

The contracting agent : concepts and architecture of a generic software component for electronic business based on outsourcing of work

Citation for published version (APA):

Dijk, van, A. (2001). *The contracting agent : concepts and architecture of a generic software component for electronic business based on outsourcing of work*. [Phd Thesis 2 (Research NOT TU/e / Graduation TU/e), Mathematics and Computer Science]. Technische Universiteit Eindhoven. <https://doi.org/10.6100/IR549629>

DOI:

[10.6100/IR549629](https://doi.org/10.6100/IR549629)

Document status and date:

Published: 01/01/2001

Document Version:

Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

- A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
- The final author version and the galley proof are versions of the publication after peer review.
- The final published version features the final layout of the paper including the volume, issue and page numbers.

[Link to publication](#)

General rights

Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy

If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

The Contracting Agent

*concepts and architecture of a generic software component
for electronic business based on outsourcing of work*

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Dijk, Andries van

The Contracting Agent: concepts and architecture of a generic software component for electronic business based on outsourcing of work / by Andries van Dijk. - Eindhoven : Technische Universiteit Eindhoven, 2001. Proefontwerp. - ISBN 90-386-0941-8

NUGI 852

Subject headings : software design / electronic commerce / Petri nets

CR Subject Classification (1998) : K.4.4, D.2.11

Cover design by Paul Verspaget

Printed by Eindhoven University Press Facilities

© 2001, A. van Dijk

Alle rechten voorbehouden. Uit deze uitgave mag niet worden gereproduceerd door middel van boekdruk, fotokopie, microfilm of welk ander medium dan ook, zonder voorafgaande schriftelijke toestemming van de auteur.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form by any means, mechanical, photocopying, recording, or otherwise, without the prior written consent of the author.

The Contracting Agent

*concepts and architecture of a generic software component
for electronic business based on outsourcing of work*

Proefontwerp

ter verkrijging van de graad van doctor aan de Technische
Universiteit Eindhoven, op gezag van de Rector Magnificus,
prof.dr. R.A. van Santen, voor een commissie aangewezen door het
College voor Promoties in het openbaar te verdedigen op
maandag 19 november 2001 om 16.00 uur

door

Andries van Dijk

geboren te Giessenburg

De documentatie van het proefontwerp is goedgekeurd door de promotoren:

prof.dr. K.M. van Hee
en
prof.dr. P.M.E. de Bra

Preface

The work on this dissertation started in August 1997 with a presentation of initial ideas at the Technical University Eindhoven and ended four years later in August 2001. During these four years I have developed the initial ideas into the results as they are today. The process brought moments of excitement as well as moments of disappointment, but was always interesting. Often, the results have grown at a speed much slower than I liked. Sometimes, I have followed an approach that did not bring the results I wanted. However, the result is a contribution to the theory and practice of electronic contracting and a basis for further research.

This dissertation could not exist without the support of many people. First of all I thank my supervisors Kees van Hee and Paul de Bra for their support during the entire period. Your suggestions and ideas were very important to me and guided me towards the results as they are now. When looking at some of the first versions of this dissertation I admire your confidence in a successful end. I also want to thank Wil van der Aalst and Ron Lee for their important contribution in the last months. By asking me the right questions and by giving me the right suggestions, I have been able to increase the quality of the dissertation significantly.

Second, I want to thank Deloitte & Touche Bakkenist for giving me the opportunity, the time and a stimulating environment to work on my dissertation. Without it this dissertation would not have come to life. I thank my colleagues from the ICT Strategy & Architecture group for sharing their knowledge and experience. Especially I want to thank Wout Hofman for working together in many projects in the area of electronic business transactions. Also thanks to my colleagues in Dordrecht for their patience at those times they had to wait at the printer when I was printing yet another version of this dissertation.

During the four years I have worked on this dissertation many people belonging to my relatives and friends have shown interest in the status of my work, especially during the last one and a half year when it was 'nearly' finished. When you asked me when I planned to finish my dissertation, I often could not give you the answer I would have liked to give. However, most of the times, your concern inspired me to continue and I want to thank you all for it. Furthermore, I want to thank my parents for always stimulating me to have a good education, which has been the basis for the work I have now finished.

Finally, I want to thank my wife and daughters, Heleen, Rachel and Loïs for their continuous support. Writing a dissertation has a price, which often had to be paid by the three of you. Though physically present, you have found me mentally absent for quite a couple of times, especially in the last year. Rachel and Loïs, you do not even remember a time when I was not upstairs and 'writing a book'. I owe the three of you a lot and hope and expect that you will see a difference. This work is dedicated to you.

Hendrik-Ido-Ambacht,
August, 2001

Table of contents

Preface	v
Table of contents	vii
Summary	xi
1. Introduction	15
1.1 Motivation	15
1.2 Electronic commerce	18
1.2.1 Introduction	18
1.2.2 Internet as enabling technology	18
1.2.3 Changes to the business	21
1.3 Industry solutions for electronic contracting.....	23
1.3.1 Electronic data interchange.....	23
1.3.2 XML based messaging frameworks.....	24
1.3.3 Workflow interoperability.....	26
1.4 Scientific research on electronic contracting.....	30
1.4.1 Definition	30
1.4.2 The Language / Action Perspective	30
1.4.3 Documentary Petri Nets	31
1.4.4 Interorganisational workflows	33
1.4.5 Mobile agents.....	34
1.4.6 Other related projects	37
1.5 Software engineering.....	39
1.5.1 Specification.....	39
1.5.2 Construction	40
1.5.3 Component based software engineering	41
1.6 Research	43
1.6.1 Research problem and objectives.....	43
1.6.2 Research scope	44
1.6.3 Research approach	45
1.6.4 Research questions	45
1.7 Outline.....	47

2.	A conceptual framework for service contracting	49
2.1	Inter-organisational systems.....	49
2.2	Underpinning concepts.....	50
2.2.1	Workflows.....	50
2.2.2	Services.....	55
2.2.3	Transactions.....	58
2.3	Basic service contracting concepts.....	65
2.3.1	Definitions.....	65
2.3.2	Frameworks.....	67
2.3.3	Structure of the rest of this chapter.....	70
2.4	Specification of the interface agreements.....	72
2.4.1	Data model.....	72
2.4.2	Transaction protocol patterns.....	76
2.5	Specification of contracting requirements.....	88
2.5.1	Definitions.....	88
2.5.2	A data model for contracting requirements.....	90
2.6	Construction of the contracting workflow.....	96
2.6.1	Basic operations in service contracting processes.....	96
2.6.2	The contracting workflow.....	104
2.6.3	Standard transitions for the contracting workflow.....	106
2.6.4	The ‘negotiation’ transition.....	113
2.6.5	The ‘execution’ transition.....	121
2.6.6	The ‘acceptance’ transition.....	125
2.6.7	Composing the contracting workflow.....	127
2.7	Use case ‘business trip’.....	132
2.7.1	Introduction.....	132
2.7.2	Case type ‘Trip’.....	132
2.7.3	Service type ‘Book flight’.....	133
2.7.4	Service type ‘Cancel flight’.....	135
2.7.5	Service type ‘Book hotel’.....	136
2.7.6	Service type ‘Book rental car’.....	138
2.7.7	Service providers.....	140
2.7.8	Contracting requirements.....	141
3.	Logical architecture of the Contracting Agent	149
3.1	Introduction.....	149
3.1.1	Relation to the conceptual framework.....	149
3.1.2	Definition of the term architecture.....	150
3.1.3	Design goals.....	151
3.2	Architecture of the ‘Contracting Agent’ component.....	153
3.2.1	Distribution of functionality over components.....	153
3.2.2	Structure of interfaces.....	154
3.2.3	Behaviour on the component interfaces.....	164
3.3	Architecture of the ‘Server’ component.....	164
3.3.1	Distribution of functionality over components.....	164
3.3.2	Structure of interfaces and persistent data.....	167
3.3.3	Behaviour on the component interfaces.....	183

3.4	Architecture of the ‘Configurator’ component.....	187
3.4.1	Distribution of functionality over components.....	187
3.4.2	Structure of interfaces and persistent data.....	189
3.4.3	Behaviour on the component interfaces.....	197
4.	Technical architecture of the Contracting Agent	199
4.1	Basic construction choices.....	199
4.1.1	Objectives.....	199
4.1.2	Windows and COM.....	200
4.1.3	XML, XML-Schema, XSLT, DOM, MSXML4.....	202
4.1.4	Relational databases, ADO and Access 2000 Jet engine.....	203
4.1.5	ExSpect as workflow engine.....	205
4.1.6	WOFLAN 2.0 as Workflow net analyser.....	206
4.1.7	MS Outlook as message exchange component.....	207
4.2	Application architecture.....	208
4.3	Relational databases.....	211
4.3.1	Relational database ‘CaConfC.mdb’.....	211
4.3.2	Relational database ‘CaState.mdb’.....	214
4.3.3	Relational database ‘CaConfS.mdb’.....	216
4.4	COM components.....	217
4.4.1	Out-of-process COM server ‘CaIntMan.exe’.....	217
4.4.2	Stand alone executable program ‘CaConfig.exe’.....	220
4.4.3	Stand alone executable program ‘CaMonitor.exe’.....	221
4.5	Structure of the generated ExSpect model.....	222
4.5.1	Hierarchic level 1: the main system.....	222
4.5.2	Hierarchic level 2: case type.....	224
4.5.3	Hierarchic level 3: candidate service type.....	224
4.5.4	Hierarchic level 4: contracting phase.....	226
4.6	Examples of the user interfaces.....	228
4.6.1	Defining available services (repository).....	228
4.6.2	Defining contracting requirements.....	236
4.6.3	Configuring the Server component.....	242
4.6.4	Monitoring the Server component.....	242
5.	Evaluation	247
5.1	The research problem in retrospective.....	247
5.2	Achievements.....	248
5.3	Contribution.....	250
5.4	Business opportunities.....	251
5.5	Conclusions.....	252
5.6	Directions for future research.....	253
5.7	Concluding remark.....	255
A.	Modelling techniques	256
A.1	Functional data modelling.....	256
A.2	High level coloured Petri nets.....	259
A.3	EBNF.....	261

References	263
Abbreviations and acronyms	273
Samenvatting (Dutch)	275

Summary

Business processes are seldom confined to the boundaries of a single organisation. Instead, there is an increasing tendency towards inter-organisational business processes for which several mechanisms can be used among which: information sharing, capacity sharing, case transfer and contracting. This research addresses inter-organisational business processes based on electronic contracting. Although a distinction between ‘products’ and ‘services’ is often made, we will use the term ‘service’ as a synonym for both.

Scope

The term ‘electronic contracting’ is used for a variety of phenomena. This research is focused on a specific part of this area, which is demarcated by the following characteristics. First, we focus on electronic contracting processes in an environment with *loosely coupled* participants, i.e. all communication is performed via business transactions consisting of structured messages of which the static aspects (data semantics and data syntax) as well as the dynamic aspects (allowed sequence of messages) are mutually agreed. We do not assume a central brokerage or mediation service between the participants, nor do we assume parties to have knowledge of each others internal business processes. Furthermore, although the issue of establishing trust between parties that have no prior relationship is very important, it is outside the scope of this research.

A contracting process involves at least two parties; one in the role of buyer and one in the role of seller, each with its distinctive type of actions. Our research addresses the actions performed by the *buyer* only. The term ‘service contracting’ will therefore be used for the following activities performed by a service client to contract a service from a service provider:

- *specify* the details of the required service;
- *negotiate* a commitment from a service provider;
- *monitor* the execution of the service;
- *accept* the result.

When we look at the type of contracting processes, we focus on complex processes where each business case requires N different services, each of which can be contracted from M service providers. For these N required services, we assume *constraints* on the order in which they must be contracted and *dependencies* between their details. For example, when a flight to New York and a rental car at the airport of destination must be contracted, the contracting of the rental car can not start until the flight is booked and the airport of destination is known. Furthermore, we assume that parties are autonomous and have at most *partial knowledge* of each others available resources. Therefore, a service client can not simply assign the execution of a service to a service provider, but has to negotiate instead. Finally, our research focuses on *completely automated* contracting processes that require a highly structured and computer interpretable specification of the contracting process.

Approach

The approach followed in this research is illustrated in Figure 1. The left part of the figure illustrates the conceptual level of the research and the right part illustrates the technical (software infrastructure) level. At the conceptual level, we consider the *internal workflow* of an organisation that contains one or more *outsourced tasks*, i.e. tasks that are not executed by internal resources. The execution of an outsourced task requires one or more services to be contracted from external service providers. Communication with these external service providers is performed via business transactions consisting of structured messages, of which the static and dynamic aspects are defined in *transaction protocols*. The activities to negotiate and monitor contracts for the required services is modelled in a separate workflow, the *contracting workflow*. The specification of contracting requirements and the construction of the contracting workflow is the subject of this research. In addition, the research also focuses on the implementation of the contracting workflow. The objective of the research is to design and develop a separate software component, the *Contracting Agent*, to which the *internal information system* delegates the execution of the entire service contracting process as defined by the contracting workflow. The Contracting Agent uses standard software for *inter-organisational message exchange* for the operations required to send and receive messages to and from external parties (i.e. conversion, authentication, communication, etc.).

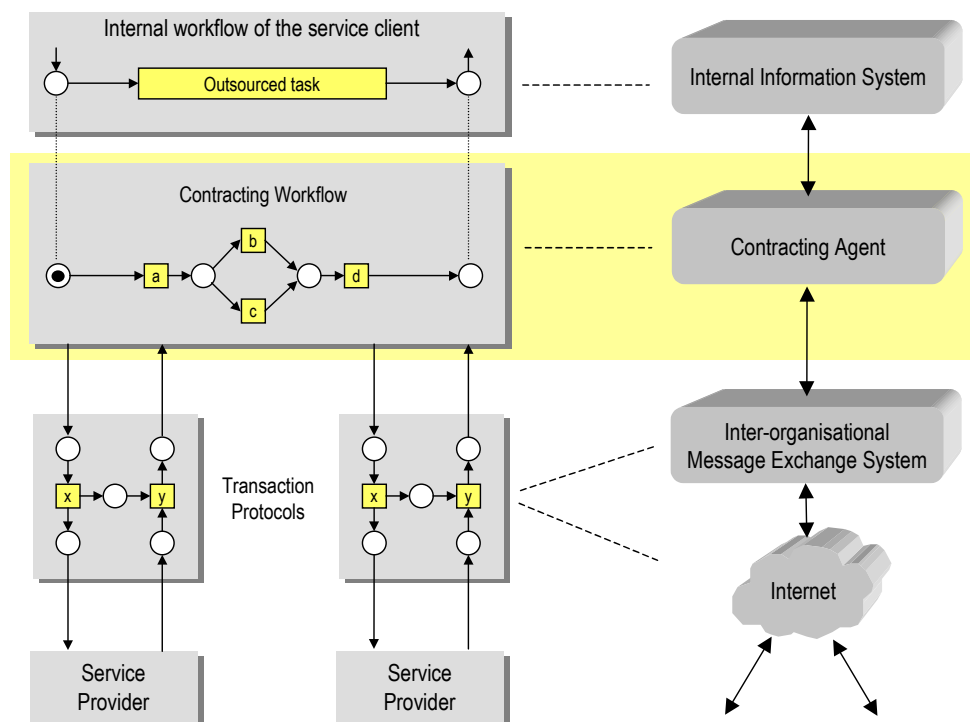


Figure 1 Illustration of the research approach

Objective and claim

The business objective of this research is to contribute to the efficiency of organisations by providing an optimal support for the demarcated class of service contracting processes with information and communication technology. The claim of this research is to have provided:

- the *explication* of the demarcated area of electronic contracting by providing a conceptual framework;
- a *specification language* for contracting requirements of outsourced tasks;

- a set of *standard transitions* from which contracting workflows can be composed;
- a mechanism by which sound contracting workflows can be *generated* from contracting requirements;
- an *architecture* (logical and technical) of a software component that supports the demarcated class of service contracting processes;
- a *proof of concept* of the conceptual framework, the specification language and the architecture in the form a working software component.

Results

The results of the research consist of (i) a conceptual framework for the class of service contracting processes, (ii) a logical architecture for the Contracting Agent, (iii) a technical architecture for the Contracting Agent and (iv) an implementation of the Contracting Agent.

• Conceptual framework (Chapter 2)

The conceptual framework starts with a definition of underpinning concepts like ‘workflows’, ‘services’ and ‘business transactions’. There after, we introduce the term ‘service contracting’ and discuss frameworks for contracting processes found in literature: Action Workflow, DEMO and BAT. A synthesis of these frameworks leads us to the distinction of four consecutive phases in service contracting processes: specification, negotiation, execution and acceptance. Here after, we define the concepts required to describe the necessary *interface agreements* between parties involved in contracting processes: ‘service type’, ‘service provider’, ‘transaction protocol’ and ‘message type’. Furthermore, for each contracting phase that involves message exchange with service providers (negotiation, execution and acceptance), we define a number of *patterns* for the transaction protocol during that phase. After having defined the interface agreements between service client and service providers, we focus on the specification of *contracting requirements* for outsourced tasks. These contracting requirements define *which* services must be contracted for an outsourced task and *how* these services must be contracted. Important parts of the contracting requirements are *specification rules* to define the details of a required service, *triggering mechanisms* that define the order in which required services are contracted and *contracting strategies* that define the behaviour of the service client within the restrictions of the transaction protocol. With the specification of the interface agreements with service providers on one hand and the specification of the contracting requirements for outsourced tasks on the other hand, we have the basis for defining the contracting workflow. First, we define the *state data* of service contracting processes, a set of *standard operations* on the state data and the *configuration parameters* used by these standard operations. After having defined this, we consider the contracting workflow as a high level coloured Petri net that defines the control flow of service contracting processes and in which standard operations on the state data are invoked. We propose a set of *standard transitions* from which contracting workflows can be assembled. We use these standard transitions to create *composite transitions* that implement an entire negotiation strategy, execution strategy or acceptance strategy. Here after, we address the rules according to which an entire contracting workflow can be constructed from the contracting requirements on one hand and the standard transitions on the other hand. Finally, the conceptual framework is illustrated by a use case that involves service contracting for a business trip: inbound flight, outbound flight, hotel and rental car.

- **Logical architecture (Chapter 3)**

The logical architecture identifies the sub-components of which the Contracting Agent consists with their structure and behaviour. The structure of the persistent data and the data exchanged on the component's interfaces is defined via functional data models. The behaviour of each component on its interfaces and the collaboration of the components via their interfaces is defined by using the modelling technique of high level coloured Petri nets. On the highest level, the logical architecture defines three major components: Server, Configurator and Monitor. The Server component is designed according to workflow management principles: separation of execution and control. Persistent data is stored in a relational database, standard operations on the persistent data are implemented in a number of smaller applications, and the control flow of the service contracting process is implemented in a workflow definition enacted by a workflow engine, from which the standard operations are invoked in the right order and with the right parameters. The Configuration component offers a repository function to store interchange agreements with service providers. Furthermore, it supports the user in defining the contracting requirements for outsourced tasks. With these contracting requirements as input, the Configuration component generates the entire workflow definition used in the Server component together with all configuration parameters used by the standard operations on the state data invoked by the contracting workflow. Finally, the Monitor component queries the state data of the Server component and presents it to the user via a graphical user interface.

- **Technical architecture (Chapter 4)**

In this part we translate the logical architecture to a technical architecture with the objective to create a working prototype of the Contracting Agent component. The prototype is used as proof of concept for the conceptual framework and the logical architecture. Therefore, we have an emphasis on functionality rather than on aspects like performance, security, multi-platform, scalability, etc. We choose the Windows operating system and the COM component framework as platform for the prototype. Furthermore, we use XML technology to store hierarchic data (XML documents) and for performing operations on XML documents: validation (XML-Schema), transformation (XSLT) and presentation (XSL). These standards allow us to use the MSXML4 component as validating XML parser and as XSL(T) processor. Furthermore, we use MSXML4 as implementation of the DOM for all operations that require creation, parsing or modification of XML documents. The second major commercial-off-the-shelf (COTS) component is the ExSpect engine, which is used as workflow engine. The ability of ExSpect to execute high level coloured Petri nets allows us to have an almost one-to-one translation of the conceptual framework to the technical architecture. Finally, we use the WOFLAN workflow validation tool to analyse Petri nets in order to proof the property of soundness.

- **Evaluation (Chapter 5)**

The last chapter discusses the achievements and contribution of the research. We will show that the use of domain knowledge in the configuration function and the generation of the contracting workflow definition brings a significant improvement in efficiency compared to current solutions. The contribution of this research is in the possibility of the results being applied by software designers who want to use ICT for service contracting processes.

1. Introduction

This research focuses on information and communication technology to support inter-organisational business processes based on outsourcing of work. The objective is to define a conceptual framework for a class of contracting processes, to provide the architecture of a new generic software component for this class of contracting processes and to prove its usefulness by creating a prototype. Before stating the research problem (1.6) and defining an outline for the research approach (1.7), we will give a rationale for our study first.

1.1 Motivation

A business process is a structured, measured set of activities designed to produce a specific output for a particular customer or market (Davenport [32]). According to this definition, a business process is an essential characteristic of an organisation, together with for instance the organisation structure, financial structure etc. The importance of business processes is reflected by a significant number of publications during the last decades. Womack et al [146] used the example of Japanese auto manufacturers who created a major competitive advantage by streamlining their business processes. Davenport [32] (p.1) argued that ‘business must not be viewed in terms of functions, divisions or products, but of key processes’. Instead of directing investments mainly to product innovation research, investments should be balanced between product innovation and process innovation. Davenport called this the ‘process approach’ [32] (p.6) and used the term ‘process innovation’ to refer to a fundamental change in business processes that brings a large improvement to an organisations performance. He argued that ‘no single business resource is better positioned than information technology to bring about radical improvement in business processes’ [32] (p.17). A similar perception of business processes was given by Hammer and Champy [60], who introduced the term ‘business process reengineering’. The central thesis in their work was that corporations must undertake a radical reinvention of their business processes. Hammer defined ‘reengineering’ as ‘the fundamental rethinking and radical redesign of business processes to achieve dramatic improvements in critical contemporary measures of performance, such as cost, quality, service and speed.’ The book starts with the observation that for two hundred years people have founded and built companies around Adam Smith’s discovery that industrial work should be broken down into its simplest and most basic tasks. Although this works fine in a period of mass markets and mass production, it is no longer the best solution in an environment characterised by rapid changes, strong competition and customers who want tailor made products and services to be available immediately. Due to these changes, organisations have to put aside knowledge of how work was done before and have to decide how the work can best be done now.

Inter-organisational business processes

A business process is seldom confined to the boundaries of a single organisation. Instead, an increasing tendency towards inter-organisational business processes is reported. Womack [146] showed that new logistic approaches require a higher level of co-ordination between customers and suppliers. Tapscott and Caston [113] (p.17) propose a new paradigm in information technology with ‘a shift from internal to interenterprise computing’. They signal the rise of what they call the ‘extended enterprise’, an organisation that reaches out to its customers and suppliers. They state that the ‘value chain’ introduced by Porter [104] is becoming a ‘value network’ encompassing multiple organisations. Normann and Ramirez [97] describe a shift towards a ‘value constellation’ as opposed to Porter’s ‘value chain’. The authors signal increasingly complex inter-organisational relationships that are subject to frequent reconfiguration (p.77). A trend that increases inter-organisational business processes is the tendency that organisations limit themselves to the activities in which they excel. This phenomenon has been reported by for instance Kalakota and Whinston [79] and Tapscott and Caston [113] (p.8). These so-called core activities are provided as services to other organisations. In return, they will use services of other organisations for the activities that do not belong to their core activities, causing networks of organisations to emerge. If the level of co-operation between corporations increases a ‘virtual corporation’ emerges, a theme addressed by for instance Davidow and Malone [33]. They describe how organisations cut down the number of suppliers and maintain a close link with the remaining suppliers. Concluding, inter-organisational business processes and inter-connectivity between business processes are becoming an issue increasingly.

Several mechanisms can be used for inter-organisational business processes, for example:

- *information sharing*: two or more organisations share information about for instance customers, sales, production, etc.
- *capacity sharing*: two or more organisations have a common pool of resources, which are shared between the organisations.
- *case transfer*: two or more organisations have interchangeable case types and transfer cases from one organisation to another to balance the workload.
- *contracting*: a client organisation contracts a provider organisation to supply products and/or services required in his business process.

Service contracting

This research addresses inter-organisational business processes that involve *contracting*. When one organisation buys something from another organisation, a distinction between ‘products’ and ‘services’ is often made. When the result of the suppliers activities consists of tangible goods and when ownership of the goods is transferred, one generally speaks of a ‘product’. If on the other hand the added value is characterised by activities performed at a particular place and time, one speaks of a ‘service’. In some cases however, it is difficult to make a clear distinction between ‘products’ and ‘services’ when a transaction involves a mix of tangible products and intangible services. When we look at their nature, products and services have different characteristics. For example, a tangible product can be produced and taken in stock, whereas intangible services can not. Although these differences exist, the question is whether they are relevant from the perspective of the contracting or buying process. This question is answered by for instance Normann and Ramirez [97], who state “whether customers buy a ‘product’ or a ‘service’, they really buy access to resources”. Hence, the authors use the term ‘offering’ to refer to both ‘product’ and ‘service’. Others, like Merz et al [93], have the same approach when

they consider payments and tangible goods as services too. In this dissertation, we will use the term ‘service’ as a synonym for both ‘product’ and ‘service’. Not because we deny the differences between products and services, but because the contracting concepts presented apply to products and services alike. Furthermore, we will use the term ‘service contracting’ for the activities that have to be performed by the service client to contract a service:

- *specify* the details of the required service;
- *negotiate* a commitment from a service provider;
- *monitor* the execution of the service;
- *accept* the result.

The simplest type of service contracting is when each business case requires one well-defined service. Situations that are more complex emerge when each business case requires a combination of different services and dependencies between the required services exists.

Service contracting and ICT

Service contracting is a process performed by the information system of a service client and requires inter-organisational information exchange. Information and communication technology (ICT) is therefore likely to contribute highly to the efficiency and effectiveness of service contracting, especially in situations of the more complex service contracting processes where a combination of services is required. An efficient service contracting process is becoming increasingly important. Due to developments like electronic commerce, customers expect a quick response 24 hours a day. When a customer enters an order on a web site, he wants to get an immediate response from the seller. This means that both the order *acceptance* (front-office) and the order *fulfilment* (back-office) must adhere to high standards. A fast front-office combined with a slow back-office makes no sense. When a seller makes a start with electronic commerce, the front-office is often given much attention. However, when the front-office is not connected to an efficient back-office, the result will not be very effective. Clearly, service contracting is part of an organisations back-office. The motivation for this research is to contribute to the efficiency of an organisations back-office in order to contribute to the performance of the organisation as a whole.

Structure of the chapter

Before stating the research problem (1.6) and the outline of this dissertation (1.7), we will first discuss recent developments in business-to-business electronic commerce (1.2 - 1.4) and software engineering (1.5). The area of electronic commerce is closely related to this research because it provides concepts and techniques for inter-organisational business transactions. The area of software engineering is closely related to this research because it provides state-of-the-art techniques for the new software component.

1.2 Electronic commerce

The application of ICT to service contracting requires some form of structured communication between the information systems of service client and service provider. This section focuses on electronic commerce in general (1.2.1), the enabling technologies (1.2.2) and the changes it brings to the business (1.2.3).

1.2.1 Introduction

Commerce and trade between individuals and between organisations has been conducted from the origin of mankind. The appearance of commerce however has changed significantly. The transition from barter to a system based on money was one of the first milestones in the evolution of commerce. Recent history has shown significant changes in the way commerce is conducted too. Almost without exception, the use of ICT has been the driving force or enabling factor behind these changes. The new possibilities for conducting commerce induced by ICT are often indicated by the notion *electronic commerce*. This notion is used for a variety of phenomena, which share the common characteristic that one or more phases of the trading process: information, transaction, delivery and financial settlement are performed electronically. With respect to electronic commerce, a distinction is often made between *business-to-consumer* electronic commerce and *business-to-business* electronic commerce. In both cases, digital information is exchanged between information systems. However, there is a difference in the *type* of information that is exchanged and in the *processing* of information. Business-to-consumer electronic commerce can involve a combination of structured and unstructured information. One party, the consumer, enters and interprets data manually. Business-to-business electronic commerce on the other hand involves only highly structured information, which is automatically generated by one application and automatically interpreted and processed by another application.

