipse-Designer

After clicking on “Finish”, the integration of Apache ODE in the Eclipse is finished. In the “Server’s” view, the integrated ODE Server appears. Click on the property view “Servers”, perform a right click on the existing ODE Server and select “Add and Remove”.

Select the processes “G, S1 and S2” and click on “Add >”. The windows should look similar to the figure A.20. Click on “Finish” to add the processes to the Apache ODE Server.

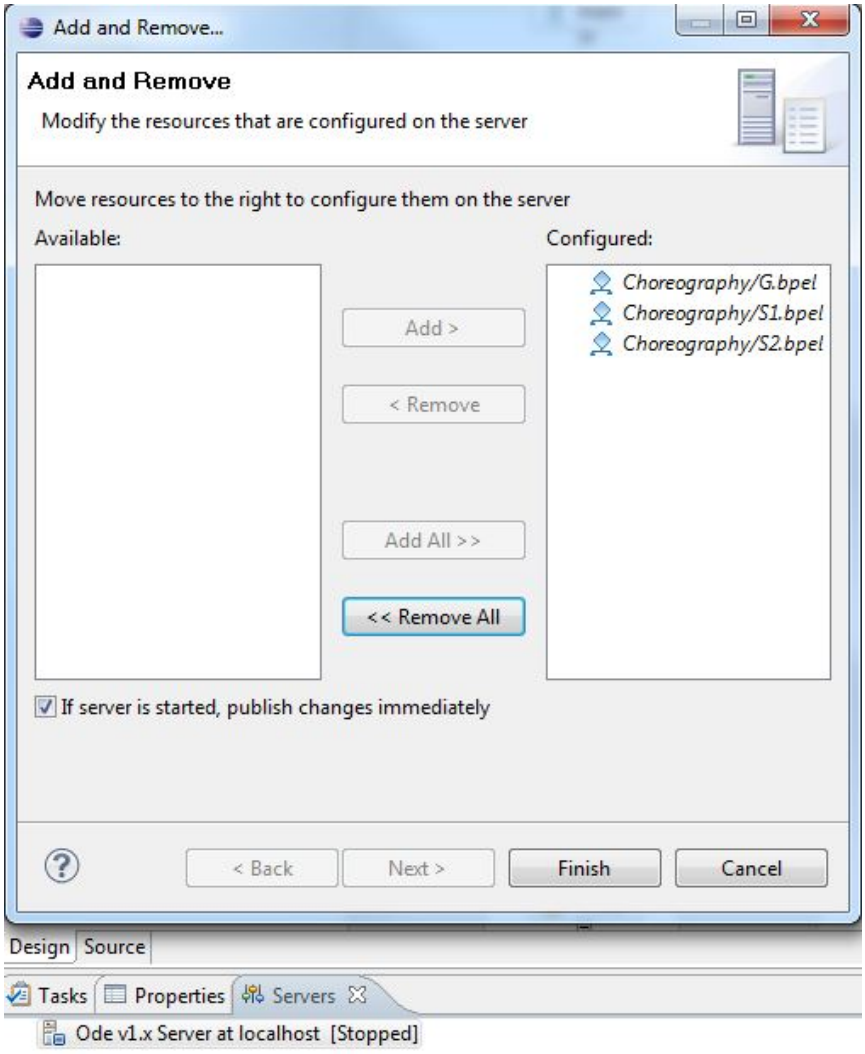

Figure A.20: Add the processes to the ODE Server

Perform a right click on the ODE Server and click “Publish”. The publishing is finished, after the following output in the console appears:

13:16:07,509 INFO [CronScheduler] Cancelling PROCESS CRON jobs for: http://www.choreography.comS1-1
13:16:07,509 INFO [CronScheduler] Scheduling PROCESS CRON jobs for: http://www.choreography.comS1-1
13:16:07,509 INFO [CronScheduler] Cancelling PROCESS CRON jobs for:

http://www.choreography.comG-1
13:16:07,510 INFO [CronScheduler] Scheduling PROCESS CRON jobs for:
http://www.choreography.comG-1
13:16:07,510 INFO [CronScheduler] Cancelling PROCESS CRON jobs for:
http://www.choreography.comS2-1
13:16:07,510 INFO [CronScheduler] Scheduling PROCESS CRON jobs for:
http://www.choreography.comS2-1

A.4.1 Instantiation of a process

For instantiating a process (for example process "G"), perform a right click on the Web Service description file "GProcessArtifacts.wsdl" and click "Web Services/Test with Web Services Explorer". In the appearing window, expand all list items in the navigator and click on "process". On the right side, enter a message in the blank text field and click on "Go" (see figure A.21). The process initiates and the traffic light appears to check the calculated durations at design time.