1.2.2 Internet as enabling technology

The Internet had a massive impact on electronic commerce and is expected to retain this impact. A definition of the term 'Internet' is given by the Federal Networking Council in the USA.

"Internet refers to the global information system that (i) is logically linked together by a globally unique address space based on the Internet Protocol (IP) or its subsequent extensions/follow-ons; (ii) is able to support communications using the Transmission Control Protocol/Internet Protocol (TCP/IP) suite or its subsequent extensions/follow-ons, and/or other IP-compatible protocols; and (iii) provides, uses or makes accessible, either publicly or privately, high level services layered on the communications and related infrastructure herein."

TCP/IP

The start of the Internet was in 1970 when the computers of four American universities were connected via the Network Control Protocol (NCP) as the beginning of ARPANET. It was in 1972 that an electronic mail application was introduced. In the mean time efforts were made to develop a new communication protocol which would eventually be called TCP/IP. As of January 1, 1983 all ARPANET hosts made the transition from NCP to TCP/IP. The TCP/IP (Transmission Control Protocol/Internet Protocol) consists of two layers. The IP (Internet Protocol) is the basis for the communication on the Internet. Each computer connected to the Internet is identified by a unique IP-address, which consists of four digits, each smaller than

256, separated by dots. All data exchanged between IP-addresses is split in packages and each package is wrapped into an IP-envelope. The IP-envelope contains the IP-address of the sender and the IP-address of the recipient. TCP (Transmission Control Protocol) is used on top of IP. At the sending computer, TCP splits messages in one or more packages. Each package is wrapped in a TCP-envelope, which is then wrapped in an IP-envelope. The IP-packages are sent via the Internet independent from each other. At the receiving computer the TCP protocol unwraps the data from the TCP-envelope and assembles the original message. If a package is received damaged, or is not received at all, the receiving computer asks the sending computer to retransmit the package.

Electronic mail

A popular application of Internet technology is Electronic Mail, which relies on the following types of standards:

- Transport mechanisms (SMTP, POP, IMAP4);
- Message encoding (RFC 822, MIME).

The *Simple Mail Transport Protocol* (SMTP), defined in RFC 821, is a peer-to-peer model and used for the exchange of email messages between SMTP servers. The *Post Office Protocol* (POP) is used for the communication between email server and mail clients. A mail client can use the POP or IMAP4 protocol to download the email messages that are waiting for him on the email server. Since its publication in 1982, RFC 822 has defined the standard format of textual mail messages on the Internet. As the format has seen wider use, a number of limitations have proven increasingly restrictive for the user community. The *Multipurpose Internet Mail Extension* (MIME) specification describes several mechanisms to solve most of these problems without introducing any serious incompatibilities with the existing world of RFC 822 mail. The MIME specification is given in RFC 2045 - 2049.

World Wide Web

World Wide Web is a network of information resources. The Web relies on three mechanisms to make these resources available to the widest possible audience:

- Communication protocols (HTTP / FTP)
- Mark-up languages (HTML, XML)
- Addressing schemes (URL)

The *HyperText Transfer Protocol* (HTTP) is an application-level protocol for distributed, collaborative information systems that has been in use since 1990. It usually takes place over TCP/IP connections, but can be implemented on top of other protocols too. The HTTP protocol is a request/response protocol. A client sends a request message to a server, which processes the information and returns a message to the client. The *HyperText Markup Language* (HTML) was developed at CERN in 1989. HTML was intended to serve as a format for documents that allowed information exchange between very different computer platforms. Complete platform independence was therefore the first requirement. Hypertext, by which a word or group of words could refer to another document, was the technique by which a user could navigate through documents. The work on HTML was based on the existing standard SGML (Standard Generalised Markup Language). SGML is a system for defining markup languages. Authors mark up their documents by representing structural, presentational and semantic information alongside content. The markup information takes the form of *tags* that are inserted in the contents. When an HTML document is displayed on a computer system the markup information is mapped to the graphic capabilities of the computer system, for instance by mapping a title to a

font Arial 18 pt, centred. The *Extensible Markup Language* (XML) is -like HTML- an application of SGML. However, unlike HTML, XML allows the user to define the tags to appear in the document. This makes XML an ideal standard for exchanging structured data. A *Uniform Resource Locator* (URL) is a reference to a document or other entity on the Internet and is used as universal address. A URL typically consists of three parts: (1) the naming scheme of the mechanism used to access the resource (2) the name of the machine hosting the resource and (3) the name of the resource itself, given as a path.

Organisation

The Internet is not a network that is owned or under the control of a central organisation. There is a number of organisations that play an important role for the development of the Internet though. The most influential international organisations are:

- **Internet Society**

The Internet Society (www.isoc.org) is a professional membership organisation of Internet experts that comments on policies and practices and oversees a number of other boards and task forces. Its principal purpose is to maintain and extend the development and availability of the Internet and its associated technologies and applications.

- **Internet Engineering Task Force (IETF)**

The Internet Engineering Task Force (www.ietf.org) is a large open international community of organisations and individuals concerned with the evolution of the Internet architecture. The actual technical work of the IETF is done in its working groups, which are organised by topic into several areas (e.g., routing, transport, security, etc.).

- **Internet Engineering Steering Group (IESG)**

The Internet Engineering Steering Group is responsible for the technical management of IETF activities and the Internet standards process. The IESG is directly responsible for the actions associated with entry into and movement along the Internet "standards track", including final approval of specifications as Internet Standards.

- **Internet Architecture Board (IAB)**

The Internet Architecture Board (www.iab.org) is the technical advisory group of the Internet Society. It is responsible for defining the overall architecture of the Internet providing guidance and broad direction to the IETF. The IETF chair and all other IESG candidates are appointed by the IAB. The IAB is further responsible for editorial management and publication of the Request for Comments (RFC) document series. It also serves as an appeal board for complaints of improper execution of the standards process.

- **Internet Assigned Numbers Authority (IANA)**

The Internet Assigned Numbers Authority (www.iana.org) is in charge of all "unique parameters" on the Internet, including IP (Internet Protocol) addresses.

An organisation that is not part of the 'official' Internet standardisation bodies as listed above, but which is very active and influential is the World Wide Web Consortium.

- **World Wide Web Consortium (W3C)**

The World Wide Web Consortium (www.w3.org) is an international industry consortium that was founded in 1994. The consortium attempts to find common specifications for the Web and to make it freely available throughout the world. W3C is funded by member organisations and is vendor neutral.

1.2.3 Changes to the business

Electronic business is expected to bring enormous changes to the way business is done. The changes that electronic commerce will bring to suppliers and consumers are described by for instance Bollier [18] and Kalakota et al [79] as: reduced transaction costs, lower product cycle times, faster consumer response and improved service quality. The effect of electronic business to economic transactions is described in reports of Deloitte Research [105, 106], which describe (among others) the following forces:

- **Markets become more like “textbook” markets**

For many retail products, internet-based software agents will be used to search products, compare prices, conduct transactions and arrange for delivery. This will lead to a situation where the same information is shared by all market participants, which in turn leads to markets that are more alike the markets described in economic textbooks.

- **Tangible products transform into digital form**

Certain types of products will “morph” into digital form in order to minimise distribution costs by delivering the product via the Internet. Products types like software and publications have already begun this process. An identical wave is expected in the music industry.

- **Development of markets for trust and privacy**

In order to move to transactions on the Internet, questions about reliability, identity, security and privacy must be answered. This gives opportunities to a whole new industry whose primary purpose to Internet users is to evaluate vendor claims, guarantee transactions and serve as a general seal of approval.

A joint research report [43] of The Economist Intelligence Unit and Booz • Allen & Hamilton identifies the following seven megatrends as driving forces behind the transformation of global business due to electronic commerce.

- **The Internet offers new marketing and sales channels**

Internet provides companies with a new channel to reach customers, in addition to the traditional brick & mortar channel. The new channel allows companies to establish a global reach combined with a one-on-one relationship with their customers. The Internet as new sales channel did not only make existing markets more efficient; it also created new markets and changed the structure of existing markets. Examples of new possibilities for marketing, sales and distribution in an electronic marketplace are described by Choi et al [27]. Banks and insurance companies have discovered the Internet as a new distribution channel that allows direct communication with customers. Companies are becoming increasingly aware of the importance to control this new distribution channel, resulting in new instruments like web-portals and free Internet access.

- **The balance of power is shifting to the customer**

Due to the possibilities of the Internet, customers become highly informed *before* they enter the retail store. These customers have been empowered by the information they find on the Internet and are able to compare products or services easily. They expect customised products and services, fast delivery times, excellent customer service and round-the-clock availability.

- **Competition is intensifying**

The Internet declined the barriers to enter international markets. Furthermore, Internet technology allows new business models to be used. These factors will increase competition. Traditional companies will have to rethink their situation and maximise their value in the new environment.

- **The pace of business has accelerated**

Internet has accelerated the speed at which companies must operate: quicker customer response, faster decision making, faster product distribution and shorter time-to-market. The driving forces behind this acceleration are a technology *push* (more possibilities) and a customer *pull* (higher expectations).

- **Companies are transforming into extended enterprises**

The Internet is enabling companies to break through organisational and geographic boundaries and create extended enterprises. Extended enterprises can result in enormous cost savings because of the ability to streamline the entire supply chain. An example of the latter is *Supply Chain Management* (SCM). A supply chain is a chain of organisations in which each organisation is the supplier of the next organisation. If all organisations in a chain try to optimise their own performance by making bilateral agreements with their direct suppliers and customers, the resulting performance of the chain as a whole may be sub-optimal. The essence of Supply Chain Management is to look at a supply chain as a whole and to optimise the total performance of the chain. This may for instance lead to a solution in which a manufacturer at the beginning of the chain becomes responsible for maintaining the stock level in the store at the end of the chain, based on actual sales information. Another advantage that can be achieved in an extended enterprise is *Efficient Customer Response* (ECR). In most cases where multiple departments of multiple organisations are involved in a customer request, the total response time to the customer is the sum of a large waiting time and a small actual processing time. If the co-ordination between departments or organisations can be enhanced, the waiting time can be decreased because of better planning.

- **Companies are re-evaluating their role in the value chain**

Internet allows companies to re-invent their existing marketing and distribution channels. Selling directly to customers or retailers has become a viable option, reducing the dependency on intermediaries. With the rise of the Internet, intermediaries in particular must add value or risk being cut out entirely. Examples of added value, which an intermediary can offer, are: matching customers with suppliers economically, providing access to targeted groups of customers, delivering goods more efficiently, etc.

- **Knowledge is becoming a key strategic asset**

Companies can use Internet technology to exploit the collective knowledge that is available. One of the most powerful features of the Internet as distribution channel is the possibility to gather detailed information about the behaviour of each individual customer. Innovative companies therefore are using the Internet to acquire, analyse and share information about individual customers and customer segments. This customer information can be used to target and secure the most profitable customers. Instead of using one marketing strategy for an entire population of customers, we are now able to use a different marketing strategy for each individual customer, based on his customer profile. This approach is often called *Customer Relationship Management* (CRM).

1.3 Industry solutions for electronic contracting

1.3.1 Electronic data interchange

Electronic Data Interchange (EDI) is one of the oldest techniques for structured business-to-business communication. A number of definitions for EDI are used, among which:

Electronic Data Interchange is the electronic transfer from computer to computer of commercial or administrative transactions using an agreed standard to structure the transaction or message data. (ISO 9735)

Electronic Data Interchange is the inter-company computer-to-computer communication of standard business transactions in a standard format that permits the receiver to perform the intended transaction. (Sokol [110])

Electronic Data Interchange is the electronic exchange of structured and standardised data between computers of parties involved in a (business) transaction. (Hofman [62])

Electronic Data Interchange is the interprocess communication (computer application to computer application) of business information in standardised electronic form. (Kalakota et al [79])

However, in general, the term EDI is often used to refer to message exchange characterised by the use of the EDIFACT standard [69, 70, 71] and message handling services like X.400. The advantages of EDI can be phrased in one sentence as a considerable reduction of transaction costs by improving speed and efficiency (Kalakota et al [79]). Others report the same advantages (Sokol [110]; Hofman [62]; Van der Vlist et al [122, 123]). The question whether the benefits of EDI exceed the costs of implementing EDI are addressed by for example Hoogewegen et al [65, 66]. Clearly, efficiency can be improved because manual activities such as document handling and data entry are no longer needed. The result of this is a decrease of costs, but also a decrease in the number of errors that occur when data is being entered manually. In addition to the direct advantages of EDI, more advantages can be reached when EDI is used as an enabling technology to change business processes.

The use of EDI started in the eighties and increased every year since then. Currently, EDI is considered as 'proven technology' for a considerable number of years already. However, the use of EDI is almost entirely restricted to stable and long-term business relationships with a high information exchange volume (Lee [91]). There are several reasons to explain this. The first reason is the combination of highly structured information and an inter-organisational context. Organisations with different business processes and different corporate data models must agree on a common interchange standard. This requires considerable information analysis and standardisation efforts, which may take months or years. The second reason is the complexity of the EDIFACT standard. Translation of functional requirements to message implementation guidelines that include a mapping to the EDIFACT syntax requires expertise that is often not available within organisations. Similarly, configuration of standard EDI software must often be performed by a third party. All this together makes implementation of EDI a costly and time-consuming affair.

Finally, we will discuss standard software components for EDI. The functionality of standard EDI software is in most cases limited to data *conversion* and *communication*. Internal applications create so-called ‘inhouse files’ with messages in internal format (often fixed record files). These inhouse files are read by the EDI software, converted to EDIFACT format and sent to the Message Handling System. In the same communication session, EDIFACT messages are received from the Message Handling System, converted to internal format and made available to internal applications in the form of inhouse files. Typical EDI software is batch-oriented; conversion and communication actions are scheduled on regular times during a day.

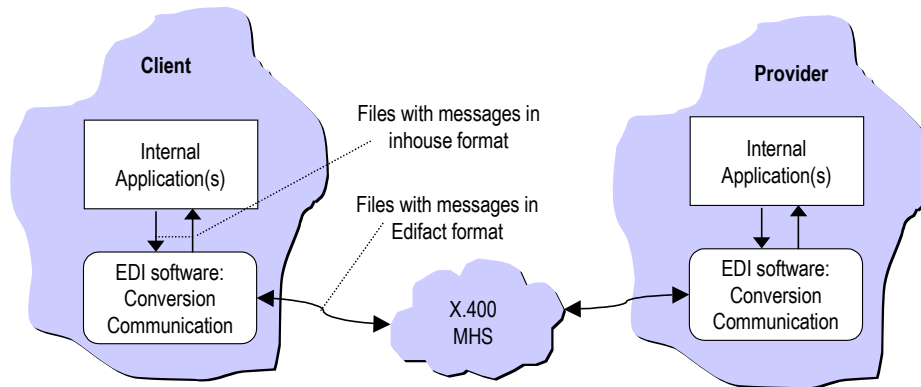


Figure 2 Position of EDI software

1.3.2 XML based messaging frameworks

The importance of Internet technology for electronic commerce can hardly be exaggerated. However, when Internet technology became first available, it was used for electronic mail and distribution of static information via web-sites mainly. The use of Internet technology for business transactions was very limited due to the following reasons. First, although a number of successful standards like HTTP, HTML, SMTP and MIME were available, the Internet was lacking a standard for exchanging structured data. Second, because the Internet is not under the control of a single entity, guaranteed delivery of messages and preventing unauthorised persons to view or alter the contents of a message could not be obtained without implementing proprietary solutions at both parties involved in a business transaction. Finally, due to the lack of standards and due to the maturity level of the Internet at that time, standard software was hardly available.

Today, using Internet technology for business-to-business electronic commerce is a viable option. A number of developments made this possible. A major breakthrough was the acceptance of the eXtensible Markup Language (XML) [125] as a standard for structured information on the Internet. Although XML is a standard with great potential itself, its power is even increased because of a large number of supporting standards. Among these are XML-Schema (validation of XML documents) [130, 131, 132], XSL (presentation of XML documents) [134], XSLT (transformation of XML documents) [128] and DOM (manipulation of XML documents) [126]. Because XML allows the user almost complete freedom in defining the structure of XML documents, there was a need for standard XML document schemas that could be used in electronic business transactions. This challenge has been answered by a number of initiatives.

- **BizTalk**

The BizTalk (www.biztalk.org) initiative provides a common approach to using XML for application integration and electronic commerce, with the objective to accelerate the rapid adoption of XML. BizTalk was launched by Microsoft in 1999 and is an open standard that can be supported by other software vendors too. The specification of BizTalk documents and transport bindings for HTTP and MIME can be found in the BizTalk Framework 2.0 [96]. The standard encompasses the use of special BizTalk XML tags for header information and the use of SOAP for exchange of XML messages.

- **ebXML**

The ebXML initiative (www.ebxml.org) is a joint initiative of the United Nations (UN/CEFACT) and OASIS in which many companies participated. The standard is based on proven Internet technologies like HTTP, MIME, SMTP, FTP, UML and XML. According to [41, 42] the ebXML standard is designed for electronic interoperability, allowing businesses to find each other, agree to become trading partners and conduct business. This ambition is realised by a shared repository with company profiles, business process models and message structures. The information can be used by partners to agree on a formal *collaboration protocol agreement*. Here after, partners can use the ebXML messages through a standard message transport mechanism.

- **cXML**

The objective of the ‘commerce XML’ specification (www.cxml.org) [14] is to provide a streamlined XML-based protocol between procurement applications and suppliers. The specification is developed by a group of organisations, among which Ariba, and is a public standard. The specification consists of a cXML document framework, which is used as a generic ‘envelope’ structure for all cXML messages (cXML.dtd). Based on this framework, a number of message types is defined, e.g. ‘OrderRequest’ and ‘OrderResponse’. The specification also defines mechanisms for transfer of messages via HTTP.

The notion that standard components for electronic purchasing require standard purchasing frameworks is addressed by a number of initiatives. The *Internet Open Trading Protocol* (IOTP), of which version 1.0 is published as RFC 2801 by the IETF [68], provides an interoperable framework for electronic commerce. The developers of IOTP seek to provide a virtual capability that safely replicates the traditional methods of trading, buying, selling and value exchanging. Another initiative is launched by the *Open Buying on the Internet* (OBI) consortium (www.openbuy.org) dedicated to developing open standards for business-to-business Internet commerce [98]. The initial focus of OBI is on automating high-volume, low-dollar transactions between trading partners. Finally, a last example is *RosettaNet* (www.rosettanet.org), a consortium of major technology companies working to create and implement industry-wide, open electronic business standards. RosettaNet standards encompass a *dictionary*, an *implementation framework* and *partner interface processes*; specialised XML-based dialogs that define business processes between supply chain partners.

Clearly, XML is very much suited to express structured data. However, in order to perform a business transaction, a messaging mechanism is required to carry messages between the business partners. The *Simple Object Access Protocol* (SOAP) [137] is an answer to this challenge. It builds on the existing XML and HTTP standards and offers a standard for invoking a service exposed by a web-application. SOAP messages can be used in a “request/response” pattern in which a client invokes a service by sending a SOAP message after which the answer is re-

turned to the client by another SOAP message. The message data is formatted as XML document, which is encapsulated in an HTTP protocol message.

The XML standard for expressing structured data and the SOAP standard for invocation of web-services in a distributed environment are major steps in creating a level of interoperability required for business-to-business transactions. In an open environment, there are millions of businesses, which are potential trading partners. Finding the right trading partner and discovering how to conduct business with that partner is a major challenge. The *Universal Description, Discovery and Integration* (UDDI) standard [114, 115] is an answer to this challenge. The objective of the UDDI project (www.uddi.org) is to create a framework that allows businesses to discover each other and define how they interact over the Internet. Information is shared in a global registry that is meant to accelerate the global adoption of business-to-business electronic commerce. The UDDI specifications build on existing Internet standards like XML, HTTP and DNS. Furthermore, it uses the SOAP messaging specification. The core information model used by UDDI contains general information about businesses and the services they offer. Technical specifications like communication protocols, interchange formats and interchange sequencing rules are not contained in UDDI. Instead, a reference to this kind of specifications is used in the UDDI registry.

1.3.3 Workflow interoperability

Information systems support *business processes*. A business process consists of a number of *tasks* and *conditions* that specify the order in which tasks are executed. An information system used to automate a business process must have functionality to execute a task for a specific case and it must have functionality to route a business case through the right tasks in the right order. We will refer to the former functionality as *execution* and to the latter as *control*. Until some years ago, execution and control were interwoven in information systems. The structure of the business process was ‘hidden’ in the application logic. Adaptation of information systems to changes in the business process or re-use of applications was difficult. The emergence of *workflow management systems* marked an important step in information technology. The essence of workflow management is the separation of execution and control. Execution takes place in a number of smaller applications that are independent of the context or place in the business process. Control is housed in a generic software component: the workflow management system.

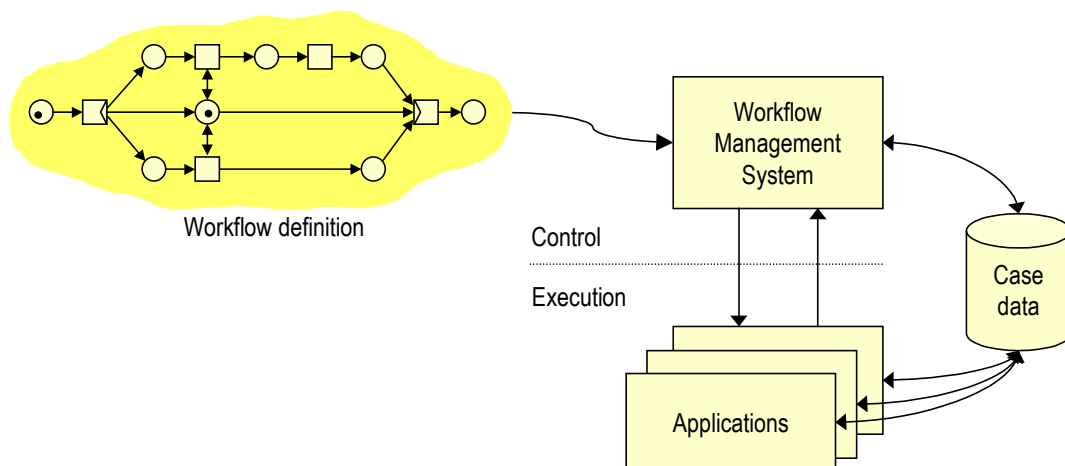


Figure 3 Workflow Management is separation of execution and control

A workflow management system is configured with an explicit model of the business process. Each business case processed by the workflow management system has its own *case attributes*. The workflow management system uses these case attributes to route the case through the business process. The routing may involve sequential execution of tasks, selection of tasks, parallelism and iteration. Each task is executed by a *resource*. The resource can be a person or (a part of) an organisation, a machine or a computer application. The workflow management system is responsible for allocating resources to tasks.

An overview of issues related to workflow management is given by for instance Joosten [78]. Van der Aalst and Van Hee [12] describe methods and techniques for modelling workflows. Standards for workflow management systems are being developed by the *Workflow Management Coalition* (WFMC). The WFMC (www.wfmc.org) is a grouping of companies who have joined to achieve a level of interoperability by common standards for various functions. The Workflow Reference Model [140] is a document in which a common reference model for workflow management systems is provided. It covers the concepts, terminology, and general structure of a workflow management system, its major functional components and the interfaces and information flows between them. The WFMC defines workflow as ‘the computerised facilitation or automation of a business process, in whole or in part’. A workflow management system is defined as: ‘a system that completely defines, manages and executes "workflows" through the execution of software whose order of execution is driven by a computer representation of the workflow logic’. The major components and interfaces within the WFMC workflow architecture are shown in Figure 4.

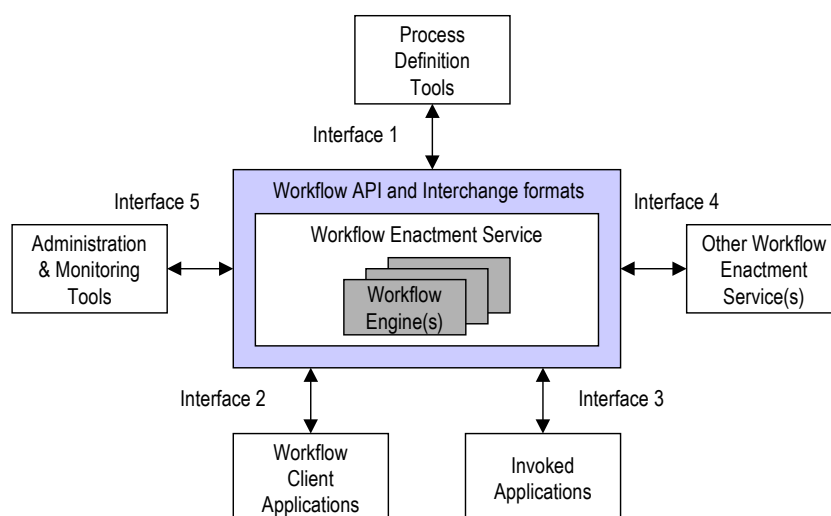


Figure 4 The Workflow Reference Model - Components and Interfaces

The heart of the system is the Workflow Enactment Service, defined as ‘a software service that may consist of one or more workflow engines in order to create, manage and execute workflow instances’. The workflow enactment service is configured by Process Definition Tools. These tools are used to analyse, model, describe and document a business process. The process definitions can be exchanged from the Process Definition Tools to the Workflow Enactment Service, or the process definitions can be stored in a repository accessible to both products. Interaction with external resources (humans, applications) occurs via one of two interfaces, the *client application* interface or the *invoked application* interface. A client application is an application that is not activated by the workflow enactment service, but communicates with the workflow enactment service via a *work list*. A work list is a queue of work items to which the workflow

enactment service adds items and from which the client application retrieves items. Workflow relevant data may be embedded in the work item. Data can also be passed to the client application via some form of shared store to which the work item contains a unique reference. An invoked application is an application that is invoked by the workflow management system to automate an activity, fully or in part, or to support a workflow participant in processing a work item.

The concepts of workflow management as well as techniques for modelling workflows have been described by Van der Aalst and Van Hee [12]. Jablonski and Bussler [74] describe expectations and fears with respect to the business implications of workflow management systems. The major expectations are increasing quality of service, improving service to the clients, increasing productivity and reducing costs. Fears and reservations caused by the introduction of these new technologies are: too rigid control, too little functionality and too much inflexibility. Although business process reengineering (Hammer and Champy [60]) is often associated with workflow management, there is no direct link between the two concepts. However, one of the advantages of workflow management systems, the ability to change business processes easily, can be a great help in the implementation of a reengineered business process. Further, the introduction of a workflow management system can be a catalyst to rethink the structure of the business process.

The Workflow Management Coalition addressed the integration of workflow management and electronic commerce in two white papers [143, 144]. The solution they propose is an integration of the workflow management systems of customer and supplier based on the WfMC Workflow Interoperability Specification [142]. The workflows of customer and supplier then become part of an inter-organisational workflow, in which each organisation executes a sub-process. This is illustrated in Figure 5 where workflow engine A initiates a process instance in workflow engine B, waits for its completion after which the own process is resumed. The WfMC published the Wf-XML standard [145], based on XML and HTTP, to be used for this purpose.

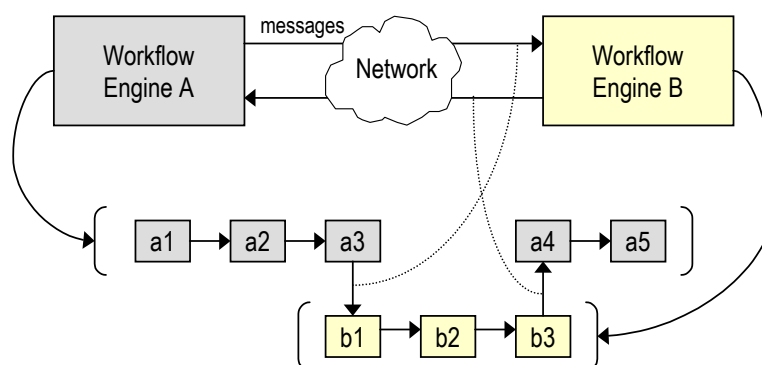


Figure 5 Interoperability between workflow engines

Interoperability between workflow engines belonging to different organisations has great potential for inter-organisational business processes. However, in order to be successfully applied, three conditions must be fulfilled.

1. Technical infrastructure

The proposed solution requires each participating organisation to have a workflow management system that conforms to the WfMC interoperability specification. Furthermore, a

communication infrastructure must be available for the exchange of messages between the workflow engines.

2. Interoperability contract

If the infrastructure is available, organisations will have to make agreements in an *interoperability contract* [144]. This contract contains the identifications of the workflow definitions that can be initiated by the other organisation and the workflow engine(s) that can be used for this purpose. It also contains agreements about semantics and syntax of workflow relevant data that is passed from one workflow engine to another. Furthermore, it specifies the communication protocol between the two workflow engines in terms of message types (requests, responses, notifications) and message sequencing. Finally, the interoperability contract describes exception handling.

3. Workflow definition

When the interoperability contract is concluded, parties will have to change their internal workflow definitions accordingly to support the agreed communication protocol. Each outgoing message must be triggered by the internal workflow and each incoming message must be a trigger for the internal workflow. It is important to check the correctness of the workflow definition in combination with the agreed communication protocol. Wrong implementation of the communication protocol may for instance cause a deadlock situation to occur in a specific process instance.

With the interoperability specification, the technical barriers for inter-organisational workflow have been lowered significantly. However, although the solution lowers *technical* barriers, it still leaves a number of *organisational* barriers that are associated with inter-organisational business processes. Some of these are described by Van der Aalst and Van Hee [12].

1. Authority

Internal resources are under the authority of the organisation to which they belong. Machines, for instance, are owned by an organisation and employees have a contract with the organisation in which the mutual obligations are arranged. This makes it possible for a workflow management system to *assign* the execution of a task to an internal resource. Because this decision can be based on full information of available internal resources, (sub) optimality in scheduling can be obtained. *External* resources are part of another, autonomous, organisation. Therefore, the workflow management system can not simply *assign* the execution of a task to an external resource but has to *negotiate* instead. A negative result of the negotiation process must be reckoned with. Furthermore, information about availability of external resources is often not available to the workflow management system. Instead, a trial-and-error approach must often be followed in the negotiation with external service providers.

2. Information semantics

Resources are controlled by information. In general, all resources belonging to one organisation have, either implicit or explicit, a common understanding of information semantics, for instance because they are based on a corporate data model. External resources however, can be based on different standards. Additional facilities such as data conversion are often required.

3. Correction mechanisms

Each organisation has its own - formal and informal - correction mechanisms. Cases can be transferred between internal resources without the intervention of the workflow management system, solving a large amount of smaller disruptions. External resources do not have a similar correction mechanism. If an external resource fails, the work is not automatically handed over to an alternative resource. Instead, the workflow management system must initiate negotiations with one or more alternative resources. Furthermore, status information of internal resources can be available immediately. Possible disruptions can be signalled in an early stage. External resources however can hide status information or make it available with a delay, possibly causing disruptions to be signalled too late. A pro-active approach of disruptions at external resources might therefore be required.

1.4 Scientific research on electronic contracting

1.4.1 Definition

This section gives an overview of publications in the field of electronic contracting. The term ‘electronic contracting’ was already mentioned by Lee in 1988 [86]. The paper discussed improvement of contracting processes through computer support. In this research, we define a contract as ‘*an agreement between two parties in which the mutual obligations are stated*’. Furthermore, we define the term ‘electronic contracting’ as ‘*a contracting process in which the communication between parties is performed by electronic means and in which the processes at the involved parties are supported by computer applications.*’

1.4.2 The Language / Action Perspective

An influential approach to the modelling of communicating information systems is called the *language / action perspective* (LAP). The basis for LAP was a growing awareness that linguistic theories are relevant for the design of communicating information systems. A cornerstone of the LAP approach is the linguistic theory of *speech acts* developed by Searle in 1969 [108]. Later, Searle and Vanderveken developed *illocutionary logic* as a logical formalisation of the theory of speech acts [109]. Further, an important contribution to the LAP was made by Kimbrough [80, 81]. Various authors like Lee [86] and Dignum and Weigand [35] have shown that *deontic logic* can be successfully applied to electronic contracting. The origin of deontic logic was in the classic philosophy of ethics and has a major application in the philosophy of law. One of the features of deontic logic that makes it attractive for electronic commerce is that concepts like ‘obligation’ ‘permission’ and ‘authorisation’ are part of the formalism intrinsically. Because of this, Weigand, van den Heuvel and Dignum [139] argue that other formalisms like Petri nets and Data Flow Diagrams are less suitable for modelling electronic commerce processes because they do not contain these concepts intrinsically. An example of a formula in which deontic logic and speech acts are combined is given in the expression below. The formula is of the form $[\alpha]\varphi$ which means that after the performance of the action α the formula φ holds. The example is taken from [39] and means that after agent “i” commits himself towards agent “k” to perform “ α ” and agent “k” declared that agent “i” is permitted to perform “ α ” then agent “i” has the obligation towards agent “k” to actually perform “ α ”.

$$[\text{COMMIT}(i, k, \alpha)] [\text{DECL}(k, P_{ik}(\alpha(i)))] O_{ik}\alpha$$

An important advantage of deontic logic and illocutionary logic is the possibility to reason about properties of the electronic commerce protocol [35]. For example, it can be possible to prove that following an order protocol leads to a state where one party has the obligation to deliver a product and the other party has the obligation to pay for it. This ability to reason about contracting protocols is used by Bons [19] and Bons, Dignum, Lee and Tan [21] to propose a formal theory on the design of trustworthy trade procedures.

The language / action perspective is heavily used in research on intelligent agents. An example of an agent architecture in which the concepts mentioned above are used is given by Verharen and Dignum in [120, 121]. In their view, a Co-operative Information Agent (CIA) has an *agenda* containing the actions to be performed by the agent (obligations). The agent can reason about the actions on the agenda, add new actions to the agenda and remove existing actions from the agenda. The architecture of the CIA is shown in Figure 6. The main engine is the *task manager*, which maintains the agenda and plans and schedules the tasks. The *contract manager* stores and monitors contracts; a formal description of the communication behaviour between two agents. The *communication manager* handles all external communication of the CIA, which is not expected to follow a fixed communication protocol. Instead, a rich communication language based on the theory of speech acts is used allowing the CIA to react when other agents do not follow the same protocol. The Lexicon is used by the communication manager for the definition of the terms that are used in communication with other CIA's. Finally, the *service manager* checks whether another agent is authorised to request a specific service from the CIA and if the service is available.

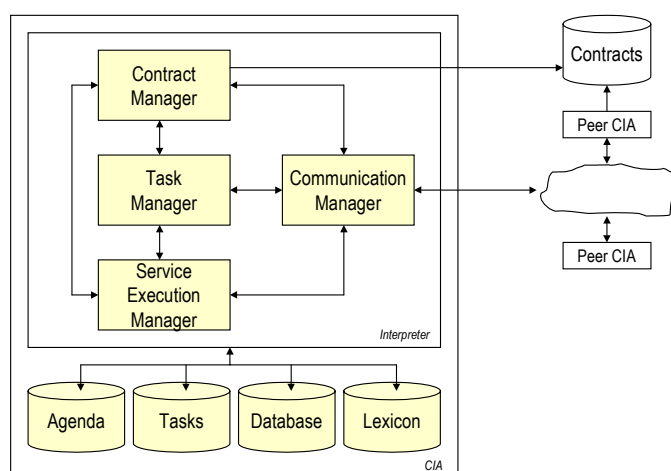


Figure 6 Architecture of a 'co-operative information agent' proposed by Verharen and Dignum

1.4.3 Documentary Petri Nets

Another approach to model the inter-organisational communication is a formalism called *Documentary Petri Nets* (DPN) proposed by Lee [86, 89, 91, 92]. The development of the formalism was initiated by the observation that most electronic data interchange links were limited to long lasting trading relationships involving a high number of transactions and between parties with a high level of mutual trust [86]. An explanation for this was the high setup costs that could only be recovered over a longer period with a large number of transactions. These high setup costs stem from the necessity to know about each others "way of doing business" before data can be exchanged electronically and to agree on a common trade procedure that will be followed by both parties. The term "trade procedure" is defined in [86] as "the mutually

agreed upon set of rules that governs the activities of all parties involved in a set of related business transactions". The idea behind Documentary Petri Nets is to decrease the setup costs significantly by making available standard trade procedures. This requires a common language in which the trade procedures are described, which must be formal, graphical and computer interpretable. There after, groups of business experts can use the formalism to specify trade procedures. Finally, companies can download standard trade procedures from a repository and configure internal software systems with it. The formalism of Documentary Petri Nets is an attempt for such a common language for expressing standard trade procedures.

The formalism of Documentary Petri Nets is described in [86, 89]. Each DPN is a coloured Petri net with the following extensions:

- The exchange of information is modelled by document places represented as a square box. The colour of the place models the structure of the information in the information parcel. Sending an information parcel is represented by a transition labelled **X to Y: D**, in which **X** identifies the sender, **Y** the receiver and **D** the type of information parcel that is exchanged. The transition has a document place of type **D** as output place. Conversely, receiving an information parcel is modelled by a transition labelled **Y from X: D**. This transition has a document place of type **D** as input place.
- The exchange of goods is modelled by goods places represented as cubes. The colour of the place models the properties of the goods like quantity, weight, etc. The transfer of goods among parties is modelled by transitions labelled **X to Y: G** and **Y from X: G** similarly to the modelling of information exchange, but in this case **G** refers to goods.
- The exchange of funds is modelled similar to the modelling of information.
- The deontic states of each individual role are modelled by tokens in the control places in the sub-net of each role. The control places are represented by a circle and have a label with the description of the deontic state, e.g. the label **oblig(x, a)** to indicate that party **x** has an obligation to perform action **a**.

Although a DPN describing a trade procedure between two roles is in fact one model, it is possible to create a view on the DPN for each individual role involved that gives the sub-net which contains only the transitions that model actions which are under control of that role. A state transition in a DPN is enabled by receiving an information parcel, goods or funds, or the expiration of a timer. Firing a transition can lead to sending information parcels, goods or funds and/or setting a timer. Finally, since the formalism of DPN's is based on Petri nets, properties like liveness and boundness can be analytically checked.

The DPN formalism and the ideas behind it are used in a practical implementation of a software system called InterProcs [90]. The motivation behind InterProcs is to validate the concepts and to demonstrate the feasibility. There are two main components: InterProcs Designer and InterProcs Executor. The Designer component offers a graphical user interface by which the user can create DPN's. The Executor component loads the specification of a DPN and simulates and executes the trade procedures defined in the DPN. An example of the InterProcs user interface is given in Figure 7 and can be found at www.euridis.fbk.eur.nl.

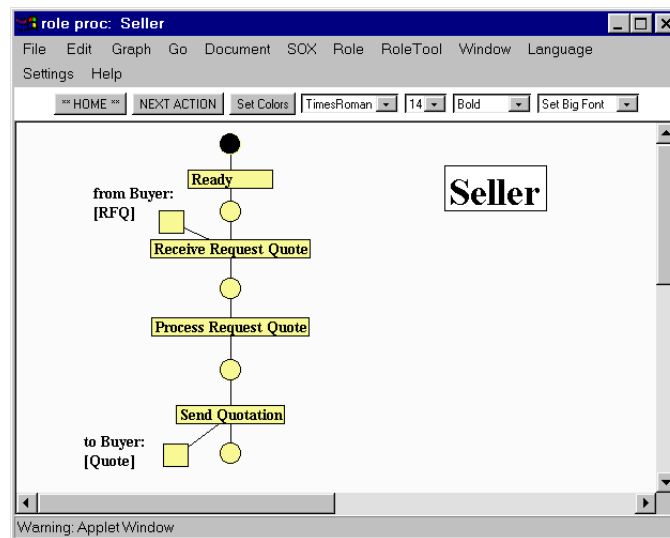


Figure 7 Example of the InterProcs user interface

1.4.4 Interorganisational workflows

The formalism of classical Petri nets was defined in 1962 By Carl Adam Petri [103]. Later, others like Jensen [77] and Van Hee [61] proposed an extension with time, colour and hierarchy. The application of Petri nets to workflow management was already described in 1979 by Ellis [48] who used the formalism of Petri nets for office information flows in what he called ‘Information Control Nets’. Since then, the application of Petri nets to workflow management has been addressed by for instance Van der Aalst [1, 2, 6] and Van Hee [12]. The reasons for using Petri nets as modelling technique for workflows are given by Van der Aalst [4] as (i) formal semantics despite graphical nature (ii) state-based instead of event-based and (iii) abundance of analysis techniques. An important notion in the analysis of workflows is a notion of correctness called ‘soundness’ [6] (see page 53). The analysis techniques to prove the property of soundness are given in [5, 9]. A software tool in which these analysis techniques are implemented is Woflan [119].

The interoperability between local workflows has been addressed by Van der Aalst [7] who identified six forms of interoperability: capacity sharing, chained execution, subcontracting, case transfer, extended case transfer and loosely coupled. Furthermore, he extended the notion of soundness of a local workflow to the notion of *global soundness* of a system of loosely coupled workflows [7]. A drawback of this approach is that in order to check the global soundness one needs a model of the entire global workflow. A formal approach where knowledge of the global workflow is not required is presented by Kindler, Martens and Reisig [84]. The authors specify the dynamics of the inter-organisational system by a set of scenarios in the form of message sequence charts. Each local workflow can then be checked for local soundness with respect to these scenarios. It is proven that if each local workflow is sound, the entire workflow is globally sound. A different approach to the same problem is proposed by Van der Aalst [10] and Van der Aalst and Weske [13]. The authors propose a P2P (Public-to-Private) approach that consists of three steps. First, the parties define a common understanding of the inter-organisational coordination by defining a global public workflow. In the second step, the public workflow is partitioned over a number of domains (organisations). Finally, for each domain a private workflow is constructed such that the private workflow is a subclass of the corresponding part of the global workflow. If this approach is followed, the global workflow is free

of deadlocks and other similar anomalies. Moreover, the overall workflow is a subclass of the public workflow, which guarantees that the protocol specified in the public workflow is actually realised.

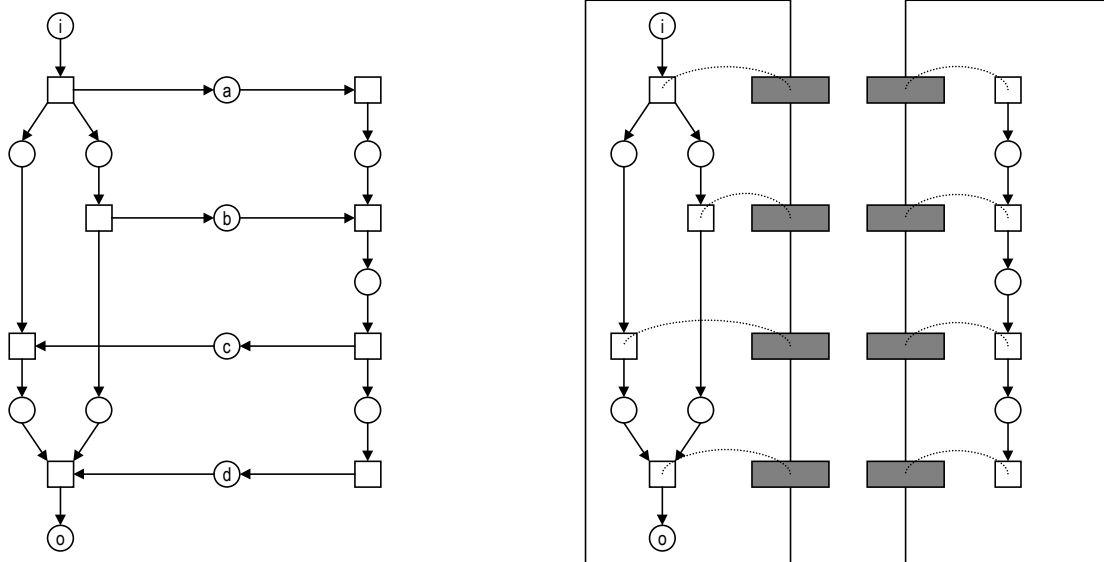


Figure 8 Example of a public workflow (left) and partitioned public workflows (right)

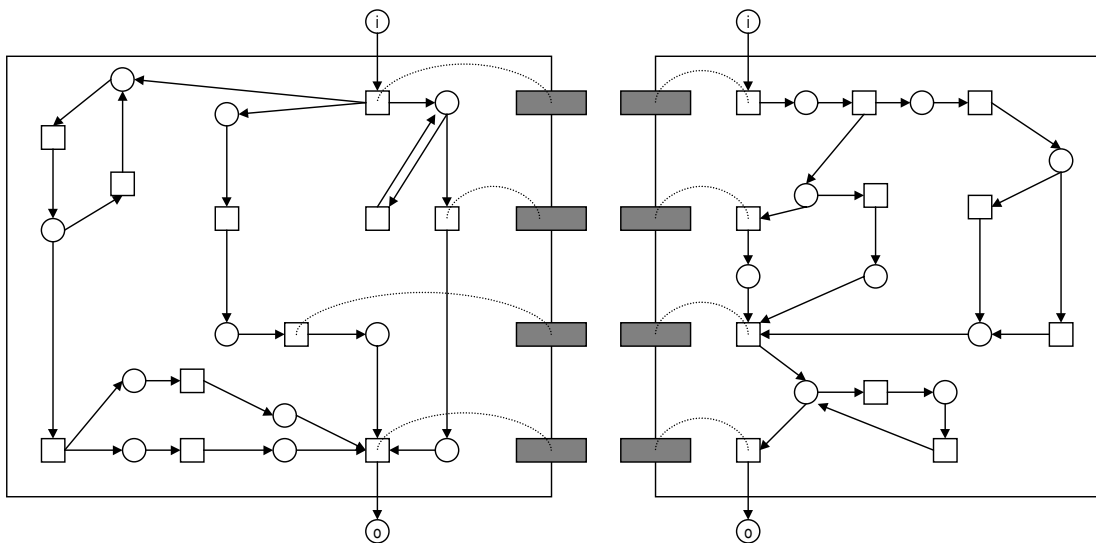


Figure 9 Example of a private workflow

1.4.5 Mobile agents

Much research on electronic contracting is focused on *agent technology*. An agent is an entity that acts on behalf of another entity. For example, a real estate agent tries to sell houses on behalf of the owners of the houses. A *software agent* is a software component that performs a task for a human or another software component (Dalmeijer, Hammer and Aerts [31]). A weak notion of agency can be given by four characteristics (Wooldridge and Jennings [147]; Jennings et al [75, 76]).

- *autonomy*: agents operate without the direct intervention of humans or others, and have some kind of control over their actions and internal state.
- *social ability*: agents interact with other agents (and possibly humans) via some kind of agent-communication language.
- *reactivity*: agents perceive their environment, and respond in a timely fashion to changes that occur in it.
- *pro-activeness*: agents do not simply act in response to their environment, they are able to exhibit goal-directed behaviour by taking the initiative.

Intelligent software agents can be used for a variety of tasks. The use of intelligent agents for retail electronic commerce is discussed by Guttman and Maes [59]. They describe a first generation of shopping agents that assist the user in making a buying decision. Given a specific article (e.g. a music CD), the agent requests its price at different on-line shops and presents the result to the user. Unfortunately for on-line retailers, the shopping agents compare on price only, ignoring value-added services by which retailers distinguish themselves from competitors. More advanced agents can be used for distributed negotiation, e.g. dynamically negotiating a price for a product. Kasbah (Chavez and Maes [25]) is a multi-agent system that supports distributed negotiation. A user that wants to sell goods creates a *selling agent* and a user that wants to buy goods creates a *buying agent*. The agents operate in a common environment (the *marketplace*) that allows them to interact. Each agent has a specification of the item to buy or sell, an initial desired price and an ultimate date at which the transaction must be completed. Furthermore, a selling agent has a lowest acceptable price and a buying agent has a maximum acceptable price. The ‘negotiation’ between a selling agent and a buying agent depends on the mechanism by which a selling agent lowers its price and a buying agent increases its price over its given time frame. When the price of a selling agent equals the price of a buying agent, a deal can be made.

The earlier mentioned weak notion of agency can be extended with other attributes, like *mobility*. Mobile agents are software agents possessing all earlier mentioned characteristics, but with the additional ability to move from one computer system to another. If an agent is relatively small, it can be more efficient to transport the agent instead of the data. As soon as an agent is transported, it can operate disconnected from its client (see Figure 10). This makes a mobile agent ideal for clients that do not have a permanent network connection, such as users of mobile computers.

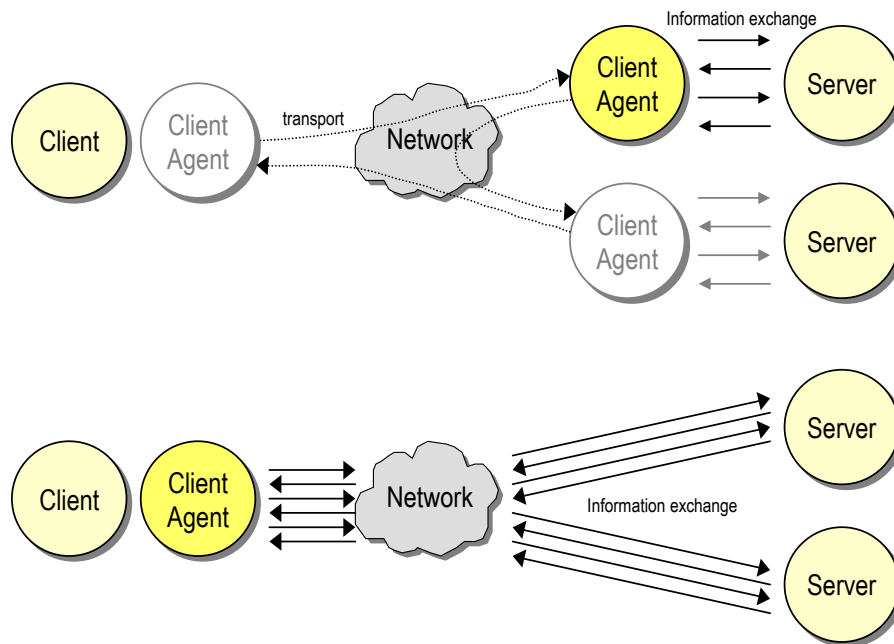


Figure 10 Mobile agents (above) versus immobile agents (below)

The architecture of the environment in which mobile agents can operate is illustrated in Figure 11 (Dalmeijer et al [31]). A ‘dock’ is a software component that acts as a front-end for local systems and allows mobile agents to anchor and perform their job. Docks provide agents with facilities to communicate with local systems and other agents and provide a mechanism for being transferred to others docks. For agent communication to be successful, agents, docks and local systems need a common ontology. This is why efforts are being made to develop a standard Agent Communication Language (ACL). An example of an ACL is the Knowledge Query and Manipulation Language (KQML) (Finin et al [52, 53]), which is both a message format and a message-handling protocol to exchange knowledge among agents. Each KQML message has a content, composed in a language of choice, and a performative. The performative is chosen from a set of mutually agreed speech acts, e.g. ‘tell’, ‘achieve’, ‘ask-if’, and ‘deny’.

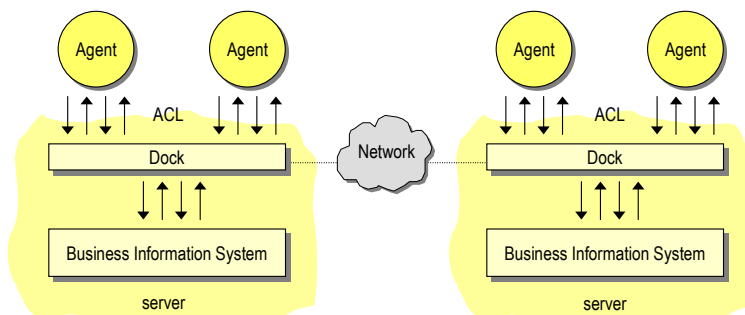


Figure 11 Architecture of the mobile agent environment

1.4.6 Other related projects

The area of electronic contracting is subject of many research projects. This section discusses a number of influential projects.

COSMOS

The aim of the COSMOS project is ‘to develop a support platform for business transactions across the Internet based on a generic contracting service which enables its users to negotiate, sign and settle electronic contracts across the Internet without leaving a uniform and flexible system environment’. The theoretical foundation for the COSMOS system is given in [58, 93]. According to [58], the COSMOS architecture consists of the components shown in Figure 12. The offer catalogue component is used to store service offerings of market participants in a structured way. Brokers are used to match offers in the catalogue with a quality of service specification given by a market participant. Output of a broker is references to those market participants that match the quality of service specification best. The negotiation support can be understood as the collaborative editing of a contract as a structured document. A contract that is edited collaboratively can be signed by the participants by mechanisms known from public key systems. Finally, workflow execution supports and monitors the activities that have to be performed according to the signed contract.

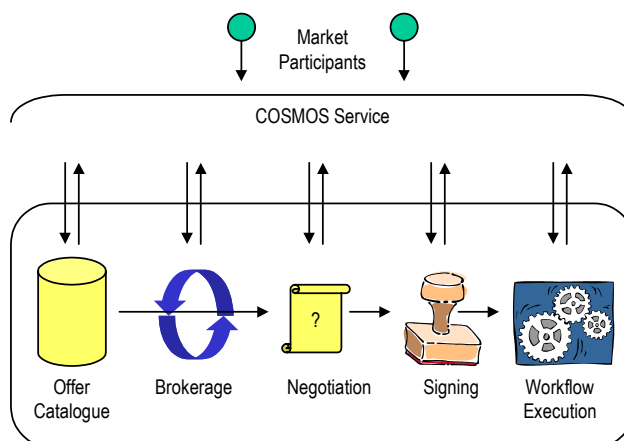


Figure 12 COSMOS architecture (after [58])

MeMo

The MeMo project (Mediation and Monitoring electronic commerce) (www.abnamro.com/memo) is another example of an initiative that aims at developing a platform to facilitate and mediate business-to-business commerce. The objective of the project is to develop an environment that will serve as an Electronic Commerce Broker Service (ECBS) for small and medium sized companies. The ECBS contains a partner searching mechanism that allows companies to quickly locate potential business partners. Furthermore, a multi-lingual negotiation module is available to facilitate the negotiation process in which a range of best-practice negotiation protocols can be used. Finally, the ECBS supports also the settlement of financial obligations.

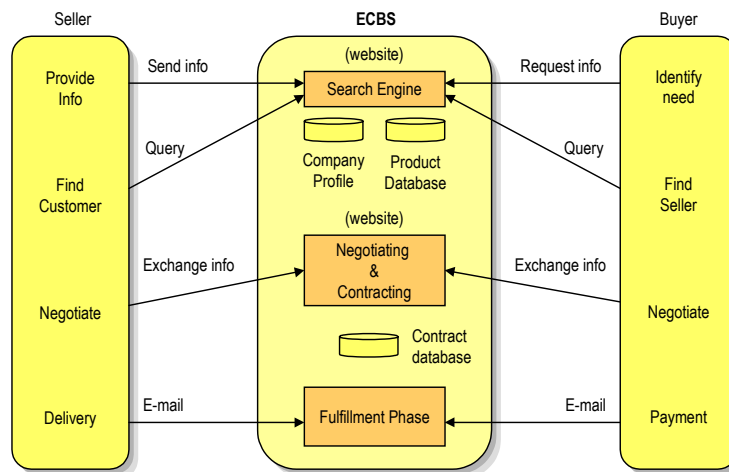


Figure 13 MEMO architecture

WISE

The WISE project focuses on the enactment of inter-organisational business processes, for which the term ‘virtual business process’ is used. A virtual business process is a process in which the tasks correspond to entire subprocesses at different organisations that together form the ‘virtual enterprise’. According to Lazcano et al [85], the WISE project aims at providing a software platform that supports (i) process definition functions for modelling virtual business processes, (ii) process enactment to execute virtual business processes during which subprocesses in different organisations are invoked, (iii) monitoring and analysis and (iv) coordination and communication.