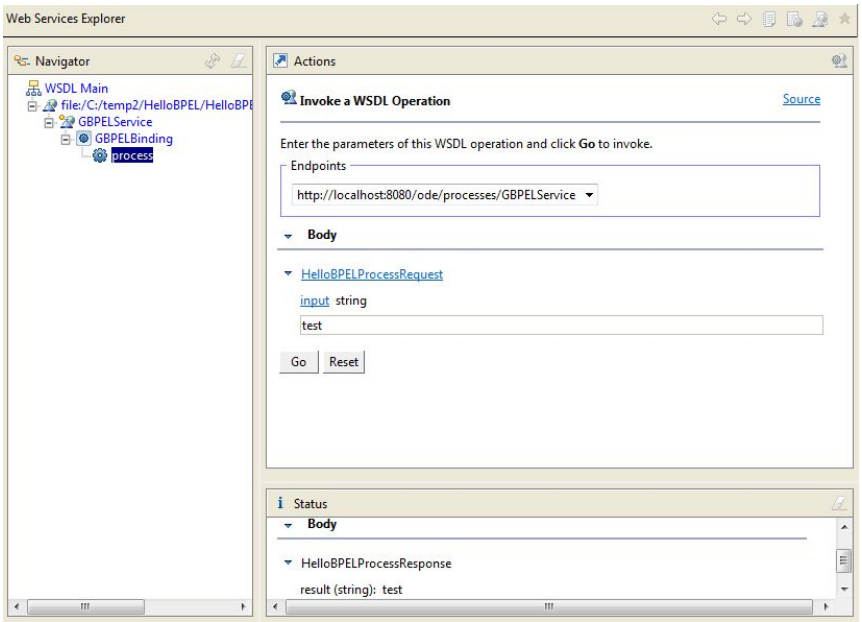

Figure A.21: Using the Web Services Explorer of Eclipse

List of Algorithms

List of Tables

List of Figures

Bibliography

[1] H. Gehring, A. Gadatsch, "Ein Rahmenkonzept für die Modellierung von Geschäftsprozessen und Workflows," tech. rep., FernUniversität Hagen, 1999.

[2] A. Tahamtan, *Modeling and Verification of Web Service Composition Based Interorganizational Workflows*. PhD thesis, Universitiy of Vienna, 2009.

[3] H. Kerzner, *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. New York, NY, USA: John Wiley & Sons, Inc., 2009.

[4] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[5] L. R. Planken, "Temporal reasoning problems and algorithms for solving them (literature survey)," literature survey, Delft University of Technology, October 2007.

[6] H. Zhuge, T.-y. Cheung„ H.-K. Pung, "A timed workflow process model," *J. Syst. Softw.*, vol. 55, no. 3, pp. 231–243, 2001.

[7] H. Li, Y. Yang, "Dynamic checking of temporal constraints for concurrent workflows," *Electron. Commer. Rec. Appl.*, vol. 4, no. 2, pp. 124–142, 2005.

[8] R. Dechter, I. Meiri„ J. Pearl, "Temporal constraint networks," *Artif. Intell.*, vol. 49, no. 1-3, pp. 61–95, 1991.

[9] K. Stergiou, M. Koubarakis, "Backtracking algorithms for disjunctions of temporal constraints," in *AAAI '98/IAAI '98: Proceedings of the fifteenth national/tenth conference on Artificial intelligence/Innovative applications of artificial intelligence*, (Menlo Park, CA, USA), pp. 248–253, American Association for Artificial Intelligence, 1998.

[10] I. Tsamardinos, M. E. Pollack, "Efficient solution techniques for disjunctive temporal reasoning problems," *Artif. Intell.*, vol. 151, no. 1-2, pp. 43–89, 2003.

[11] Y. Liu, H. Qian,, Y. Jiang, "Graph-dtp: Graph-based algorithm for solving disjunctive temporal problems," in *TIME '07: Proceedings of the 14th International Symposium on Temporal Representation and Reasoning*, (Washington, DC, USA), p. 190, IEEE Computer Society, 2007.