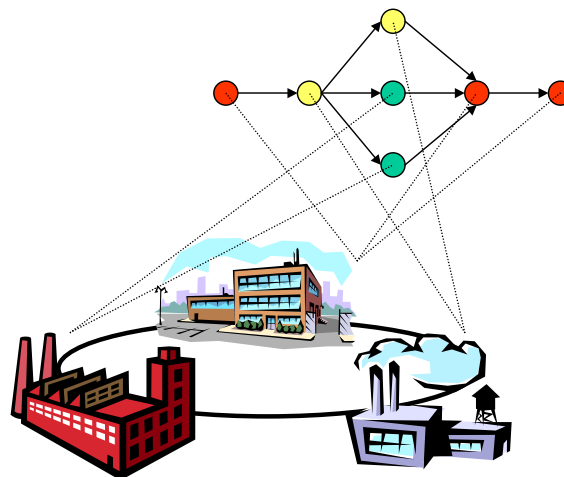


Figure 14 WISE architecture (after [85])

CrossFlow

A research project addressing the inter-operability between workflow engines is the *CrossFlow* project [30] (www.crossflow.org); a European research project aiming at cross-organisational workflow support for virtual enterprises (Grefen et al [57]). Currently, the functionality of workflow management systems is often confined to the boundaries of a single organisation. The objective of the CrossFlow project is to define additional functionality required for inter-organisational workflows. The deliverables of the CrossFlow project include a detailed architecture of inter-organisational workflow. Furthermore, a framework for describing service

contracts will be developed. Service clients and service providers can use this framework to define their requests and offers, hereby allowing dynamic matchmaking of service contracts. Finally, the CrossFlow project will deliver software components that can be used in combination with existing workflow management systems to establish a virtual enterprise.

1.5 Software engineering

Like many other complex technologies, software has a life cycle that can be divided into five phases: specification, construction, testing, integration and maintenance. Although the individual phases remain in place, the content of the phases has shown a rapid evolution over the past years. Some of the developments in the specification and construction phases relevant to the research will be described in this section. Thereafter, we will discuss recent developments in component based software engineering.

1.5.1 Specification

A specification is a description of the structure and behaviour of a system, encompassing both a data and a process perspective. The practice of developing information systems has revealed a structural weakness in the specification techniques for software. This weakness is the informal way in which specifications are developed. Vague specifications often lead to information systems that do not meet the expectations and requirements of the users. If unclear parts of a specification force constructors to fill in the details themselves, the resulting product may deviate from the system as it was meant to be originally. Errors in the specification may be revealed only in the testing phase, causing high costs to be corrected. These observations revealed the need for better methods and techniques for specification of software. A 'better' technique is a technique which allows the designer to be more precise in his specifications and which allows verifications and validations to be performed before the construction starts. High-level Petri nets described by Jensen [77] and Van Hee [61] are an example of a formal specification language used to define the dynamic behaviour of a system. The formalism allows a very precise specification on one hand and the possibility of automatic verification and simulation on the other hand. The ExSpect [50] tool is an example of software with which a system can be defined as a high level Petri net and can be simulated. Another type of specification languages is based on process algebra's. An example of such a specification language is LOTOS [47], which is adopted as ISO standard for modelling communicating processes. The communication between processes is modelled by offering events on gates. The behaviour of each process is defined by expressions.

A recent approach to software specification is the Object-Oriented (OO) approach. The object-oriented paradigm is based on the concept of an *object*, which combines both data structure and behaviour in a single entity. Object oriented approaches have been described by for instance Booch [22] and Rumbaugh et al [107]. The oldest Object Oriented ideas go back to O. Dahl with its language Simula. Applying the object-oriented paradigm to software design means that software is organised as a collection of discrete objects that incorporate both data structure and behaviour. This is in contrast with conventional programming in which data structure and behaviour are only loosely connected. Objects have attractive features, of which the most appealing are: *encapsulation*, *classification*, *polymorphism* and *inheritance*. *Encapsulation* consists of separating the external aspects of an object, which are accessible to other objects, from the internal implementation details of the object, which are hidden from other objects. Encapsulation prevents a program from becoming so interdependent that a small change has massive

effects. *Classification* means that objects with the same data structure (attributes) and behaviour (operations) are grouped into a *class*. An object is said to be an *instance* of a class. The implementation of an operation by a certain class is called a *method*. *Polymorphism* means that the same operation may behave differently on different types of objects. The classes 'circle', 'square' and 'triangle' for instance may share a 'draw' method, which can be implemented differently for each class. Finally, *inheritance* is the sharing of attributes and operations among classes based on a hierarchical relationship. A class can be broadly defined and then refined into successively specialised subclasses. Each subclass incorporates, or inherits, all of the properties of its superclass and adds its own unique properties. The ability to factor out common properties of several classes into a common superclass and to inherit the properties from the superclass can greatly reduce repetition within programs and is one of the main advantages of an object-oriented system.

Efforts have been made to unify different OO specification languages into a unified language. The result of these efforts is the *Unified Modelling Language* (UML) [101], which is presented as a language for specifying, constructing, visualising and documenting the artefacts of a software-intensive system. UML tries to fuse the concepts of Booch [22], OMT by Rumbaugh [107] and OOSE, incorporating the object-oriented community's consensus on core modelling concepts. The focus of UML however is aimed at a common meta-model and common notation, not on the development methods in the context of which the language is used.

Design patterns are another example of efforts aiming at improving the quality of software design by reusing existing knowledge. The concept of design patterns originates from the world of architecture. A design pattern is a solution to a generic design problem and has the ambition to capture the knowledge and experience of the one who created the solution and make it available to other designers who might encounter the same problem. A design pattern is more than just a solution, it gives insight in the opposite forces that create the design problem and it shows why the solution is a good solution. Software patterns first became popular with the book of Gamma et al [55]. Others, like Florijn et al [54], reported successful use of patterns in large software engineering projects.

1.5.2 Construction

The construction methods for creating software have shown a significant evolution too. In the construction phase, four basic construction techniques are available to the software engineer. These techniques can be characterised with the key words: *programming*, *generation*, *configuration* and *assembly*. The *programming* method consists of developing software by writing code in third or fourth generation programming languages based on specifications. Klint and Verhoef [83] describe techniques used by programmers, among which: (i) information hiding, (ii) subroutine libraries, (iii) code scavenging and (iv) tools. Code scavenging is a frequently used, but largely under-documented technique: when a programmer needs to implement a certain function, he searches sources of existing programs for code that is comparable to the one that is desired. Tools, as for editing, compiling, debugging, testing and configuration and version management are frequently used and had a large impact on software quality. An important, but often underestimated, issue is the maintenance or renovation of existing software systems. Klint and Verhoef [83] report that only 30% of the total costs of a system are devoted to its initial construction. The remaining 70% are spent on maintenance and adjustments to new requirements. Klint and Wijers [82] describe the problems that can be encountered during renovation of software systems and propose techniques for successful renovation of software systems.

The *generation* method is based on the automatic generation of software from (formal) specifications. Standard case-tools are already on the market for the generation of classical database applications (data entry and retrieval). More advanced specification languages and tools are being researched in laboratory environments, but these methods still require much knowledge to be used.

The *configuration* method aims at creating a specific information system by configuring the parameters of a generic information system. Examples of generic systems that are configured to the needs of a specific situation can for instance be found in the area of enterprise resource planning (ERP) systems.

Finally, the *assembly* method consists of assembling a number of standard components into one information system. A software engineer can create a component by himself, or he can buy the component. Advantages of using components developed by others can be found in reduction of costs (the investments for developing the component are shared by a number of parties), reduction of time (buying a component takes less time than developing one) and finally quality improvement (standard components have already passed the testing phase). The use of components brings about a number of new issues that require the attention of the programmer too. First, there is the question of how to find the (best) component. Various sources of information can be used for this purpose. The Internet is an ideal medium for publishing catalogues of generic components, closing the deal, distributing the software and making the payment. Second, since components are created by a third party an organisation must trust the component well enough to use it in his business information system. This introduces the concept of 'trusted components'. Finally, a programmer may be confronted with a component from which the behaviour is only known partially.

1.5.3 Component based software engineering

As we have seen in discussing the assembly construction technique, components are becoming increasingly important in software engineering. Therefore, we will now discuss component based software engineering in more detail. A generic software component is a piece of software that offers well-defined 'services' to client applications. A component has an *operational* interface and a *management* interface. The operational interface with a client application is a transaction that consists of one or more messages. A generic component must be able to support concurrent transactions with different client applications. The specification of a generic software component includes the transaction protocol (the possible orders of messages) and the data types of each possible message type. The management interface consists of a configuration function and a monitor function. The configuration parameters exchanged on the configuration interface can have a simple structure like a table or a more complex structure like expressions or diagrams. The monitor function lets a client application query the internal state of the software component.

The search for standard components is not new. The historic development in recognising generic functions is described by Van der Aalst and Van Hee [12]. In the seventies data management was recognised as a generic function and data base management systems (DBMS) emerged, releasing the application from data management. In the eighties a similar thing happened with user interfaces which lead to the emergence of user interface management systems (UIMS), releasing the applications of user interfacing. The development of the nineties was the recognition of the support of business processes as a generic function. Because of this,

workflow management systems (WFMS) emerged, as a generic software component to support the logistics of business processes.

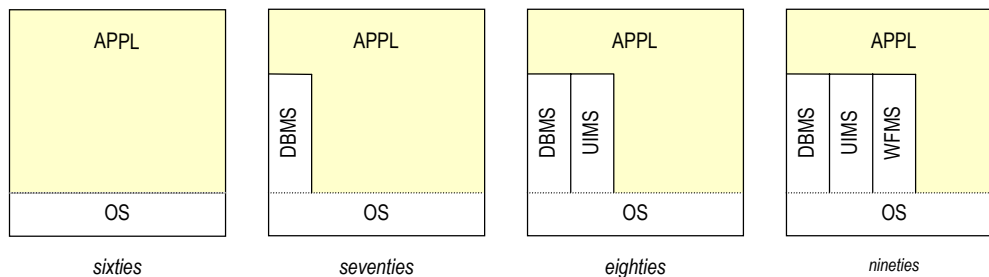


Figure 15 Historic development in standard components

The use of the concept *component* in the software industry has an analogy in many other production sectors. The electronics industry for instance is based on the existence of a large amount of standard components like semi-conductors, microchips, etc. Each component type is strictly defined, allowing the same type of component to be manufactured by different parties and offered to the market at competitive prices. The examples where a manufacturer does not produce a single component by itself, but only configures and assembles components bought from others are not rare.

Bergstra and Klint [15] proposed the *Toolbus* as a means for the co-operation of individual components. They suggest a clear distinction between *computation* (performed by the components) and *co-ordination* (performed by the Toolbus). The Toolbus architecture is shown in Figure 16. There are m components (C_1, \dots, C_m) and a Toolbus in which n parallel processes (P_1, \dots, P_n) are active. A component does not communicate directly with other components, but only with the Toolbus. The initiative for communication can be taken by either the component or the Toolbus.

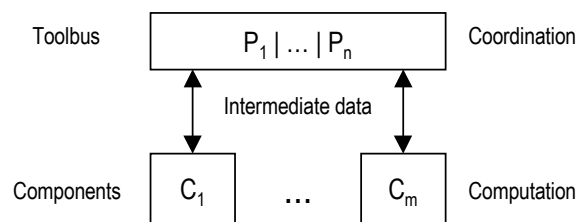


Figure 16 The Toolbus architecture

The object-oriented paradigm in software design is matched with object-oriented technology for the construction of software. Object-oriented programming languages like C++ and Java make it possible to make an almost seamless transition from specification to software. A C++ or Java class can be seen as a component that can be assembled seamlessly with tailor-made classes into one information system. Every Java programmer who uses the standard classes in for instance the `java.awt` package is assembling a standard component. However, it should be noted that the use of standard classes is limited to one programming language. One can not assemble a standard C++ class with one or more tailor-made Java classes.

The Java language has powerful facilities for the development and assembly of standard components. A part of the Java framework is the *Java Beans* specification by Sun [111], a component based software model for building and using dynamic Java components. A Java Bean is

designed to be assembled with other components via a builder tool. One of the interesting characteristics of a Java Bean is *introspection*, the ability to expose its properties, events and methods to other objects both at runtime and in the builder environment (design time). Support for *customisation*, by which the appearance and behaviour of the Java Bean can be customised in the design environment, is another appealing feature of Java Beans.

The area of communication between objects sitting on different platforms, connected via a network is addressed by the Object Management Group (OMG). Standards as these are very important for the implementation of distributed applications on a wide scale. The OMG (www.omg.org) has proposed a common object communication bus called CORBA [100] (Common Object Request Broker Architecture) which is a common messaging standard for distributed objects. The Object Request Broker (ORB) is the middleware that establishes the client-server relationships between objects. Using an ORB the client can transparently invoke a method on a server object, which can be on the same machine, or across a network. The client does not have to be aware of the location of the server object, nor the programming language used for the implementation, nor any other aspect that is not reflected in the object's interface.

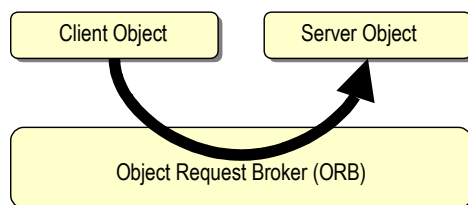


Figure 17 Communication between objects via the ORB

A technology for component based software development that is widely used on the MS-Windows platform is the Microsoft *Component Based Object Model* (COM) [94]. The COM standard is an object-based programming model designed for interoperability between binary software components that can be developed in different software languages. COM defines and implements mechanisms that allow applications to connect to each other as *software objects*, an instantiation of a class that conforms to the COM standard. A COM object is accessed only via its *interfaces*, a set of strongly typed semantically related functions (called member functions of that interface). Microsoft extended the COM technology to support distributed computing with the DCOM (Distributed COM) standard [95]. Distributed applications consist of components that reside on different hardware components, connected by a network.

1.6 Research

1.6.1 Research problem and objectives

Electronic commerce is expected to have an enormous impact on organisations, both on the business processes and on the information systems that support the business processes. These expectations can only become true when high-quality standard software components for electronic commerce are available on the market. Although a significant number of such standard software components is already available, electronic commerce components are still in a diverging stage and there is little consensus on concepts, architectures, building blocks, etc. [17]. However, as we have seen before, many research activities are devoted to the area of electronic contracting. This research contributes to the existing research and focuses on a specific sub-

class of service contracting processes. The business objective behind this research is to improve the competitiveness of organisations by providing an optimal support of these service contracting processes with ICT. More specific, the objective is to provide a generic software component for service contracting that can be assembled with business applications seamlessly and requires minimal configuration time and maximal flexibility in the type of service contracting processes supported.

1.6.2 Research scope

The term ‘electronic contracting’ is used for a variety of phenomena. This research is focused on a specific part of this area, which is demarcated by the following characteristics that define a class of service contracting processes.

- **Loosely coupled organisations**

We assume a *loosely coupled* relationship between service clients and service providers. This means that all communication is performed by exchanging structured messages, of which only the data types (static aspects) and constraints on the sequence of message types (dynamic aspects) are mutually agreed. In particular, we assume no central brokerage or mediation service between service clients and service providers. Furthermore, we assume no knowledge of each others business processes for the participating organisations.

- **Buyer side only**

Electronic contracting of services always involves a buyer (client) and a seller (provider). Although these parties communicate via a common message protocol, they execute different processes. *This research focuses on the part of the process executed by the buyer (service client) only.* Therefore, the seller (service provider) is treated as a black box, of which only the external interface (transaction protocol) is known. The software component that will be developed is therefore not a complete solution for electronic contracting (buyer side + seller side), but can only be used for the buyer side.

- **N required services, M available providers**

A service contracting process is performed for a business case in the enterprise information system. This research focuses on the more complex service contracting processes where each business case requires N different services to be contracted, for which M different service providers are available.

- **Dependencies between services**

Furthermore, *we assume dependencies between required services.* We consider two types of dependencies. First, there can be constraints on the *order* in which required services must be contracted. For example, service B must be contracted when service A has been completed (sequential relation). Or: service B must be contracted only if service A could not be contracted (alternative relation). The second dependency type involves the *details* of the required services. For example, a logistic chain requires two services: sea transport from a port of departure to a port of destination and road transport from the production plant to the port of departure. Clearly the ‘place of delivery’ of the road transport service must be equal to the ‘port of departure’ of the sea transport service. Furthermore, the ‘date/time of delivery’ of the road transport service must be before the ‘date/time of loading’ of the sea transport service. Therefore, the details of the road transport service can only be defined after a contract for the sea transport service is established.

- **Completely automated processes**

The research focuses on completely automated service contracting processes. Therefore, the specification of the service contracting process must be highly structured and computer interpretable. All decisions must be made automatically, based on configuration parameters defined in design time.

1.6.3 Research approach

The literature overview in Section 1.4 showed different approaches to electronic contracting. The approaches differ in postulates and modelling techniques. One of the approaches is the LAP approach that uses deontic logic as formalism and is heavily used in research on intelligent agents. A different approach views contracting processes as interorganisational workflows and uses Petri nets as modelling technique. This research chooses the workflow approach based on Petri nets as starting-point. By making this choice we do not claim that the workflow approach is superior to the LAP approach. Instead, we have made this choice for the following reasons.

- Workflow management techniques have been successfully applied to internal business processes. Since business processes are becoming inter-organisational increasingly, the application of workflow management techniques to inter-organisational processes is an obvious choice.
- Workflow management techniques have proven to be a good solution for repeating, well-structured and potentially long-running processes. The character of the demarcated class of service contracting processes has many similarities with this kind of processes.
- Workflow management techniques are increasingly integrated in software tools for electronic messaging. Apparently, the market recognises the usefulness of workflow management in combination with electronic business.

1.6.4 Research questions

We will now formulate the research questions that will be addressed in this dissertation. First, we need a clear understanding of the underpinning concepts for service contracting processes. We have chosen the approach of modelling a service contracting process as a workflow. Therefore, our first research question is:

Research question 1:

“What is a suitable underlying conceptual framework to model the demarcated service contracting processes via workflow management techniques?”

The hypothesis behind this research is that service contracting is a generic business function, for which a generic software component can be created. Clearly, although designed as a generic system, each software component has its limitations on the types of service contracting processes that can be handled. Since service contracting is still in an early stage and there is no common understanding of requirements to a service contracting component, it is important to

have a software component that can be adapted to changing requirements easily. Our second research question is therefore:

Research question 2:

“What is a suitable logical architecture of a service contracting software component that maximises the flexibility of the software component to handle different types of service contracting processes?”

One of the results of the research is a generic software component for service contracting, the ‘Contracting Agent’, which can be configured to a specific business situation. When it comes to the design of the configuration parameters and the configuration user interface, we are confronted with conflicting goals: *ease-of-use* and *flexibility*. On one hand we want to maximise the ease-of-use, especially in the configuration of the component. This goal can be obtained by adding domain knowledge to the component, hereby inevitably limiting its flexibility. On the other hand however, we want to maximise the flexibility of the component to make it suited for the widest class of service contracting processes possible. We therefore formulate our third research question as follows:

Research question 3:

“How can domain knowledge of service contracting processes be used in the design of the configuration parameters and the configuration user interface of the service contracting software component in order to bring maximum ease-of-use while maintaining the flexibility of the component to handle a wide variety of contracting processes?”

A short time-to-market is one of the critical success factors for software components. One of the lessons learned from component-based software engineering is to re-use existing software components maximally. Our fourth research question is therefore:

Research question 4:

“What is a suitable technical architecture of the service contracting software component that utilises existing standards and existing software components maximally in order to minimise the effort to develop the component and to maximise maintainability?”

1.7 Outline

The structure of this dissertation is as follows. This chapter describes the background of the research, the research problem and the research approach. The rest of this dissertation consists of the following chapters.

- **Chapter 2: Conceptual framework**

This chapter addresses the first research question and provides a conceptual framework for the demarcated class of service contracting processes. We start with a brief introduction into inter-organisational systems based on contracting and define basic concepts like ‘workflow’, ‘service’ and ‘business transaction’. Furthermore, we define the term ‘service contracting’ and its position in inter-organisational systems. Next, we provide a conceptual framework for the agreements that have to be made between service clients and service providers in order to establish a loosely coupled collaboration successfully. Here after, we propose a meta model for the contracting requirements of an outsourced task in the internal workflow. Next, we define the service contracting process as a workflow, propose standard building blocks from which this workflow can be composed and define correctness criteria to which a contracting workflow must adhere. Finally, we illustrate the concepts by a use case. Modelling techniques used in this chapter are high-level coloured Petri nets to model the contracting workflow and functional data models to capture the data structure of tokens in the workflow.

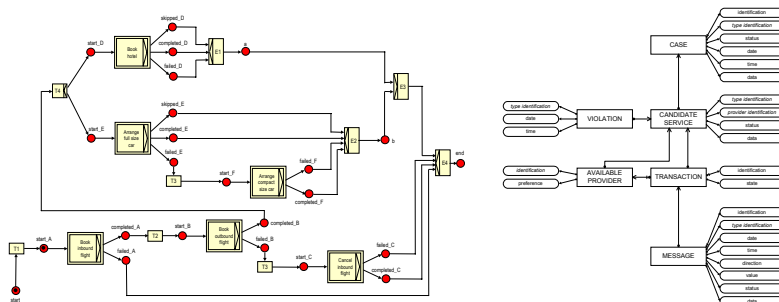


Figure 18 Illustration of modelling techniques used in Chapter 2

- **Chapter 3: Logical architecture**

The objective of this chapter is to address the second and the third research question and to provide a logical architecture for a software component that executes the service contracting processes defined in Chapter 2. The software component will be designed as a workflow application with a clear separation of execution and control. One of the issues in this chapter is to recognise generic tasks in the service contracting processes that can be housed in autonomous applications triggered by the workflow engine. Another issue is the transformation of the conceptual contracting workflow net into an implementation contracting workflow net. Modelling techniques used in this chapter are Petri nets to model the components and interfaces and functional data models to capture the data structure of data stores and interfaces.

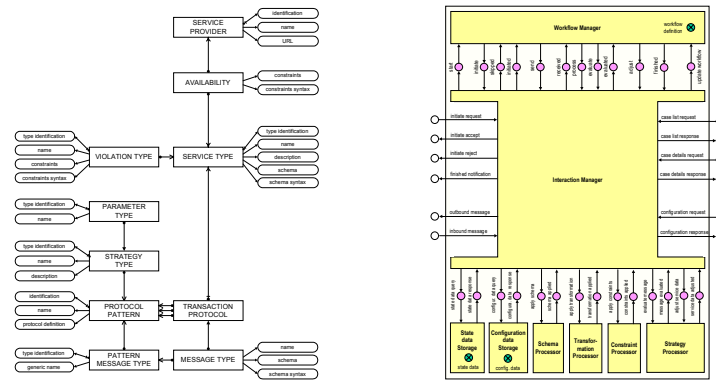


Figure 19 Illustration of modelling techniques used in Chapter 3

• **Chapter 4: Technical architecture**

This chapter addresses the fourth research question and discusses the transformation of the logical architecture into a technical architecture consisting of commercial-off-the-shelf software components and custom-made software components. Modelling techniques used in this chapter are software component architectures, XML document type definitions, Ex-Spect models, relational database schema's and screen layouts.

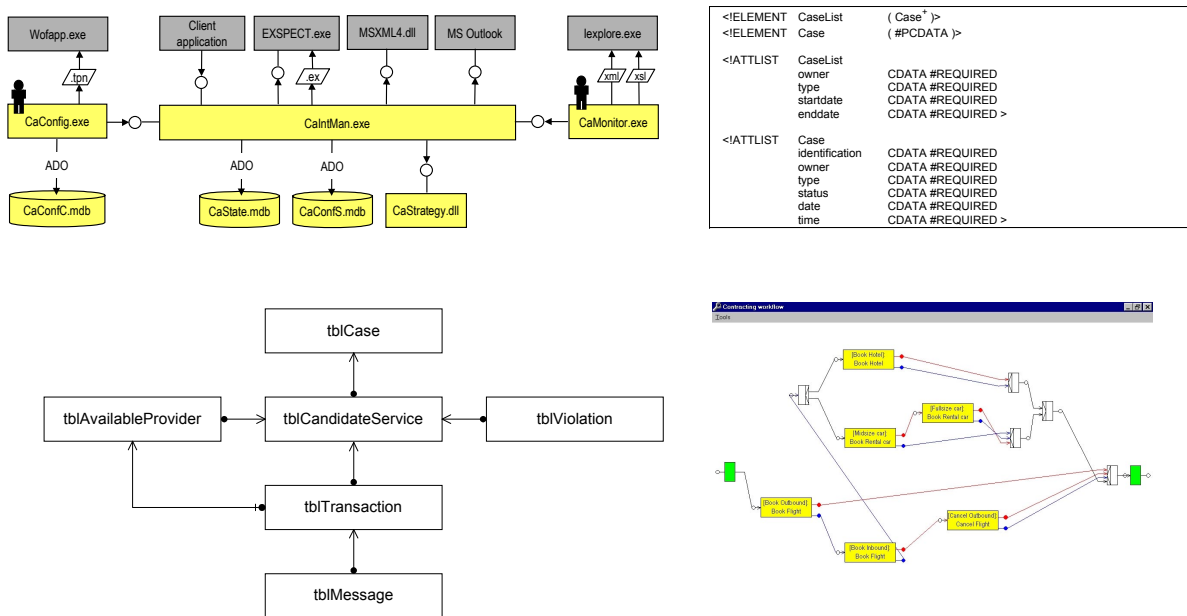


Figure 20 Illustration of modelling techniques used in Chapter 4

• **Chapter 5: Evaluation and conclusions**

A final reflection on the research is given in Chapter 5. We will discuss the achievements and draw conclusions. Furthermore, we will compare our work with the work of others. There after, we will point out directions for future research.

2. A conceptual framework for service contracting

This chapter addresses the first research question and provides a conceptual framework for service contracting processes. We start with a brief introduction into inter-organisational systems based on contracting in Section 2.1 and define the underpinning concepts ‘workflow’, ‘service’ and ‘business transaction’ in Section 2.2. Here after, we introduce the term ‘service contracting’ in Section 2.3. A detailed specification of the conceptual model is given in sections 2.4-2.6. Finally, we illustrate the conceptual framework by a use case in Section 2.7.

2.1 Inter-organisational systems

An organisation can be seen as a system composed of *actors* and *objects*. Actors are the active components, they consume and produce objects, which are the passive components. Actors can for instance be human beings, computer applications, machines, vehicles, etc. Objects can be material objects such as containers or abstract objects such as insurance and money. An organisational system can be broken down into an *information system* and a *business system* which have a relation to each other. In the business system, objects have an economical value and are often of a physical nature. The objective of the business system is to produce objects with an economical value higher than the economical value of the consumed objects. The information system is the part of the organisational system that deals with information objects only and uses this information to control the business system.

We speak of an *inter-organisational system* if the systems of two or more organisations are connected. In this research, we focus on inter-organisational systems that emerge from outsourcing of work. Outsourcing of work implies the exchange of physical and/or abstract objects between two business systems. In order to control the exchange of objects between the business systems, information objects are exchanged between the corresponding information system. Figure 21 illustrates these concepts.

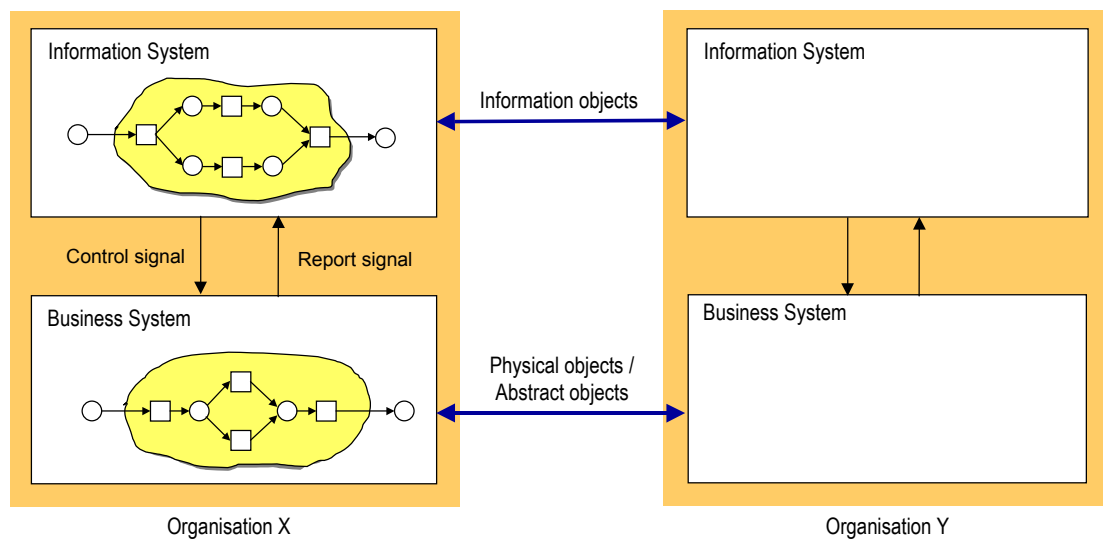


Figure 21 Structure of an inter-organisational system

Formalisms used

The objective of this chapter is to provide a conceptual framework for service contracting. We will use the formalism of high-level coloured Petri nets to model processes and we will use functional data modelling to capture data structures. These formalisms are discussed in Appendix A.