[12] N. R. Sahkar, V. Sireesha, "Using modified dijkstra's algorithm for critical path method in a project network," *International Journal of Computational and Applied Mathematics*, vol. 5, no. 2, pp. 217–225, 2010.

[13] N. Thumb, *Grundlagen und Praxis der Netzplantechnik*. Verlag Moderne Industrie, 1975.

[14] J. Kerbosch, H. Schell, "Network planning by the extended metra potential method," tech. rep., University of Technology Eindhoven, 1975.

[15] H. Wieczorrek, P. Mertens, *Management von IT-Projekten. Von der Planung zur Realisierung*. Springer, 2005.

[16] H. S. Swanson, R. E. D. Woolsey, "A pert-cpm tutorial," *SIGMAP Bull.*, no. 16, pp. 54–62, 1974.

[17] R. Berbig, F. Franke, *Netzplantechnik*. VEB Verlag für Bauwesen, 1969.

[18] D. E. Douglas, "Pert and simulation," in *WSC '78: Proceedings of the 10th conference on Winter simulation*, (Piscataway, NJ, USA), pp. 89–98, IEEE Press, 1978.

[19] J. E. Hebert, "Applications of simulation in project management," in *WSC '79: Proceedings of the 11th conference on Winter simulation*, (Piscataway, NJ, USA), pp. 211–219, IEEE Press, 1979.

[20] D. Liebhart, *SOA goes real. Service-orientierte Architekturen erfolgreich planen und einführen*. Hanser, 2007.

[21] T. Vogel, *Servicebasiertes Business Networking*. PhD thesis, University of St. Gallen, 2009.

[22] R. Heutschi, *Serviceorientierte Architektur. Architekturprinzipien und Umsetzung in die Praxis*. Springer, 2007.

[23] H. Haas, A. Brown, "Web services glossary." http://www.w3.org/TR/ws-gloss/, 2004.

[24] V. Vasudevan, "A Web Services Primer," tech. rep., University of Konstanz, 2001.

[25] S. Dustdar, H. Gall„ M. Hauswirth, *Software-Architekturen für Verteilte Systeme*. Springer, 2003.

[26] "Oasis." http://www.oasis-open.org/who/.

[27] F. Leymann, "Web services flow language (wsfl 1.0)," tech. rep., IBM, 2001.

[28] S. Thatte, "Web services for business process design," tech. rep., Microsoft, 2001.

[29] F. Leymann, D. Roller„ S. Thatte, "Goals of the bpel4ws specification." http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf.

[30] ORACLE, "Building stateless and stateful business processes." http://download.oracle.com/docs/cd/E14981_01/wli/docs1031/bpguide-/bpguideState.html, 2008.

[31] C. Peltz, "Web services orchestration and choreography." http://soa.sys-con.com/node/39800, 2003.

[32] D. Jordan, J. Evdemon, "Web services business process execution language version 2.0." http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html, 2007.

[33] M. Cobban, "What is bpel and why is it so important to my business?." http://www.softcare.com/whitepapers/wp_whatis_bpel.php, 2004.

[34] J. Matlis, "Quickstudy: Business process execution language (bpel)." http://www.computerworld.com/s/article/102580/BPEL?taxonomyId=061, 2005.

[35] O. Kopp, F. Leymann, "Choreography Design Using WS-BPEL," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 31, pp. 31–34, September 2008.

[36] K. R. Moorthy, "An introduction to bpel." http://www.developer.com/services-/article.php/3609381/An-Introduction-to-BPEL.htm, 2006.

[37] M. Juric, "Bpel and java." http://www.theserverside.com/news/1364554/BPEL-and-Java, 2005.

[38] C. Bettini, X. S. Wang„ S. Jajodia, "Temporal reasoning in workflow systems," *Distrib. Parallel Databases*, vol. 11, pp. 269–306, May 2002.

[39] W. Gruber, *Modeling and Transformation of Workflows with Temporal Constraints*. PhD thesis, Universitiy of Klagenfurt, 2004.

[40] V. D. Aalst, "The application of petri nets to workflow management," *The Journal of Circuits, Systems and Computers*, vol. 8, pp. 21–66, 1998.

[41] D. Wodtke, G. Weikum, "A formal foundation for distributed workflow execution based on state charts," in *Proceedings of the 6th International Conference on Database Theory*, (London, UK), pp. 230–246, Springer-Verlag, 1997.

[42] J. Eder, H. Groiss„ W. Liebhart, "The workflow management system panta rhei," *Advances in Workflow Management Systems and Interoperability*, pp. 129–144, 1997.

[43] B. Kao, H. Garcia-Molina, "Deadline assignment in a distributed soft real-time system," *IEEE Trans. Parallel Distrib. Syst.*, vol. 8, pp. 1268–1274, December 1997.

[44] B. Kao, H. Garcia-Molina, "Subtask deadline assignment for complex distributed soft real-time tasks," tech. rep., Stanford University, Stanford, CA, USA, 1993.

[45] P. Dadam, M. Reichert„ K. Kuhn, "Clinical workflows - the killer application for process-oriented information systems?," in *4th International Conference on Business Information Systems (BIS 2000*, pp. 36–59, 1997.

[46] O. Marjanovic, "Dynamic verification of temporal constraints in production workflows," in *Proceedings of the Australasian Database Conference*, ADC '00, (Washington, DC, USA), pp. 74–, IEEE Computer Society, 2000.

[47] O. Marjanovic, M. Orlowska, "On modeling and verification of temporal constraints in production workflows," in *Knowledge and Information Systems*, pp. 157–192, 1999.

[48] C. Bussler, "Workflow instance scheduling with project management tools," in *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, DEXA '98, (Washington, DC, USA), pp. 753–, IEEE Computer Society, 1998.

[49] H. Pozewaunig, J. Eder„ W. Liebhart, "epert: Extending pert for workflow management systems," in *In First European Symposium in Advances in Databases and Information Systems (ADBIS*, pp. 217–224, 1997.

[50] R. Kazhamiakin, P. Pandya„ M. Pistore, "Timed modelling and analysis inweb service compositions," in *ARES '06: Proceedings of the First International Conference on*

Availability, Reliability and Security, (Washington, DC, USA), pp. 840–846, IEEE Computer Society, 2006.

[51] R. Kazhamiakin, P. Pandya„ M. Pistore, "Representation, verification, and computation of timed properties in web," in *ICWS '06: Proceedings of the IEEE International Conference on Web Services*, (Washington, DC, USA), pp. 497–504, IEEE Computer Society, 2006.

[52] P. K. Pandya, "Specifying and deciding quantified discrete-time duration calculus formulae using dcvalid," tech. rep., Proc. Real-Time Tools, 2000.

[53] C. Hoare, A. Ravn, "A calculus of durations," tech. rep., Information Processing Letters, 1991.

[54] S. Kallel, A. Charfi, T. Dinkelaker, M. Mezini„ M. Jmaiel, "Specifying and monitoring temporal properties in web services compositions," in *ECOWS '09: Proceedings of the 2009 Seventh IEEE European Conference on Web Services*, (Washington, DC, USA), pp. 148–157, IEEE Computer Society, 2009.

[55] A. Charfi, M. Mezini, "Ao4bpel: An aspect-oriented extension to bpel," *World Wide Web*, vol. 10, no. 3, pp. 309–344, 2007.

[56] N. Guermouche, O. Perrin„ C. Ringeissen, "Timed specification for web services compatibility analysis," *Electron. Notes Theor. Comput. Sci.*, vol. 200, no. 3, pp. 155–170, 2008.

[57] D. Berardi, *Automatic Service Composition. Models, Techniques and Tools.* PhD thesis, La Sapienza University Roma, 2005.

[58] D. Berardi, D. Calvanese„ G. D. Giacomo, "Automatic composition of e-services that export their behavior," tech. rep., University of Rome, 2003.

[59] J. Eder, E. Panagos„ M. Rabinovich, "Time constraints in workflow systems," *Proceedings of the 11th International Conference of Advanced information Systems Engineering*, pp. 286 – 300, 1999.

[60] J. Eder, E. Panagos, "Managing time in workflow systems," *Workflow Handbook 2001*, pp. 109–132, 2000.

[61] J. Eder, H. Pichler, "Duration histograms for workflow systems," in *Proceedings of the IFIP TC8 / WG8.1 Working Conference on Engineering Information Systems in the Internet Context*, (Deventer, The Netherlands, The Netherlands), pp. 239–253, Kluwer, B.V., 2002.

[62] Apache, "Apache ODE." http://ode.apache.org/.

[63] "Orchestra." http://orchestra.ow2.org/xwiki/bin/view/Main/WebHome.