2.2 Underpinning concepts

2.2.1 Workflows

We will now focus on modelling the structure and behaviour of information systems. We will follow the wording used by Van der Aalst and Van Hee [12] and the WorkFlow Management Coalition [141].

The work that most organisations do is often *case* based and therefore of a discrete nature. Examples of cases are: a purchase order from a customer, a patient in a hospital, the repair of a car in a garage, etc. Most organisations focus on a limited set of activities in which they excel and handle cases with specific characteristics only. A set of cases with identical structure that can be processed identically is called a *case type*. For example, the case type ‘container road transport’ is the set of all cases that require a container to be transported from one location to another by truck. Although each case involves a different container and has different transport details, it is described by a common data structure and is handled by a common process in the information system. Case types are part of the *structure* of the system whereas cases are part of the *behaviour* of the system.

Definition: *case, case type*

A *case* is the information system representation of a discrete piece of work in the business process of an organisation. A *case type* is a class of cases, with a common representation and common processing in the information system.

A case has a unique *identification* by which it can be distinguished from other cases and is further defined by information elements called *case attributes* which together are called the *case data*. In the example of the purchase order the case data could consist of the customer name and address, the article number, ordered quantity and the requested delivery date. The case data consists of *objects* and *attributes*. An object is described by one or more attributes and one attribute belongs to one object. Objects are organised in a tree structure, in which one object is the root of the tree and all other objects are nested below another object. An object belongs to an *object type* and an attribute belongs to an *attribute type* and has an *attribute value* too. Each case type has a fixed set of object types and attribute types by which the structure of its case data is defined. Each attribute type defines constraints on the attribute value (e.g. data type).

Definition: *case attribute, case data*

A *case attribute* is an information element that describes a property of a case in the information system. The set of all case attributes of a case is called the *case data*. The structure of case data is defined by the case type entirely.

The work that is performed on a case can be divided in one or more *tasks*. A task is an indivisible unit of work and is therefore executed entirely or not at all.

Definition: *task, work item, resource, activity*

A *task* is an indivisible unit of work. A *work item* is a task that must be performed for a specific case. A *resource* is an entity (person, machine or computer application) that is required for executing a work item. An *activity* is the execution of a work item by a resource.

The work to be performed on a case consists of executing one or more tasks. A *workflow* defines the tasks that must be performed on a case and the conditions to the order in which they must be performed. Examples of such conditions are sequencing, parallelism, choice, iteration, etc. All cases that belong to the same case type are handled by the same workflow. A workflow however can handle one or more case types.

Definition: *workflow*

A *workflow* defines the tasks that must be performed on cases of one type and the conditions to the order in which they must be performed.

The actual execution of a workflow requires triggering mechanisms to start the execution of tasks. Three different triggering mechanisms can be distinguished. First, the *resource* can take the initiative to start the execution of a task. Second, an *external event* can trigger the start of the execution of a task. An example of an external event is the receipt of an EDI-message. Finally, a task can be triggered by a *clock event*: a certain amount of time elapsed or a time is reached.

The relationships between the concepts that are introduced in this section are shown in Figure 22. We use the formalism of functional data modelling, discussed in Appendix A.

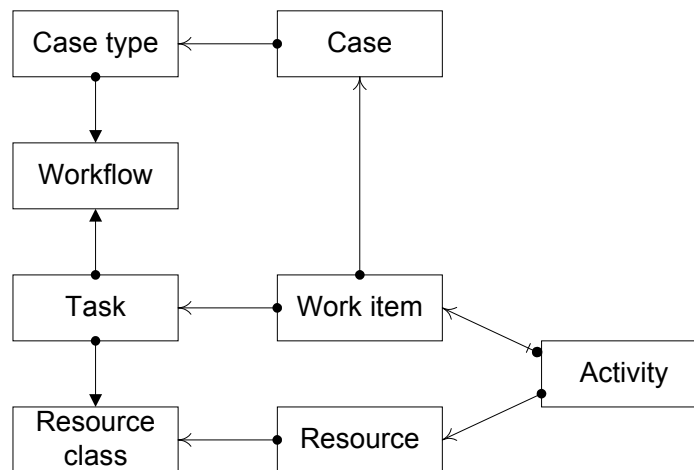


Figure 22 Object model of workflow concepts

Workflow nets

We will now discuss modelling techniques for the workflow concepts defined before. First, a workflow is modelled as a workflow-net (WF-net), a high-level Petri net with a special structure. A formal definition of WF-nets can for instance be found in [5]. We will now give an informal definition of WF-nets. A high-level Petri net is called a WF-net if and only if:

- (i) there are two special places i and o , of which i is a source place (no incoming connectors) and o is a sink place (no outgoing connectors);
- (ii) every place and every transition is on a path from the source place to the sink place.

A task is modelled by a transition and the partial ordering of tasks is modelled by places connecting these transitions. A case is modelled as a token and the case data is modelled as part of the colour of the token. The processing of a case starts when the case token is put in place i and ends the moment a token appears in place o . The execution of a task is modelled as the firing of a transition. The routing of a case through the process is defined by the way transitions are connected to each other by places and by the pre- and post-conditions of the transitions. The Work Flow Management Coalition [140] identified the following six routing primitives.

- **Sequential routing:** a segment of a process instance in which several activities are executed in sequence under a single thread of execution.
- **Iteration:** a cycle in the process involving the repetitive execution of one (or more) activity(s) until a condition is met.
- **AND-split:** a point within the process where a single thread of control splits into two or more parallel activities.
- **AND-join:** a point in the process where two or more parallel executing activities converge into a single common thread of control.
- **OR-split:** a point within the process where a single thread of control makes a decision upon which branch to take when encountered with multiple alternative process branches.
- **OR-join:** a point within the process where two or more alternative branches converge to a single common activity as the next step in the process.

These six routing primitives can be mapped onto classical Petri nets as shown in Figure 23.

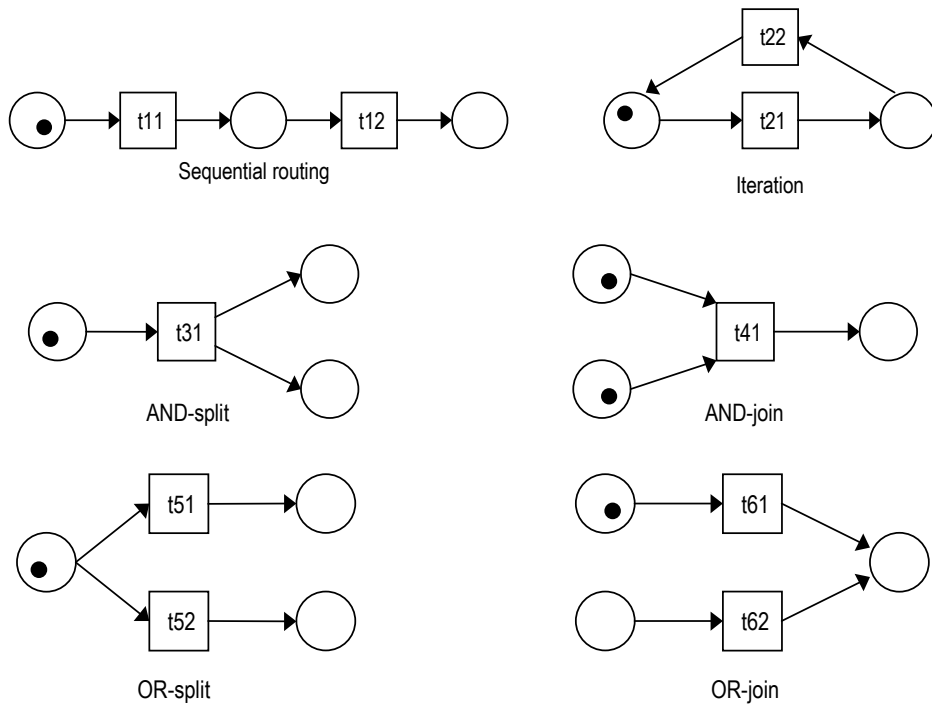


Figure 23 Routing primitives mapped onto classical Petri nets

Instead of the OR-split and OR-join constructs shown in Figure 23 we will use the equivalent high-level Petri net constructs shown in Figure 24.

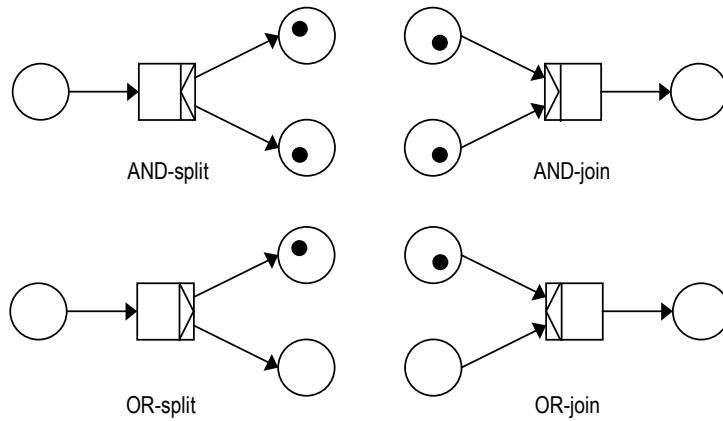


Figure 24 Routing primitives mapped onto high-level Petri nets

Soundness

An important property of WF-nets is called *soundness*. A formal definition of soundness can be found in for instance [9]. We will now give an informal definition of soundness. In this definition, we will refer to state i as the state in which place i contains one token and all other places contain no tokens. Similarly, state o is the state with one token in place o and no tokens in other places. A WF-net is sound if the following requirements are satisfied.

- (i) For any state reachable from state i there exists a firing sequence leading to state o .
- (ii) State o is the only state reachable from state i with at least one token in place o .
- (iii) There are no dead tasks, i.e. for each transition t there is a firing sequence from state i in which transition t fires.

When a process is modelled as a Petri net, cases are modelled as tokens and the case data is modelled as the colour of the token. The colour of a token is in fact defined as a complex and the complex class of a case token is defined by the object model in Figure 25. The complex class contains an entity 'CASE' with attributes '*identification*' and '*type identification*'. The '*identification*' attribute models the unique identification of the case in the information system. The '*type identification*' attribute models the unique identification of the case type to which the case belongs. A 'CASE' entity has a relationship to one or more 'OBJECT' entities. An 'OBJECT' entity has one attribute '*type identification*', which models the identification of the object type to which the object belongs. The hierarchic relationship between objects is modelled by the tree-relationship (see A.1) from entity 'OBJECT' to itself. Finally, an 'OBJECT' entity has a relationship to one or more 'ATTRIBUTE' entities with attributes '*type identification*' and '*value*'. The attribute '*type identification*' models the unique identification of the attribute type to which the attribute belongs, whereas the attribute '*value*' models the value of the case attribute in the case data.

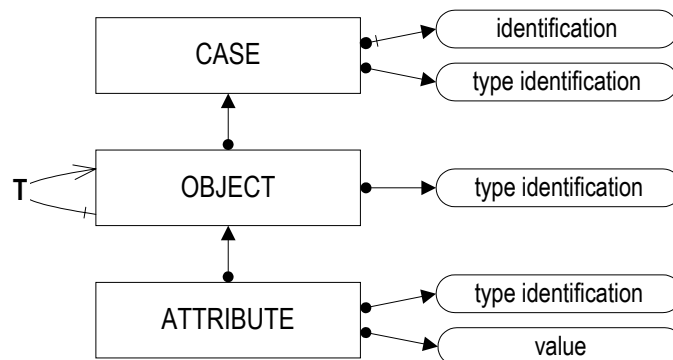


Figure 25 Object model of complex class 'case'

When a case is modelled as a complex, the specification of the case data structure is modelled as a case type complex of which the complex class is shown in Figure 26. The case type itself is modelled by an entity 'CASE TYPE' with attributes '*type identification*', '*name*' and '*description*'. The attribute '*type identification*' models the unique identification of a case type, which is used in a case complex to reference the case type. The attributes '*name*' and '*description*' model the name and description of the case type in clear text. Each 'CASE TYPE' entity has a relationship to one or more 'OBJECT TYPE' entities and the hierarchical relationship between object types is modelled by the tree-relationship from entity 'OBJECT TYPE' to itself. An 'OBJECT TYPE' entity has five attributes: '*type identification*', '*name*', '*description*', '*status*' and '*maximum repeats*'. The '*type identification*' attribute models the unique identification of the object type, which is used in a case complex to reference the object type to which an object belongs. The '*name*' and '*description*' attributes model the name and description of the object type in free text. The '*status*' attribute models the status of an object type in the object type hierarchy, and is either 'required' or 'optional'. If the status of an object type is 'required', each 'CASE' complex must contain an 'OBJECT' simplex of this object type. Otherwise, the 'OBJECT' simplex of this type can be omitted from the 'CASE' complex. The '*maximum repeats*' attribute models the maximum number of instances of the object type that is allowed. Both the status and the maximum number of repeats of an object type are relative to the parent object type. Each 'OBJECT TYPE' entity has a relationship to one or more 'ATTRIBUTE TYPE' entities with attributes '*type identification*', '*name*', '*description*', '*status*' and '*data type*'. The '*type identification*' attribute models the unique identification of the attribute type, which is used in a case complex to reference the attribute type to which an attribute belongs. The '*name*' and '*de-*

scription’ attributes model the name and description of an attribute type in free text. The *‘status’* attribute models the status of an attribute type in the object type, and is either *‘required’* or *‘optional’*. The *‘data type’* attribute models the data type of an attribute. Finally, if the domain of an attribute type consists of a fixed set of values, the *‘ATTRIBUTE TYPE’* entity has a relationship to a *‘CODE LIST’* entity which has two attributes *‘name’* and *‘description’*. These attributes model the name and description of a code list in free text. A *‘CODE LIST’* entity has a relationship to one or more *‘CODE’* entities with attributes *‘value’*, *‘name’* and *‘description’*. The *‘value’* attribute models one of the possible code values. The *‘name’* and *‘description’* attributes model the name and description of the code in free text.

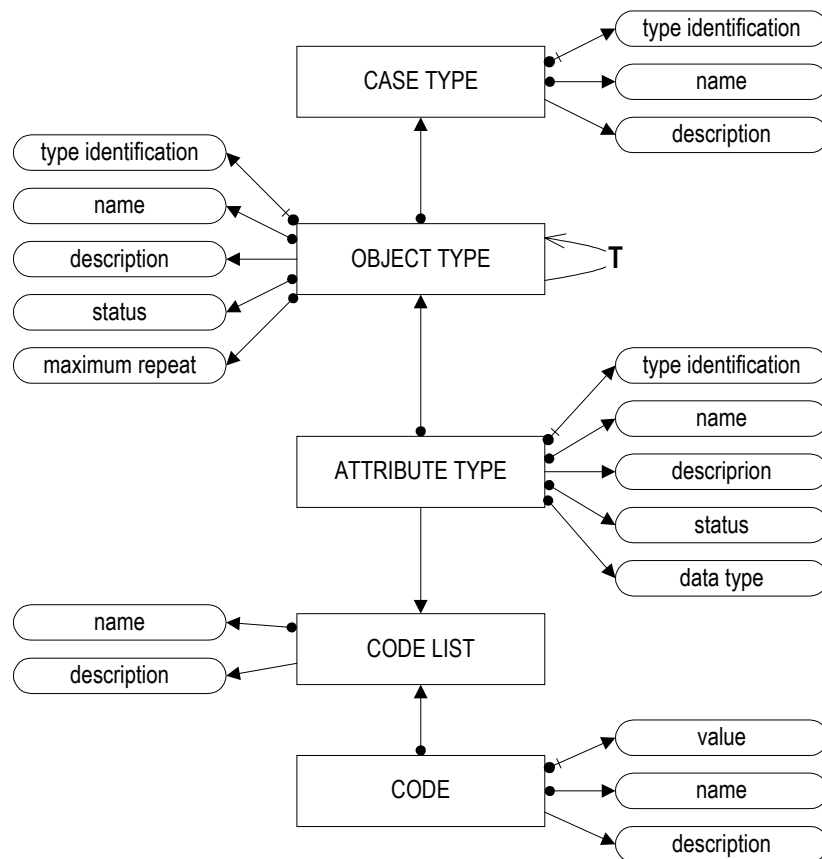


Figure 26 Object model of complex class 'case type'

2.2.2 Services

It is possible that an organisation does not execute a task in its business process itself, but outsources the execution of the task to another organisation. In case of outsourcing of work, we will call the execution of a task by another organisation a *service* that is provided. The service is contracted by the *service client* and executed by the *service provider*. For instance, if a manufacturer outsources the execution of a task 'transport' to a road carrier, the manufacturer has the role of service client and the road carrier has the role of service provider. Services can have a physical nature like the transport of a container or the delivery of materials. A service can also have an abstract nature like the insurance of a consignment, the transfer of money from one account to another or the custom declaration to get approval for exporting goods.

Definition: *service, service client, service provider*

A *service* is the execution of a business process in one organisation with the objective to execute a task in the business process of another organisation. The organisation that executes the work is called the *service provider*. The organisation that contracts the work is called the *service client*.

Outsourcing of tasks shows that the indivisibility of a task depends on the perspective of the observer. If a task is outsourced by a superior to a subordinate, it is indivisible (atomic) for the superior (the task is outsourced entirely or not at all) but the subordinate may look at it as composed of a number of smaller tasks. An example of this is the task ‘transport’ which is outsourced by a manufacturer to a carrier, who in turn sees the work as composed of the tasks ‘loading’, ‘transport’ and ‘discharge’.

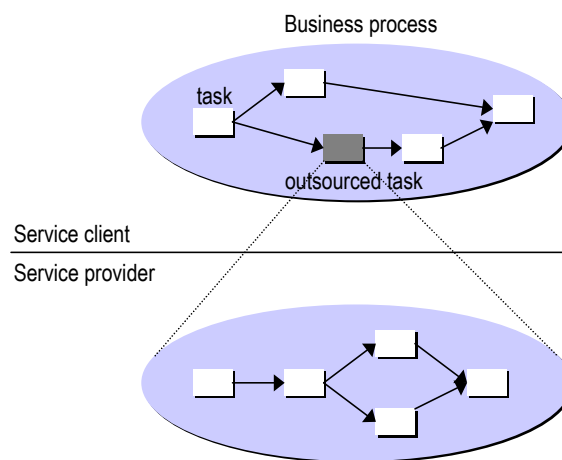


Figure 27 Indivisibility of a task depends on the perspective

In general, organisations are designed to provide a limited set of *service types* to their customers. Their business process is designed to produce this limited set of service types in the most efficient way. A transport company for instance may limit itself to the transport of containers from the northern part of Holland to Rotterdam and back. A shipping line may limit itself to container transport between Rotterdam and Asia. The behaviour of a service provider to service clients is defined by the service types he is able to provide.

Definition: *service type*

A *service type* is a set of services with identical type of results and with identical structure of the service data. One service type can be offered by one or more service providers. Each service provider can offer one or more service types.

Because the information system is responsible for controlling the business system, the information system is also responsible for controlling the services used by the business system. For this purpose, the information system needs a specification of the required service. The information by which a service is represented in the information system is called the *service data*.

Definition: *service attribute, service data*

A *service attribute* is an information element that describes a property of a service in the information system. All service attributes of one service together are called the *service data* of that service.

Service data consists of objects and attributes. The structure of the service data of one service type is defined in terms of the allowed object types and attribute types. Services are modelled as tokens of which the colour is defined as a complex. The complex class of a service token is defined by the object model in Figure 28. The complex class contains an entity ‘SERVICE’ with attributes ‘*identification*’, ‘*type identification*’, ‘*case identification*’, ‘*provider identification*’ and ‘*client identification*’. The attribute ‘*identification*’ models the identification of the service, which must be unique in the relation between service client and service provider. The attribute ‘*type identification*’ models the unique identification of the service type to which the service belongs. The attribute ‘*case identification*’ models the identification of the business case for which the service is required. The attributes ‘*provider identification*’ and ‘*client identification*’ model the identification of the service provider and service client. The rest of the complex class is similar to the case complex class defined Figure 25.

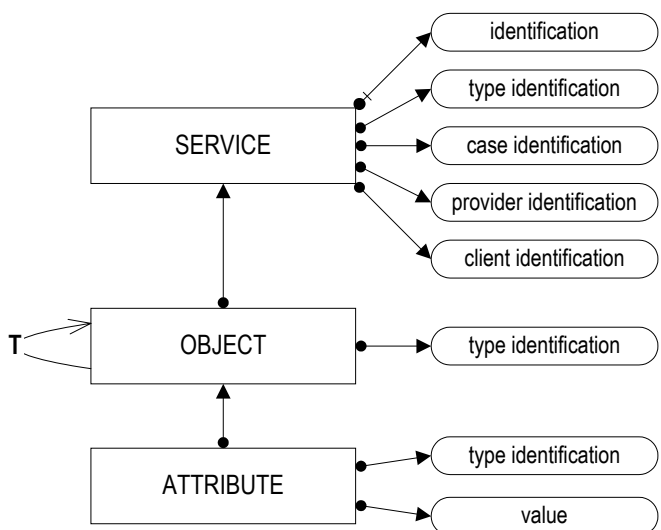


Figure 28 Object model of complex class ‘service’

A complex of complex class service type models the specification of a service type. The object model of the service type complex class is defined in Figure 29. Each service type is modelled by a ‘SERVICE TYPE’ entity with attributes ‘*type identification*’, ‘*name*’ and ‘*description*’. The attribute ‘*type identification*’ models the identification of the service type which is unique in the relation between one service client and one service provider. The ‘*name*’ and ‘*description*’ attributes model the name and description of the service type in free text. The remaining part of the complex class is similar to the complex class that defines a case type from Figure 26.

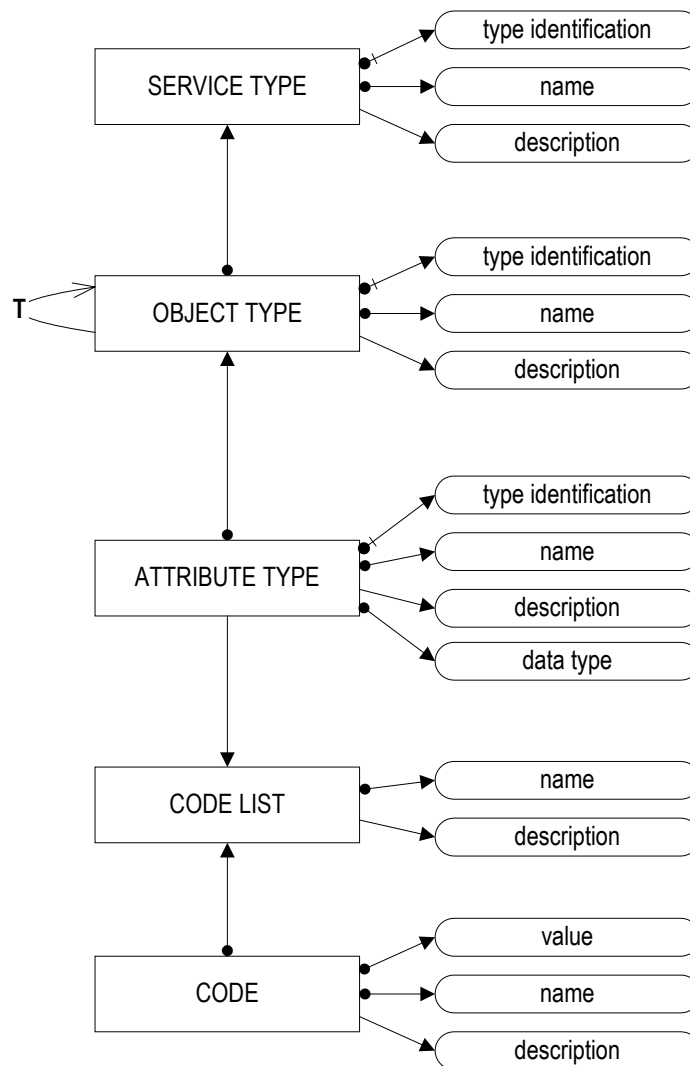


Figure 29 Object model of the complex class 'service type'

2.2.3 Transactions

The execution of services by service providers is controlled by information exchange between the information systems of service client and service provider. Information is exchanged via information objects called *messages*. To control the execution of a service one or more messages are exchanged, e.g. a request, a confirmation, status information and a report. We define a *transaction* as the sequence of messages that is exchanged to control the execution of one service. In this dissertation we will use the phrase 'business transaction' to distinguish from other types of transactions, e.g. financial transactions or database transactions. However, when we use just the word 'transaction', it always has the meaning of 'business transaction'.

Definition: *message, message type, message data*

A *message* is the smallest unit of information exchanged between the information systems of two organisations and consists of one or more information elements, which together are called the *message data*. A *message type* is a set of messages with identical function and identical message data structure.

Definition: *transaction, superior, subordinate*

A *transaction* is a sequence of messages exchanged between a service client and a service provider to control the execution of one service. In each transaction, the service client has the role of *superior* and the service provider has the role of *subordinate*.

There is a minimal set of information elements that must be present in all messages, which we will call the *message control attributes*. This set consists of the following information elements:

- sender identification;
- receiver identification;
- message type identification;
- transaction identification.

The message attributes ‘sender identification’ and ‘receiver identification’ are required to identify the actors involved in the business transaction. The message attribute ‘message type identification’ is required to identify the function of the message in the business transaction (e.g. ‘order’, ‘planning’ or ‘report’). The message attribute ‘transaction identification’ is required to identify the business transaction to which the message belongs. A transaction identification should be unique in the context of the relation between service client and service provider. All messages that belong to one business transaction must have the same transaction identification.

We have seen that each message belongs to one message type and that one or more message types can be used in a business transaction. In most cases the order in which message types are exchanged is not entirely free, but must follow a pattern called the transaction protocol.

Definition: *transaction protocol, scenario*

A *transaction protocol* defines the message types that can be exchanged in a business transaction and the conditions to the order in which this can be done. One specific sequence of message types in a transaction is called a *scenario*. A transaction protocol allows one or more scenario’s.

Although the transaction protocol and the underlying message types must be agreed upon by the business partners involved in a business transaction, exceptions can occur if the specification is not implemented correctly. In general, there are two types of errors:

Definition: *data error, protocol error*

A *data error* occurs when a message is exchanged that does not conform to the definition of its message type. A *protocol error* occurs when a message is exchanged that is not valid in its business transaction at that moment according to the transaction protocol.

The relationships between the concepts that are introduced in this section are shown in Figure 30. We have added the concepts ‘service type’ and ‘service’ to show the relationship between ‘transaction protocol’ and ‘service type’, and between ‘transaction’ and ‘service’.

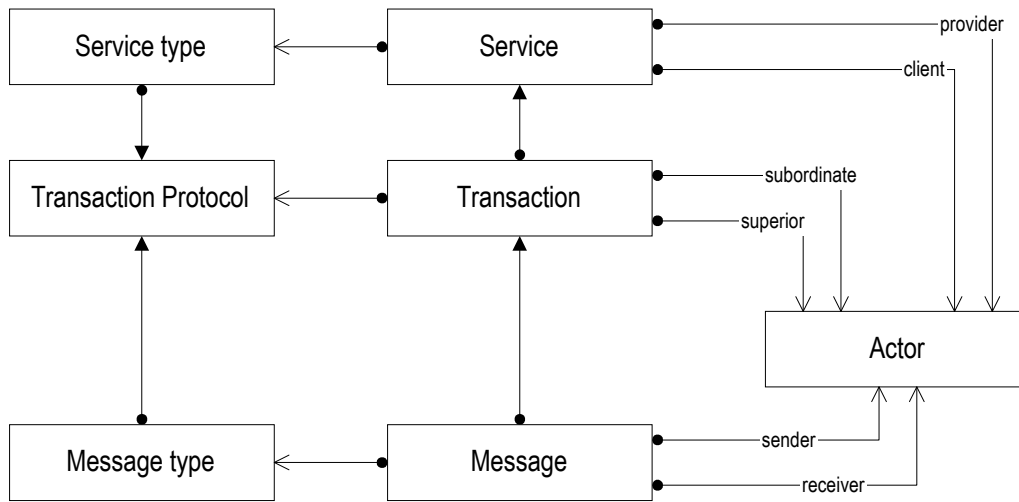


Figure 30 Relationships between business transaction concepts

Messages are modelled as tokens of which the colour is defined as a complex. The complex class of a message token is defined by the object model in Figure 31. The complex class contains an entity ‘MESSAGE’ with attributes ‘*identification*’, ‘*type identification*’, ‘*transaction identification*’, ‘*sender identification*’ and ‘*receiver identification*’. The attribute ‘*identification*’ models the identification of the message, which should be unique in the transaction. The attribute ‘*type identification*’ models the unique identification of the message type to which the message belongs. The attribute ‘*transaction identification*’ models the identification of the transaction to which the message belongs. The rest of the complex class is analogous to the ‘*case*’ and ‘*service*’ complex classes defined in Figure 25 and Figure 28.

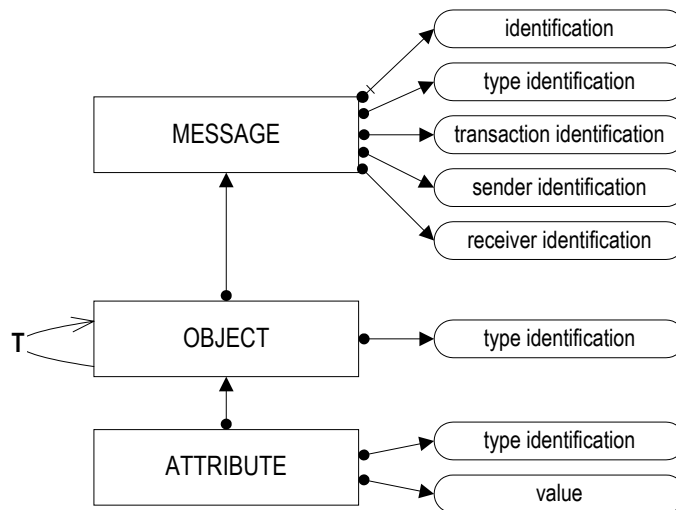


Figure 31 Object model of complex class ‘message’

We will now discuss the structure of the message data. Since a service is specified by its service data, both service client and service provider must at each time have the complete and up-to-date service data at their disposal. However, service attributes are created and updated by the service client as well as the service provider, and at different moments too. For instance, the service attribute ‘*requested date of delivery*’ is created by the service client at the start of the business transaction, whereas the service attribute ‘*planned time of delivery*’ is created by the service provider after the route planning has been performed. To avoid miscommunication, it is

important that service client and service provider always have the same service data at their disposal. This goal can only be obtained if service client and service provider notify the other party of each relevant change in the service data. Concluding, messages in a business transaction can be viewed as a means to synchronise the service data of service client and service provider. This is illustrated in Figure 32 where service client and service provider each have a place containing service tokens. The service token at the client side is created by the service contracting process, where after a message is sent to the provider to create the service token there too. Here after, service client and service provider notify the other party by a message if changes are made to the service data in order to make the same changes in the service token of the other party. This exchange of messages makes it possible for service client and service provider to use consistent service data at all times.

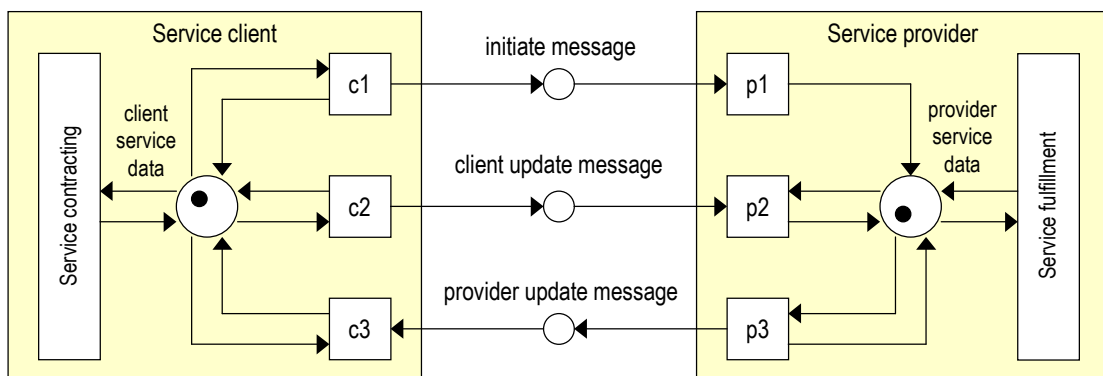


Figure 32 Message are exchanged to synchronise service data

If messages in a business transaction are used for synchronisation of service data, the structure of the message data can be defined as a subset to the structure of the service data. The structure of the message data in a message depends on its message type which is modelled as a complex of which the object model is given in Figure 33. The complex contains an entity 'MESSAGE TYPE' with attributes 'type identification', 'name' and 'description'. The attribute 'type identification' is a unique identification of the message type. The attributes 'name' and 'description' contain the name and description of the message type in clear text. The relationship between the entities 'MESSAGE TYPE' and 'SERVICE TYPE' models the service type for which a message type is used and the message types that can be used for a service type. A message type defines the structure of its message data as a subset to the service data. Therefore, the entities 'OBJECT TYPE' and 'ATTRIBUTE TYPE' are used to model the structure of the service data associated with the service type. Both entities have only one attribute 'type identification', which is used as references to the service type complex (see Figure 29). The entities 'OBJECT TYPE USAGE' and 'ATTRIBUTE TYPE USAGE' define the status in the message data model (required, optional, not used) of each object type and attribute type in the service data model. The attribute 'maximum repeat' of entity 'OBJECT TYPE USAGE' models the maximum number of instances allowed for an object type.

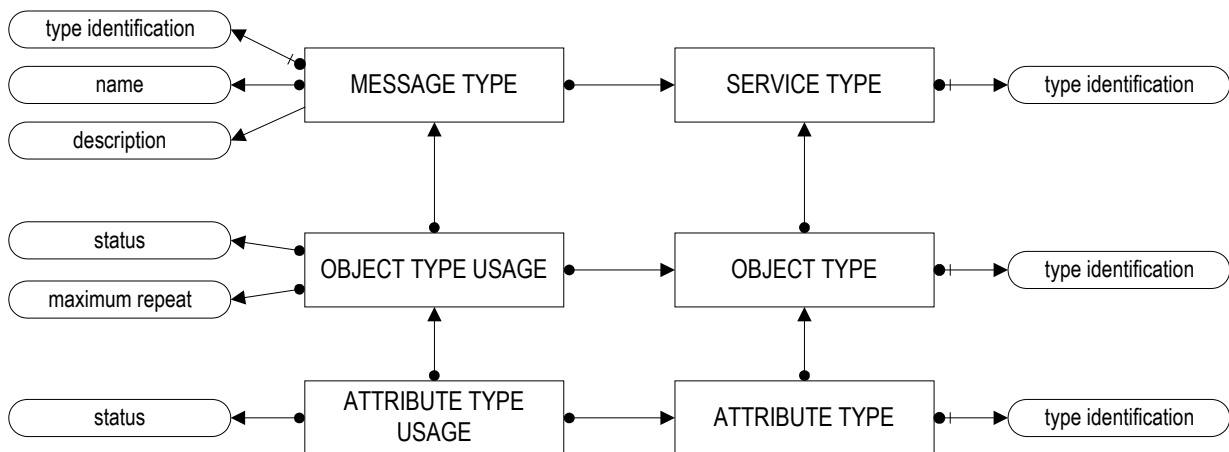


Figure 33 Object model of complex class 'message type'

A transaction protocol defines conditions to the order of message types in a transaction. A number of techniques can be used to define a transaction protocol. Frequently used techniques are: *message sequence charts*, *state transition diagrams* and *Petri nets*. All three techniques have a graphical representation and have their own advantages and disadvantages. The message sequence chart [73, 101] models two or more actors, between which a sequence of message types is exchanged. The actors are drawn as vertical bars, the message types are drawn as arrows between the vertical bars. The order of the messages in the diagram, read from top to bottom, represents the order of the messages in the transaction. An example of a message sequence chart is shown in Figure 34. Message sequence charts have the advantage of simplicity, which allows easy communication with people. However, the formalism has limited expressive power. It is therefore the appropriate technique to model a scenario.

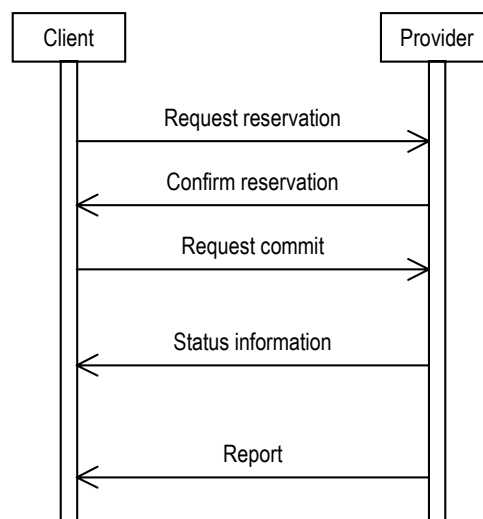


Figure 34 Example of a message sequence chart (UML notation)

The second technique, *state transition diagrams*, has more expressive power. A state transition diagram (or *statechart diagram* [101] in UML) consists of *states* and *transitions* between states. A transition corresponds to the exchange of a message between actors involved in the transaction. States are drawn as rectangles with rounded corners and transitions are drawn as arrows from one state to another. The major advantage of a state transition diagram is the capability to model all possible orders of message types in a transaction, if the transaction protocol does not contain parallelism. Combined with a graphical representation that is still easy to

understand, this makes the technique a good choice to model transaction protocols without parallelism.

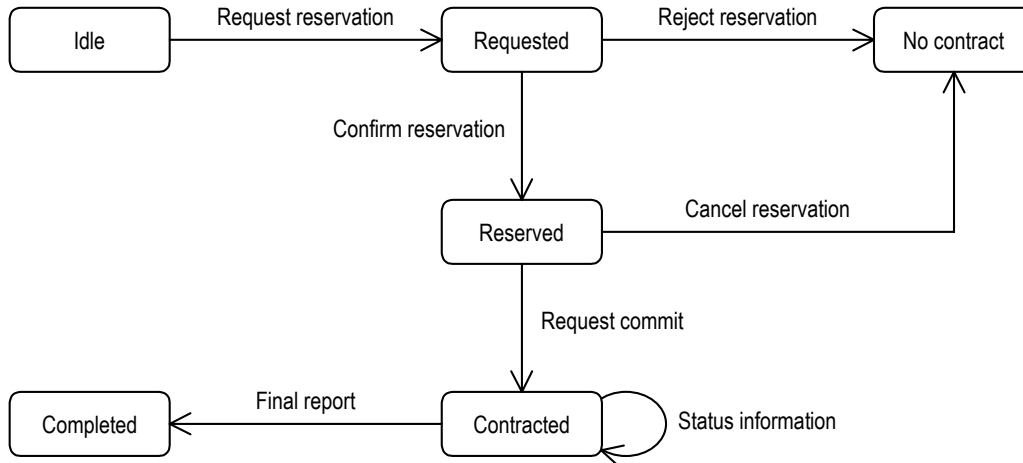


Figure 35 Example of a state transition diagram (UML notation)

The third technique to define a transaction protocol is coloured Petri nets. This technique can be used for transaction protocols that contain parallelism too. We will construct a Petri net for a transaction protocol as follows. A transaction protocol is modelled as a process between the superior and the subordinate of the transaction. Each actor has an input place for each message type he can receive and an output place for each message type he can send. A message is a token that is produced in an output place of the sending actor and has a colour defined by the object model in Figure 31. The transaction protocol consists of a place for each possible state and a processor for each allowed message type. The tokens in the places that model the transaction state have a colour defined by the object model in Figure 36.

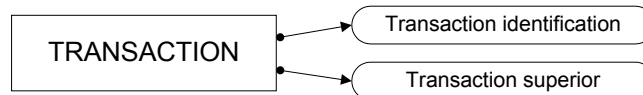


Figure 36 Object model of 'transaction' complex

The processor that models a message type has two input places: the state in which the message type can be sent and the place in which the sending actor produces the message token. There are also two output places: the state of the transaction after sending the message and the place from which the receiving actor consumes the message token. Because multiple transactions can be active simultaneously, each processor has a precondition. The precondition of a processor that models a message from service client to service provider is:

MESSAGE.transaction identification	=	TRANSACTION.transaction identification
MESSAGE.sender identification	=	TRANSACTION.transaction superior

The precondition of a processor that models a message from service provider to service client is:

MESSAGE.transaction identification	=	TRANSACTION.transaction identification
MESSAGE.receiver identification	=	TRANSACTION.transaction superior

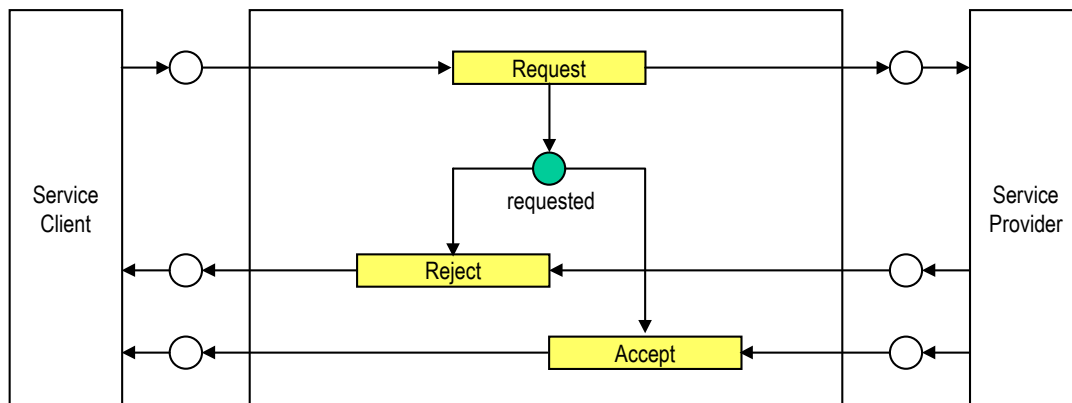


Figure 37 Example of a transaction protocol modelled as a Petri net

The example in Figure 37 shows a simple request / response protocol in which the response is either an accept or a reject. In this example, the response of the service provider can be sent at any moment after the request has been made. In reality, there will be a need to specify a maximum period of time between the request and the response. If the response is not sent within this time interval, a time out occurs and the transaction moves to a different state. This can be modelled by adding an extra '*time stamp*' attribute to the token colour which contains a time stamp and by assuming that transitions marked with a ⌚ symbol have access to a real time clock. When the '*request*' transition fires, it produces a token in place '*requested*' of which the time stamp attribute is equal to the current time. Transition '*time out*' is enabled when the current time is larger than the value of the '*time stamp*' attribute of the token in the '*requested*' place incremented with the maximum delay. If a time out occurs, the '*time out*' transition fires and the transaction protocol moves to the end state. If no time out occurs, the '*time out*' transition will not fire because the transaction token is already consumed from place '*requested*'.

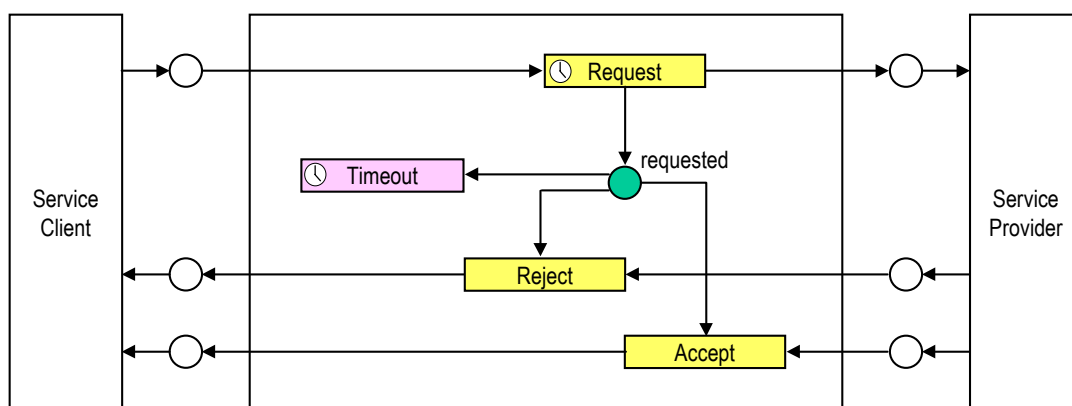


Figure 38 Example of time outs in a transaction protocol modelled as a Petri net

Finally, the information to create the transaction protocol Petri nets, used in this research, is defined by a complex of which the object model is given in Figure 39.

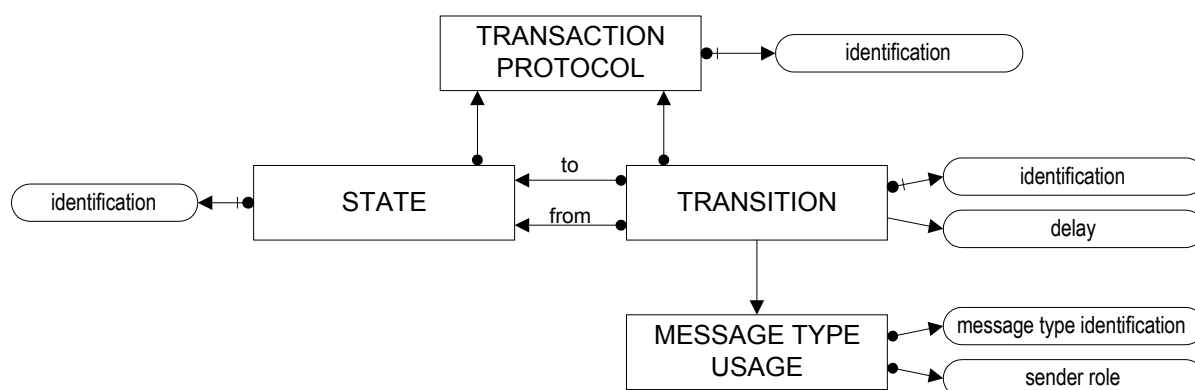


Figure 39 Object model of complex class ‘transaction protocol’

2.3 Basic service contracting concepts

This section introduces the concept ‘service contracting’ (2.3.1) and discusses frameworks for contracting processes found in literature (2.3.2). Section 2.3.3 describes the structure of the rest of this chapter in which the conceptual framework for service contracting is further developed.

2.3.1 Definitions

As we have seen in Chapter 1, the term ‘contracting’ is used for a variety of phenomena. This research focuses on the type of contracting that emerges when one organisation *outsources* the execution of one or more tasks in his business process to another organisation. In case of outsourcing, there must be an agreement between service client and service provider in which the mutual obligations are defined. We will use the term *service contract* for that purpose.



Definition: *service contract*

A *service contract* is a commitment from a service provider to execute a specific service for a service client, and the commitment of the service client to use the service executed by the service provider.

Clearly, when a service client outsources a task to a service provider, he has to perform activities to *establish* the service contract, to *monitor* the execution of the service according to the service contract and to *maintain* the service contract after it has been established (e.g. updates, aborts). We will use the term *service contracting* for this activity.



Definition: *service contracting*

Service contracting is the activity performed in the information system of a service client which is directed at establishing service contracts for business cases in the information system and monitoring and maintaining those service contracts thereafter.

The position of service contracting in an organisational system is illustrated in Figure 40. Service contracting is an activity that consists of information processing only and is therefore part of the information system of a service client. The information system of an organisation is responsible for allocating resources to tasks in the business process of that organisation. It is the specific responsibility of the service contracting process to find *external* resources for *out-*

sourced tasks in the business process. Service contracting processes communicate with the information system of service providers via messages. There can be message exchange between service client and service provider which is not related to service contracting too.

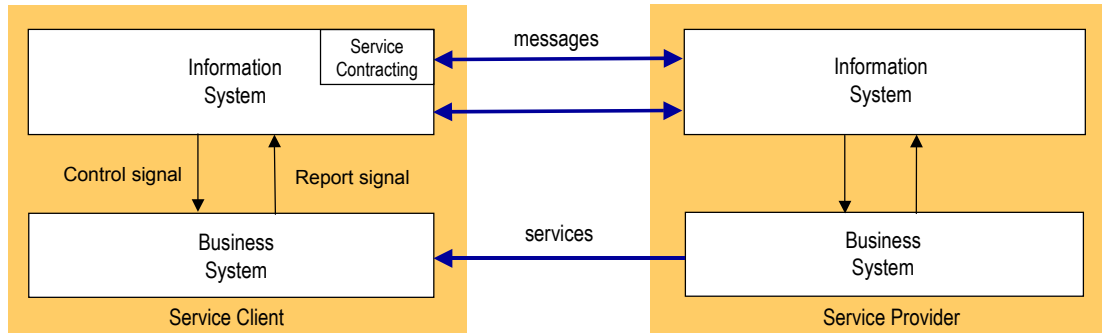


Figure 40 Position of service contracting in an organisational system

This research aims at automating service contracting processes for which we want to use workflow management and electronic commerce technology. For this reason, we will view the service contracting process as a workflow, expressed as a sound WF-net. In the remaining part of this research we will use the term *contracting workflow*, defined as follows.

Definition: *contracting workflow*

A contracting workflow is a representation of a service contracting process in the form of a sound WF-net.

It is important that a contracting workflow is sound for the obvious reasons of preventing deadlocks and livelocks. Furthermore, the property of soundness also guarantees that in the end-state one token is present in the sink place and all other places are empty. Because all internal places are empty, we can prove that after the workflow reaches the end-state, no actions like sending or receiving messages will be performed any more and no changes to the state data will be made.

In addition to establishing service contracts, service contracting also involves monitoring the execution of the service thereafter. All monitoring activities are based on information received from the service provider after the service contract is established (either spontaneously or on request). The result of this monitoring activities can be the detection of a service contract violation, defined as follows.

Definition: *service contract violation, violation type*

A service contract violation occurs when a service provider does not execute a contracted service according to the service contract. A violation type is a class of service contract violations with similar characteristics.

We will now introduce some properties of service contracting processes, which we will use to demarcate the class of service contracting processes to which this research focuses. First, we address the amount of information that is available to the service client. We distinguish between the situations of *full* knowledge and *partial* knowledge, which are defined as follows. We will speak of *full knowledge* when a service client has access to the entire resource planning of its service providers. We will speak of *partial knowledge* when a service client has no knowledge or partial knowledge of the availability of its service providers resources. This re-

search focuses on service contracting processes in situations of *partial* knowledge. This, and the fact that external service providers are often autonomous organisations, implies that a service client can not simply *assign* a task to a service provider but has to *negotiate* with the service provider instead. A contract is established only if there is an offer made by the service provider and an acceptance of the offer by the service client. Second, we will focus on the complexity of service contracting processes and the associated service contracting strategy. For this purpose, we make a distinction between *simple* service contracting processes and *compound* service contracting processes. A service contracting process is *simple* if a business case requires exactly one service of a specific type to be contracted. A service contracting process is *compound* when two or more services must be contracted and it can not be divided into two or more isolated simple service contracting processes. This research focuses on *compound* service contracting processes, primarily because in these cases the service contracting process is not trivial and the added value of a software component for service contracting is highest. Finally, we will make a distinction between *repeating* and *one-of-a-kind* service contracting processes. We will speak of a *repeating* service contracting process when a business case type has many instances and each instance requires the same service contracting process to be executed. We will speak of a *one-of-a-kind* service contracting process when each business case type has only one instance or each instance requires a different service contracting process to be executed. This research focuses on *repeating* service contracting processes, because automating these kind of processes can bring a large efficiency gain to an organisation.

2.3.2 Frameworks

In Chapter one we have mentioned different types of inter-organisational processes, e.g. those based on information sharing, capacity sharing, case transfer and contracting. This research focuses on inter-organisational processes that emerge because of contracting, which means buying products or services from a third party. We will now present examples of frameworks for buying products or services found in literature. The examples are: Action Workflow, DEMO and BAT.

- **Action Workflow**

Action Workflow can be seen as a generic framework for business between a customer and a performer, and is the name of a supporting software tool. The transactions that occur in a business process consist of four steps: preparation, negotiation, performance and acceptance. The preparation and negotiation steps aim at establishing a commitment to perform an action. The performance and acceptance steps aim at establishing an agreement that the action has been performed. In both parts there is negotiation aimed at mutual agreement of what has to be established.

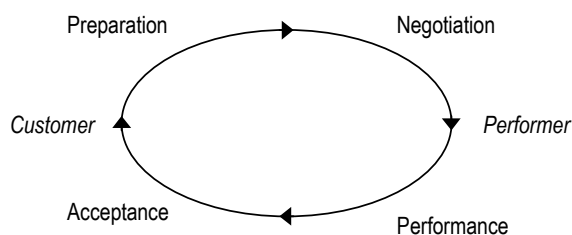


Figure 41 Action Workflow transaction concept

- **DEMO**

DEMO (Dynamic Essential Modelling of Organisations) is motivated by the need to have a theory about the dynamics of activities in organisations for the purpose of Information Systems analysis (Dietz [34]). The concept for business transactions that is part of DEMO is shown in Figure 42.

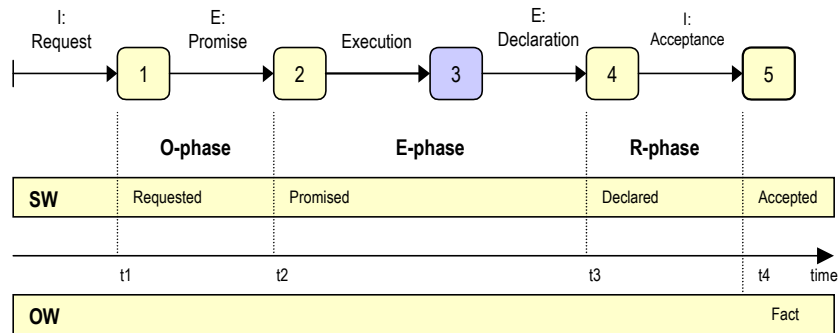


Figure 42 DEMO transaction concept

The upper part of the figure contains the so called ‘transaction diagram’. Each rounded square is a state in the transaction and the arrows are state transitions. A transaction consists of three phases: the *order phase* (O-phase), the *execution phase* (E-phase) and the *result phase* (R-phase). The figure shows the transactions in the System World (SW) and the Object World (OW) too. The order phase starts at time t_1 when the status ‘requested’ is reached and ends at time t_2 when the status ‘promised’ is reached. The result phase starts at time t_3 when the status ‘declared’ is reached and ends at t_4 in the status ‘accepted’. In-between the order phase and the result phase is the execution phase. The actor that performs the ‘request’ action is called the initiator or superior of the transaction. The actor that performs the ‘promise’ action is called the executor or subordinate of the transaction. All transactions have one actor as superior and one actor as subordinate.

- **Business as Action game Theory (BAT)**

The Business as Action game Theory of Goldkuhl [56] describes a generic framework for a business transaction between a supplier and a customer of products (goods or services). An illustration of the generic business framework is given in Figure 43. Goldkuhl views the business transaction as an interchange process between supplier and customer that involves the creation and sustainment of business relations. Four different phases can be recognised in a business transaction:

- proposal phase;
- commitment (contractual) phase;
- fulfilment phase;
- completion (acceptance/claim) phase.

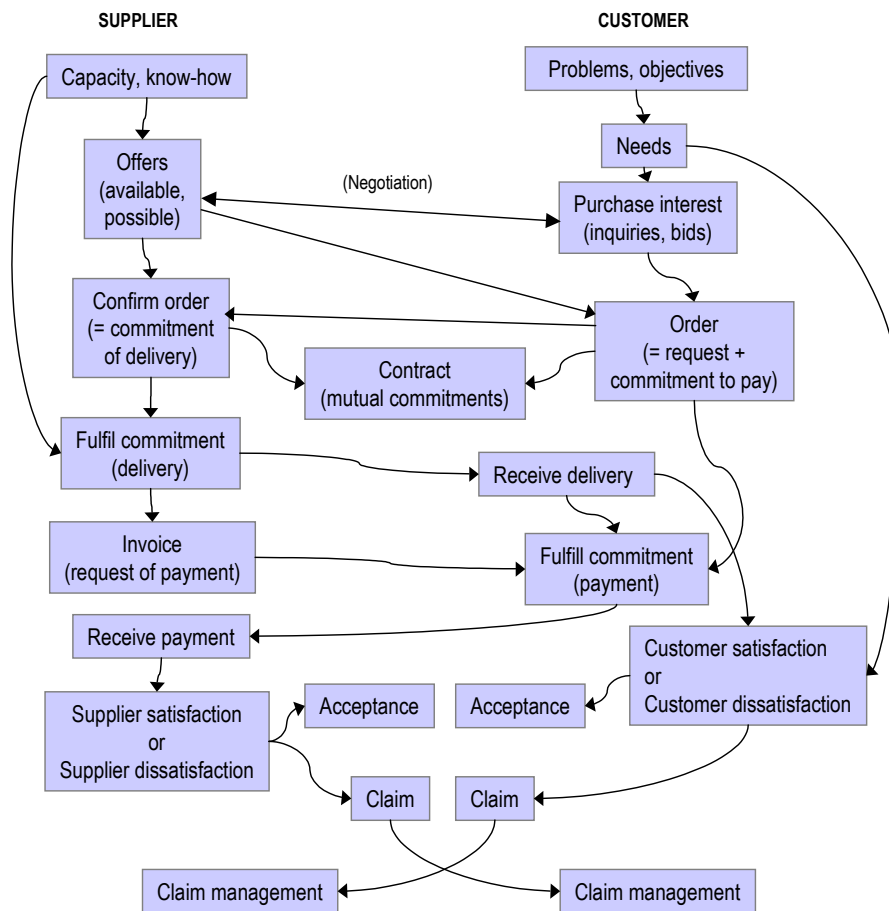


Figure 43 Business as Action Theory transaction concept

The three examples of frameworks for business transactions found in literature share the idea that business transactions consist of four phases:

- **Phase 1: Specification**

In the *specification* phase, the service client specifies the details of the service to be contracted. Since a service is modelled in the information system by its service data, the specification phase is about creating the initial value of the service data with which the negotiation phase starts. In fact, because each service requires a service provider to execute his business process, the specification phase is in its essence the creation of a case token for the workflow in the service providers information system.

- **Phase 2: Negotiation**

The *negotiation* phase aims at establishing a contract with a service provider for the specified service. This research focuses on service contracting processes in situations of *partial* knowledge. This, and the fact that external service providers are often autonomous organisations, implies that a service client can not simply *assign* a task to a service provider but has to *negotiate* with the service provider instead. A contract is established only if there is an offer made by the provider and an acceptance of the offer by the client. The negotiation phase ends either with a contract after which the execution phase starts, or without contract after which the process ends (failed).

- **Phase 3: Execution**

If a negotiation process resulted in a service contract, both service client and service provider will have to *fulfil* the commitments they entered in the contract. An important aspect of the execution phase is the exchange of *status information* from service provider to service client, used by the service client to *monitor* the fulfilment of the contract. This information informs the client about the planned execution and/or actual execution of the service. A typical example of status information is tracking and tracing information during a transport. Status information can be send by the service provider spontaneously or as a response to a request from the service client. The latter allows the service client to have a pro-active behaviour. The execution phase ends either with the completion of the execution after which the acceptance phase starts, or it ends with an abortion of the execution after which the process ends.

- **Phase 4: Acceptance**

The objective of the acceptance phase is to obtain a mutual agreement on the fulfilment of commitments. The service provider declares the fulfilment of his commitments, the service client accepts this declaration and settles the financial obligations towards the service provider. Settlement of financial obligations is however outside the scope of this research. During the acceptance phase, information must be exchanged between service client and service provider. At this point, mutual satisfaction is obtained and the transaction is completed.

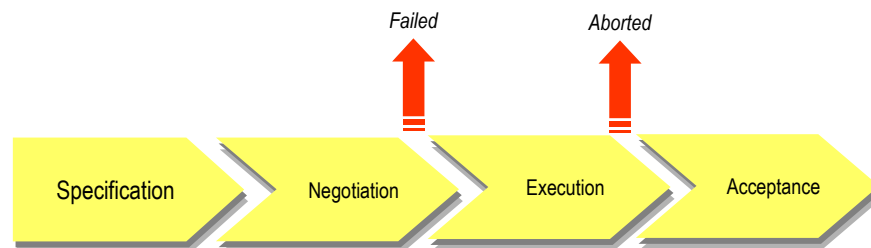


Figure 44 Flow of service contracting processes

2.3.3 Structure of the rest of this chapter

A detailed framework for service contracting processes is given in the rest of this chapter, of which the structure is illustrated in Figure 45. First, Section 2.4 addresses the specification of interchange agreements between service clients and service providers (service types, transaction protocols). There after, Section 2.5 addresses the specification of contracting requirements for outsourced tasks in the internal workflow. Contracting requirements define *which* services must be contracted for a specific business case and *how* these services must be contracted (e.g. negotiation strategy). Next, Section 2.6 describes the structure of the contracting workflow as a high level coloured Petri net. We will define standard transitions of which contracting workflows can be composed. Furthermore, we will give rules by which the contracting workflow can be generated from the interface agreements (2.4) and the contracting requirements (2.5).

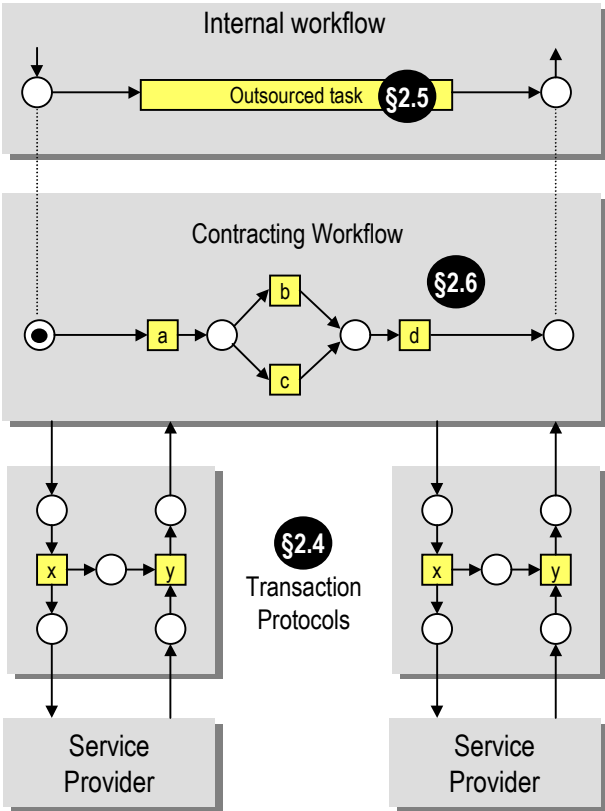
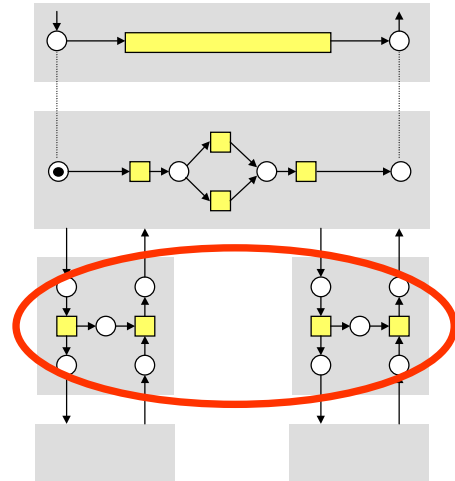


Figure 45 Elements of the conceptual framework

2.4 Specification of the interface agreements

Service providers and service clients are autonomous organisations, for which we assume no knowledge of each others business processes. However, service contracting involves co-operation between service client and service provider via the mechanism of business transactions. To make this co-operation possible, service providers and service clients must have a minimal set of agreements acting as interface specification. A meta model for this interface specification is given in this section.



2.4.1 Data model

The definition of the interchange agreements in terms of available services is a complex of which the object model is given in Figure 46. The basis is the ‘SERVICE TYPE’ entity that models a service type as defined in Section 2.2.1. The ‘*type identification*’ attribute is a unique identification of the service type. The ‘*name*’ attribute is a textual representation of the main characteristics of the service type, e.g. ‘container road transport’. The ‘*description*’ attribute models a detailed textual specification of the service type’s nature including conditions like payment terms, etc. Finally, the ‘*data model*’ attribute defines the service data structure in the form of a hierarchic data model consisting of entities, attributes, code lists and codes. In the object model in Figure 46 we modelled a hierarchic data model as a single attribute ‘*data model*’ to keep the object model simple. However, each time we use an attribute ‘*data model*’ in an object model, the attribute is in fact a complex of which the object model is given in Figure 47.

Service types are offered to the market by service providers, modelled by the ‘SERVICE PROVIDER’ entity. The ‘*identification*’ attribute contains a unique identification of the service provider, whereas the ‘*name*’ attribute contains a textual representation of the service providers identity. Finally, the attribute ‘*URL*’ models the web location where further details of service provider like addresses, financial accounts, communication addresses, etc. can be found.

The relationship between service types and service providers is modelled by the ‘AVAILABILITY’ entity. The ‘*constraints*’ attribute models constraints to the service data and are used to demarcate a class of services that can be contracted from a specific service provider. Constraints can for instance be used to limit the available services to a specific geographic area. For example, a transport company could impose the constraints “country of loading = ‘NL’ and country of delivery = ‘NL’” to offer transport services in the Netherlands to the global market.

The ‘VIOLATION TYPE’ entity models a possible violation type for a service type. The ‘*type identification*’ attribute is a unique identification of a violation type. The ‘*name*’ attribute models a textual representation of the nature of the violation type. Finally, the ‘*constraints*’ attrib-

ute models one or more constraints to the service data that are used to detect the occurrence of a violation. A violation occurs when these constraints are not fulfilled. For example, a constraint “latest date of delivery \geq actual date of delivery” can be used to detect a violation type with name ‘delivery date exceeded’.

The co-ordination between service client and service provider takes place via a business transaction according to a transaction protocol. Because defining a transaction protocol from scratch can be a complex task, we use patterns for transaction protocols. A transaction protocol pattern is a transaction protocol with a generic structure in which message types have generic names. A transaction protocol for a specific service type is defined by selecting a transaction protocol pattern in which the generic message names are replaced by specific message names and by defining a data model for each message type in the pattern as a subset of the service type data model. A transaction protocol pattern is modelled by a ‘PROTOCOL PATTERN’ entity. The ‘*identification*’ attribute models the unique identification of the pattern and the ‘*protocol definition*’ attribute models the constraints on the sequence in which the message types are used during the transaction. In order to keep the object model simple we used a single attribute ‘*protocol definition*’ which is in fact itself a complex of which the object model is given in Figure 39. A message type used in a transaction protocol pattern is modelled by a ‘PATTERN MESSAGE TYPE’ entity. The ‘*type identification*’ attribute models the unique identification of the message type. The ‘*generic name*’ attribute models a textual representation of the function and purpose of the message type, stated in generic terms since we are dealing with a pattern. Examples of transaction protocol patterns are given in Section 2.4.2.

The transaction protocol for a service type is modelled by a ‘TRANSACTION PROTOCOL’ entity. The entity has no attributes of its own and refers to exactly one ‘SERVICE TYPE’ entity, exactly three ‘PROTOCOL PATTERN’ entities and one or more ‘MESSAGE TYPE’ entities. The three relations to a ‘PROTOCOL PATTERN’ entity model the underlying negotiation, execution and acceptance protocol patterns. A ‘MESSAGE TYPE’ entity models a message type used in a transaction protocol for a specific service type. It refers to exactly one ‘PATTERN MESSAGE TYPE’ entity and has two attributes. The ‘*name*’ attribute models the specific name of the message type in the context of the service type, e.g. ‘transport instruction’ instead of the generic name ‘request contract’. The ‘*data model*’ attribute models the structure of the message data and is itself a complex of which the object model is given in Figure 47. The data model of a message type must be a subset of the data model of the corresponding service type. Furthermore, if a ‘PROTOCOL PATTERN’ entity has a relation to a ‘TRANSACTION PROTOCOL’ entity, we impose the restriction that each ‘PATTERN MESSAGE TYPE’ entity related to the ‘PROTOCOL PATTERN’ entity has a relation to exactly one ‘MESSAGE TYPE’ entity related to the ‘TRANSACTION PROTOCOL’ entity.

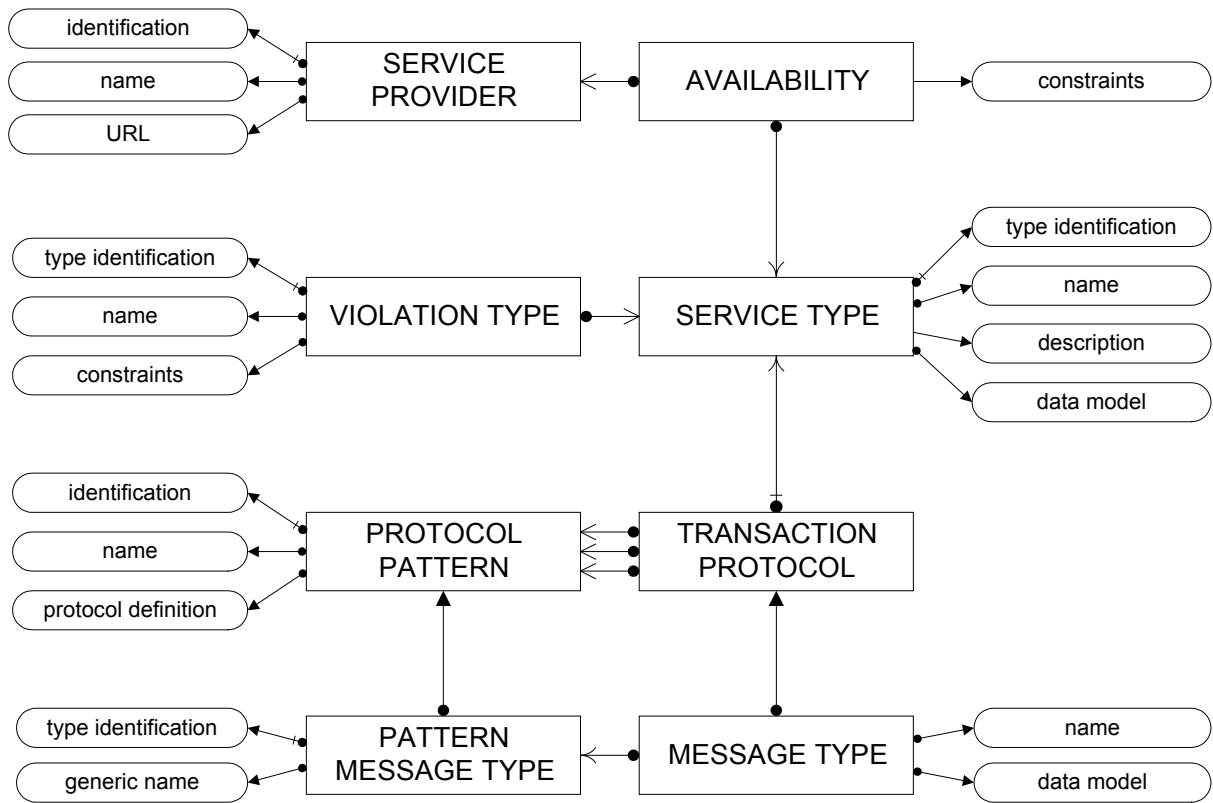


Figure 46 Object model of the complex class 'available services'

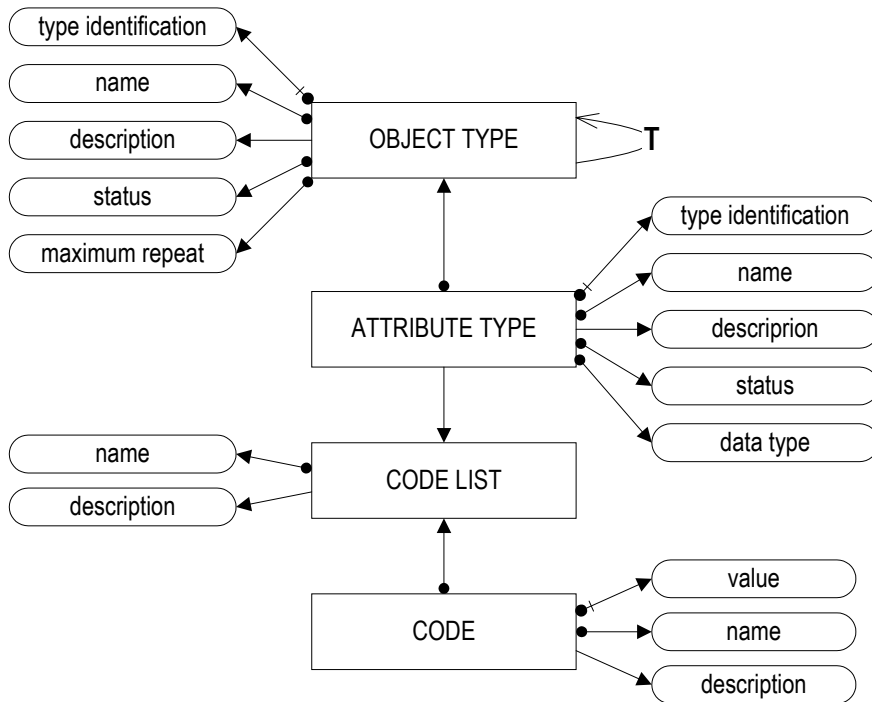


Figure 47 Object model of the complex class 'data model'

A grammar for the ‘constraints’ attribute

The ‘constraints’ attribute defines constraints on a hierarchic data set of which the structure is defined by a ‘data model’ attribute. A constraint is a Boolean expression in which attributes of the input data set are operands. We will now present a simple grammar for expressing constraints as an example. If necessary, this grammar can be replaced by another one with more expressive power. We will first define a grammar to reference a single attribute in a hierarchic data set of which the structure is defined by a ‘data model’ attribute. An attribute is identified by the attribute type name, prefixed by the entity type name. Because we allow an entity type to occur on multiple locations in the hierarchy, we will use a vector of entity type names instead of the name of a single entity type.

Letter	:=	<a-zA-Z>
Digit	:=	<0-9>
Char	:=	Letter Digit ‘_’ ‘ ’
Name	:=	Char ⁺
EntityType	:=	‘[‘ Name ‘]’
AttributeType	:=	‘(‘ Name ‘)’
DataRef	:=	EntityType ⁺ AttributeType

Figure 48 Grammar for references to data attributes

For example, the following expression refers to the attribute type ‘name’ in entity type ‘CUSTOMER’ nested below root entity ‘ORDER’ of the case type.

[ORDER] [CUSTOMER] (name)

We use this grammar for referencing attributes to build Boolean expressions of which the syntax is defined by the element BoolExpr in Figure 49.

Constant	:=	Digit ⁺ (‘.’ Digit ⁺) ‘” Char ⁺ “
Opr1	:=	(‘+’ ‘-’ ‘*’ ‘/’ ‘&’)
Opr2	:=	(‘=’ ‘<>’ ‘>’ ‘<’ ‘>=’ ‘<=’)
Opr3	:=	(‘AND’ ‘OR’) [‘NOT’]
Expr	:=	Constant DataRef ‘(‘ Expr Opr1 Expr ‘)’
Comp1	:=	‘(‘ Expr Opr2 Expr ‘)’
Comp2	:=	DataRef ‘IN {‘ Constant (‘,’ Constant)* ‘}’
BoolExpr	:=	Comp1 Comp2 ‘(‘ BoolExpr Opr3 BoolExpr ‘)’

Figure 49 Grammar for constraints

A few examples of the use of constraints is given below. First, an example of a constraint used for detecting a contract violation is given. The expression compares the planned date of delivery received from the service provider with the latest date of delivery created when the service was specified.

[TRANSPORT] (planned date of delivery) <= [TRANSPORT] (latest date of delivery)

An example of a constraint used in the service type ‘AVAILABILITY’ entity is given next. The expression imposes constraints on the length, width and height of a consignment and limits the places of loading and delivery to locations in the Netherlands, Belgium or Luxembourg.

([TRANSPORT] [GOODS] (length) <= 5.0)	AND
([TRANSPORT] [GOODS] (width) <= 2.8)	AND
([TRANSPORT] [GOODS] (height) <= 2.2)	AND
([TRANSPORT] [PLACE OF LOADING] (country) IN {‘NL’, ‘BE’, ‘LU’})	AND
([TRANSPORT] [PLACE OF DELIVERY] (country) IN {‘NL’, ‘BE’, ‘LU’})	

2.4.2 Transaction protocol patterns

An essential part of the available service specification is the transaction protocol used for a service type. Clearly, there is not a single transaction protocol common to all possible service types. Differences in transaction protocols are likely to occur due to differences in legislation, business model, fulfilment processes, etc. However, although we can not present a single transaction protocol for all services, we are able to define *patterns* for transaction protocols. These patterns can be used to derive the actual transaction protocol by replacing the generic names of message types with specific names. Since a transaction protocol encompasses the consecutive negotiation, execution and acceptance phases, it can be seen as composed of three smaller transaction protocols, one for each phase. In the rest of this section, we will present the patterns per phase, starting with patterns for the negotiation phase.

Definition: *transaction protocol pattern*

A transaction protocol pattern captures the underlying common structure of a set of transaction protocols with different message types but identical dynamic behaviour. It is used to derive specific transaction protocols by replacing the generic message type names by specific ones.

The examples of negotiation patterns presented in this section are not taken randomly, but form a coherent set of which the taxonomy is given in Figure 50.

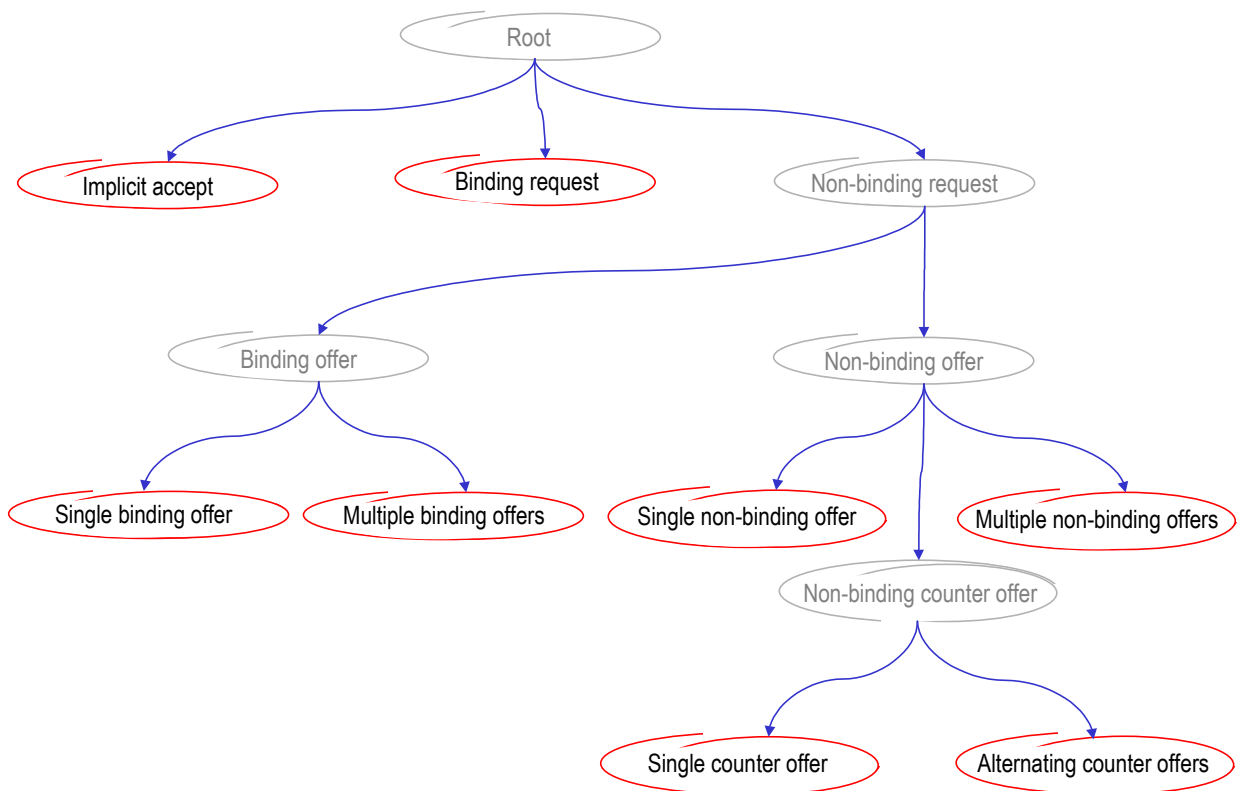


Figure 50 Taxonomy of negotiation patterns

Correctness criteria for transaction protocols

As we have seen before, we use high-level coloured Petri nets to model transaction protocols. It is important that transaction protocol specifications must adhere to correctness criteria to prevent anomalies in the message exchange between the parties involved in the contract. Essential criteria are for instance the absence of deadlocks and livelocks. Furthermore, when the transaction reached an end-state, it must not be possible to exchange any more messages in the transaction. These criteria are covered by the soundness property of WF-nets. However, in their original form the transaction protocols are not WF-nets. Therefore we define extensions to the transaction protocol net after which we insist that the resulting net is a sound WF-net. First, we impose the following constraints on negotiation, execution and acceptance protocols.

- A negotiation protocol contains no source place and exactly one sink place ‘*committed*’.
- An execution protocol contains one source place ‘*committed*’ and exactly one sink place ‘*executed*’.
- An acceptance protocol contains exactly one source place ‘*executed*’ and no sink place.

A transaction protocol WF-net is created from one negotiation protocol, one execution protocol and one acceptance protocol as follows.

- Delete all places that model messages exchanged between service client and service provider.
- Add an extra source place ‘*start*’ and connect it to each transition in the negotiation protocol that has no inbound connector.
- Combine the sink place ‘*committed*’ in the negotiation protocol and the source place ‘*committed*’ in the execution protocol into one place ‘*committed*’.
- Combine the sink place ‘*executed*’ in the execution protocol and the source place ‘*executed*’ in the acceptance protocol into one place ‘*executed*’.
- Add an extra sink place ‘*end*’ and connect it to each transition in the protocol that has no outbound connector.

The resulting Petri net must be a sound WF-net.

Negotiation patterns

- **Negotiation pattern: ‘implicit accept’**

This negotiation pattern is used in situations where there is no explicit response by the service provider to a request made by the service client. Instead, the contract is considered to be established after the request has been made. Clearly, this variant can only be applied under circumstances where the implicit accept is agreed in previous agreements or laws.

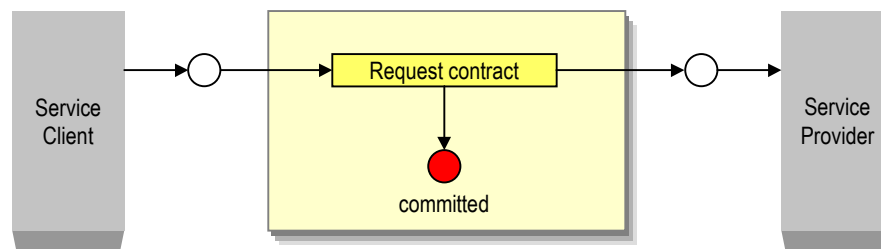


Figure 51 The ‘implicit accept’ protocol pattern for the negotiation phase

- **Negotiation pattern: ‘binding request’**

This negotiation pattern is used in a situation where a service client makes a binding request to a service provider, who responds by either accepting or rejecting the request. If the service provider accepts the request, a service contract is established, after which the execution protocol starts. If the service provider rejects the request, neither of the parties has a commitment to each other and the transaction ends.

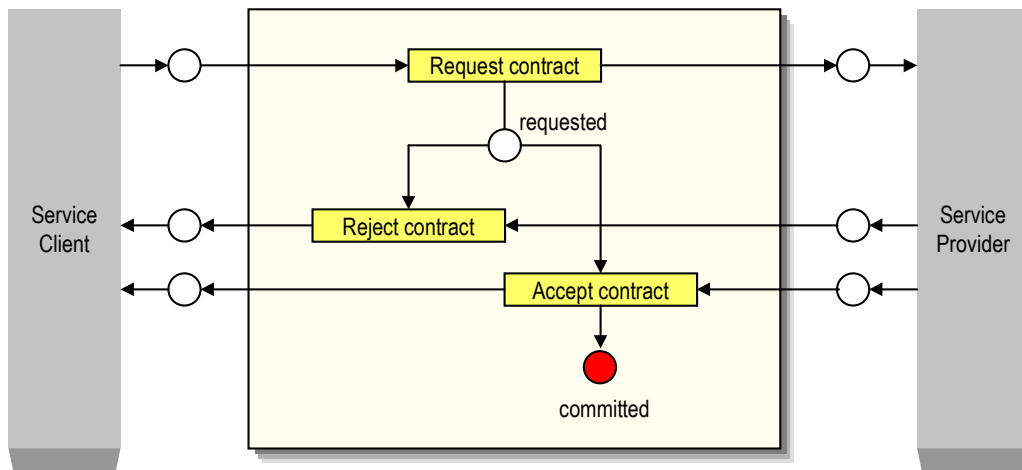


Figure 52 The 'binding request' protocol pattern for the negotiation phase

- **Negotiation pattern: 'single binding offer'**

Instead of requesting a contract from a service provider directly, a service client can also request an *offer* from a service provider. Offers can be binding or non-binding. This negotiation pattern is based on a single binding offer given by the service provider to the service client. When the service provider receives a request for an offer, he either responds by sending a notification that he will not make an offer (e.g. because he is not able to fulfil the request) or he responds by sending an offer message. When the service client receives an offer, he will either accept the offer after which a service contract is established and the execution phase starts, or he rejects the offer after which neither of the parties has a commitment to each other and the transaction ends.

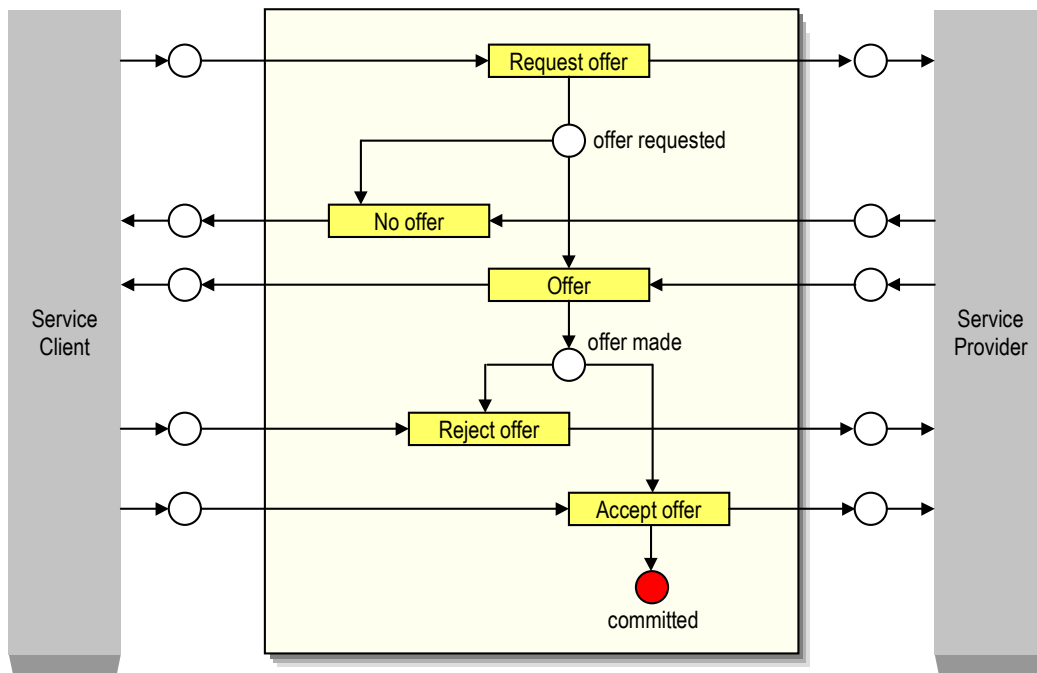


Figure 53 The 'single binding offer' protocol pattern for the negotiation phase

- **Negotiation pattern: ‘single non-binding offer’**

An extension to the ‘single binding offer’ pattern emerges when the service provider sends a *non-binding* offer instead of a binding offer. This leaves the possibility that after the service client accepted the offer the contract can still not be established, e.g. because the resources required for the fulfilment have been exhausted in the period between sending the offer and accepting it. The pattern is equal to the ‘single binding offer’ pattern, but has two additional message types that can be received by the service client after he accepted the offer. The confirm accept message indicates that a service contract has been established and the execution phase started. The reject accept message indicates that no contract could be established after all which ends the transaction and leaves both parties without any obligation towards each other.

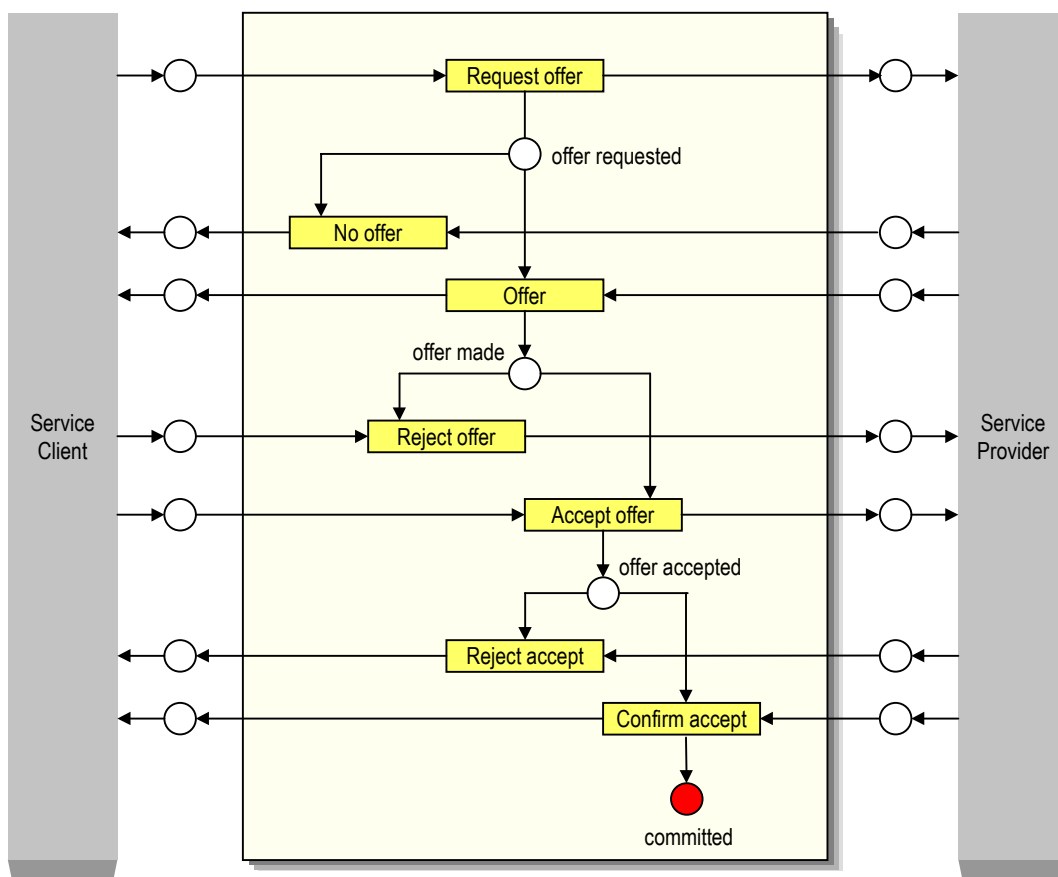


Figure 54 The ‘single non-binding offer’ protocol pattern for the negotiation phase

- **Negotiation pattern: ‘multiple non-binding offers’**

An extension to the ‘multiple binding offers’ pattern emerges when the service provider sends *non-binding* offers instead of binding offers. This leaves the possibility that after the service client accepted an offer the contract can still not be established, e.g. because the resources required for the fulfilment have been exhausted in the period between sending the offer and accepting it. The pattern is equal to the ‘multiple binding offers’ pattern, but has two additional message types that can be received by the service client after he accepted the offer. The confirm accept message indicates that a service contract has been established and the execution phase started. The reject accept message indicates that no contract could be established. However, the transaction returns to state ‘offer made’, in which the service client can cancel the negotiation or accept another offer from the pool of offers received from the service provider. In the same state, the service provider can send new offers to the service client.

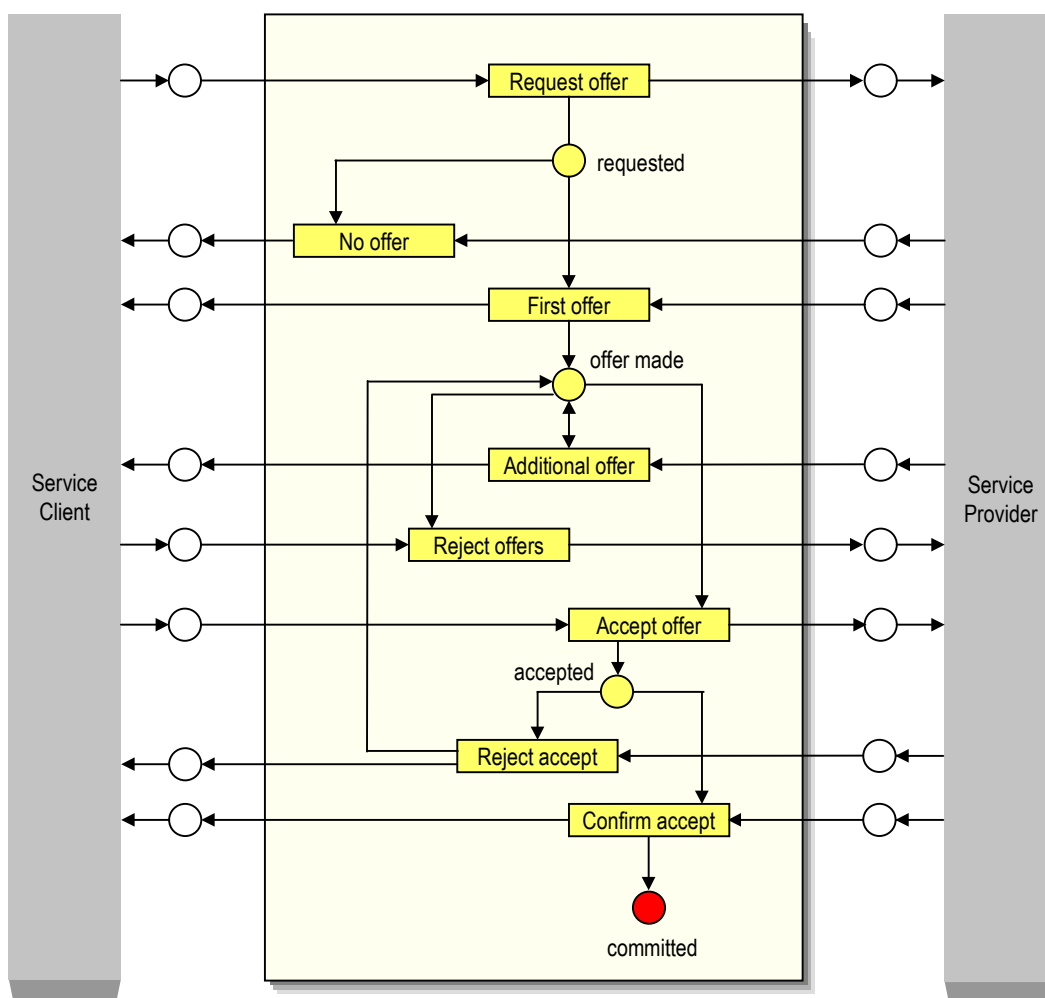


Figure 56 The ‘multiple non-binding offers’ protocol pattern for the negotiation phase

- **Negotiation pattern: ‘single binding counter offer’**

An extension to the ‘binding request’ pattern is to allow the possibility of a binding counter offer by the service provider as a third type of response to a direct request for a contract. When the service provider makes a counter offer, the negotiation process enters a state in which the service client can either accept or reject the counter offer and in which the service provider can withdraw the counter offer. If the service client accepts the counter offer, a service contract is established and the execution phase starts. Otherwise, the transaction ends leaving both parties without any obligations towards each other.

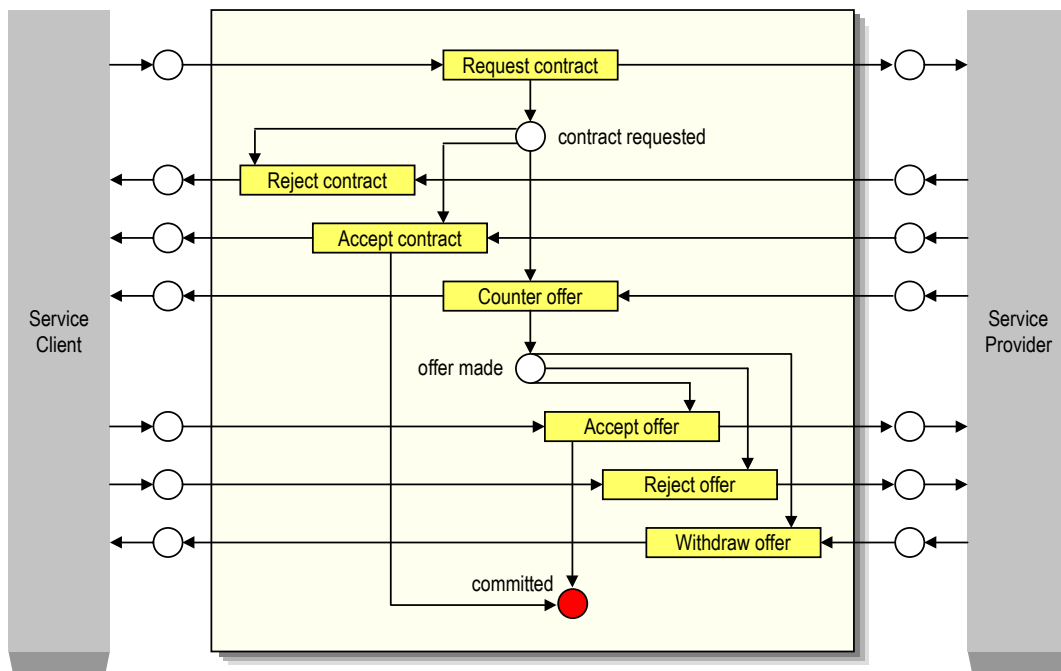


Figure 57 The ‘single binding counter offer’ protocol pattern for the negotiation phase

- **Negotiation pattern: ‘alternating binding counter offers’**

An extension to the ‘single binding counter offer’ pattern is to allow the possibility of a counter offer to be followed by a different counter offer made by either the service client or service provider. If a counter offer is made, it replaces all earlier made counter offers. Hence, a maximum of one counter offer can be under consideration at each moment. The party that made the current counter offer can replace it by a different counter offer or withdraw it. The party that did not make the counter offer under consideration can either accept it, reject it, or make a counter offer himself. If a party accepts an offer made by the other party, a service contract is established and the execution phase starts. Otherwise, the transaction ends leaving both parties without any obligations towards each other.

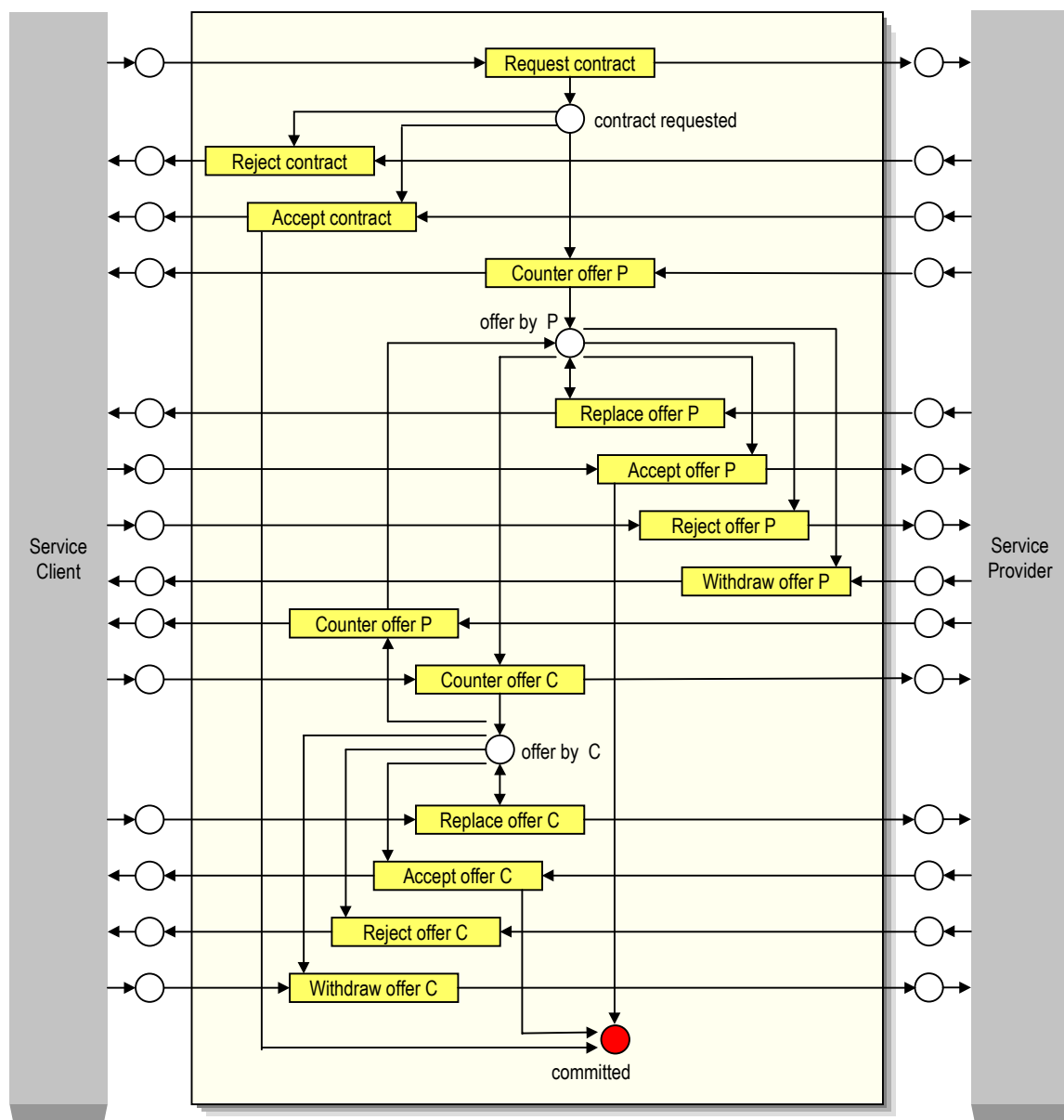


Figure 58 The ‘alternating binding counter offers’ protocol pattern for the negotiation phase

Execution phase patterns

For the execution phase, the following patterns are given as an example:

- **Execution pattern: ‘silent execution’**

This execution pattern is used in a transaction protocol when no messages are exchanged during the execution phase. The state of the transaction protocol moves directly from ‘committed’ to ‘executed’.

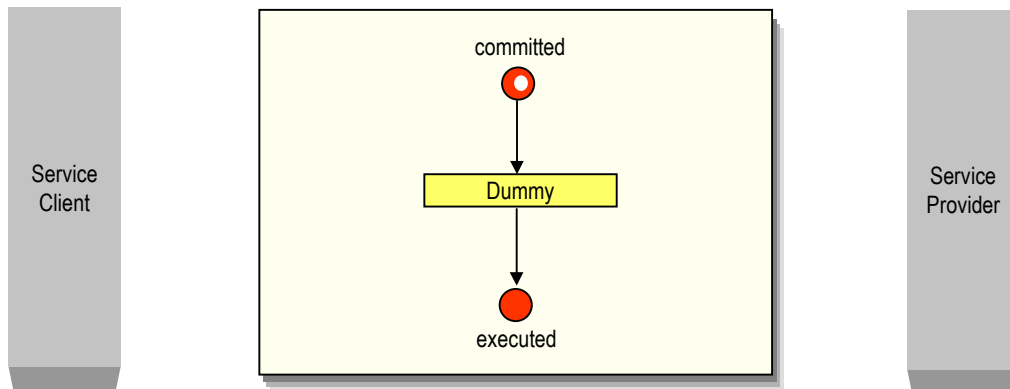


Figure 59 The ‘silent execution’ protocol pattern for the execution phase

- **Execution pattern: ‘single phase execution’**

In this execution pattern, there is a state ‘committed’ in which the service provider can send an arbitrary number of ‘intermediate status’ messages. The provider will eventually inform the service client on the completion of the execution by sending a ‘final report’ message or he will notify the client that the service could not be executed after all by sending an ‘abort notification’ message after which the transaction ends.

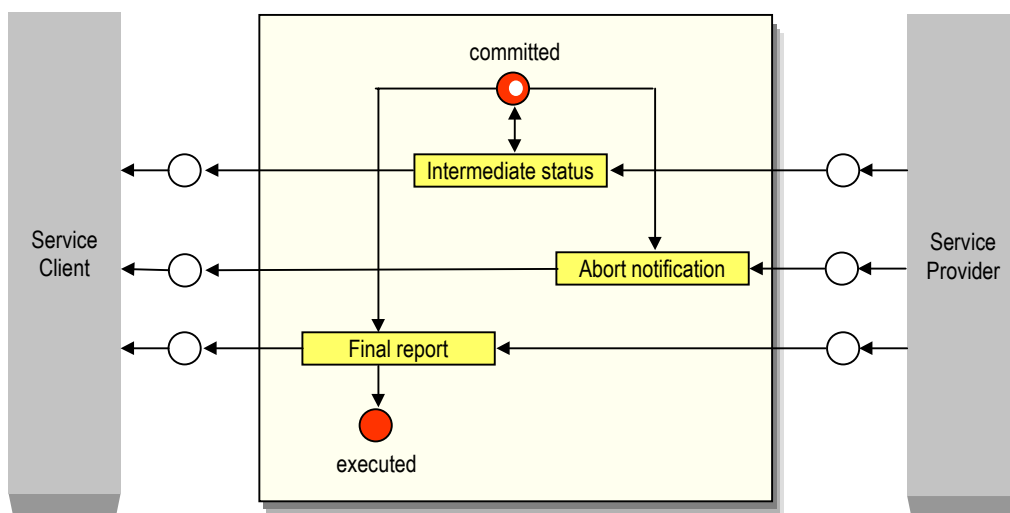


Figure 60 The ‘single phase execution’ protocol pattern for the execution phase

- **Execution pattern: ‘two phase execution’**

In this execution pattern, we distinguish a phase in which execution of the service is prepared followed by a phase in which the actual execution takes place. During the preparation phase, the service provider can send an arbitrary number of planning information messages. The preparation phase ends when the service provider informs the service client of the actual start of the execution. Here after, the service provider is no longer able to abort the service execution. During the execution, the service provider can send an arbitrary number of intermediate status reports. Eventually, the service provider will inform the service client on the completion of the execution (either successful or not successful) by sending a ‘final report’ message that describes the end state of the execution phase after which the acceptance phase starts.

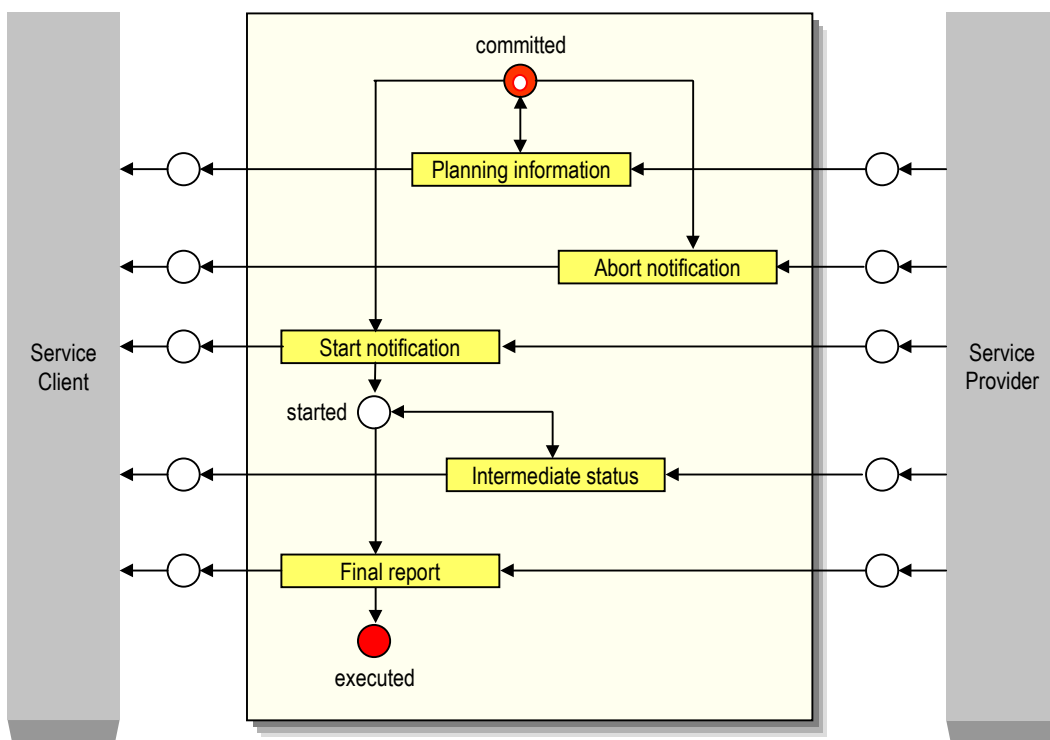


Figure 61 The ‘two phase execution’ protocol pattern for the execution phase

- **Execution pattern: ‘prepare to start / start’**

This execution pattern is used when the actual execution of the service does not follow the completion of the negotiation immediately, but is postponed to a later moment when the service client explicitly triggers the service provider to start the execution.

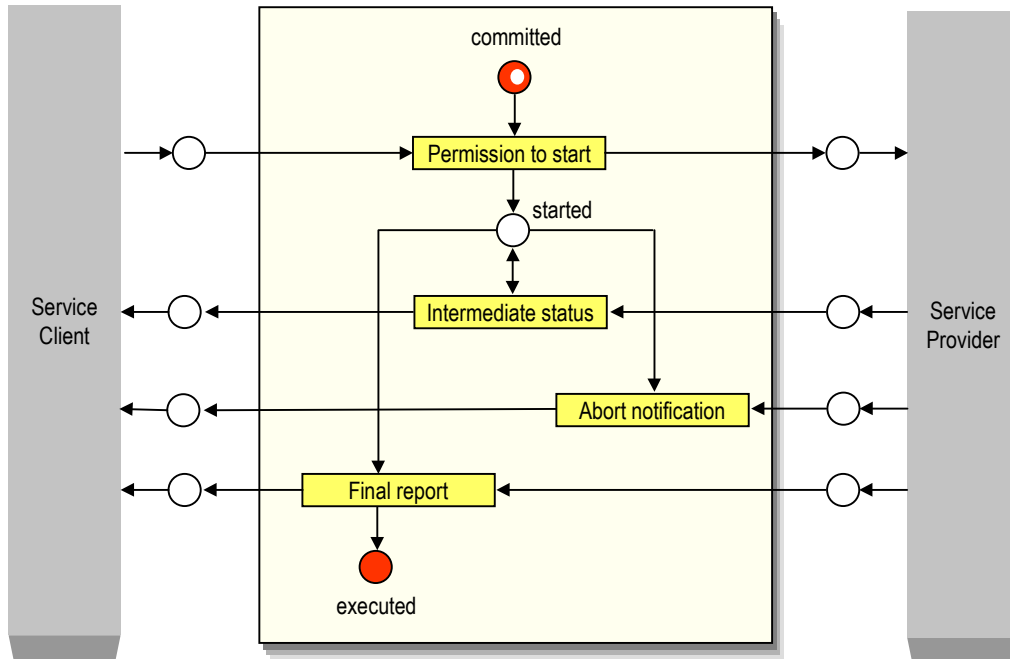


Figure 62 The ‘prepare to start / start’ protocol pattern for the execution phase

Acceptance phase patterns

For the acceptance phase, we define the following two patterns as an example:

- **Acceptance pattern: ‘silent accept’**

It is possible that no message exchange occurs during the acceptance phase, because acceptance of the result is either implicit or handled via other channels.

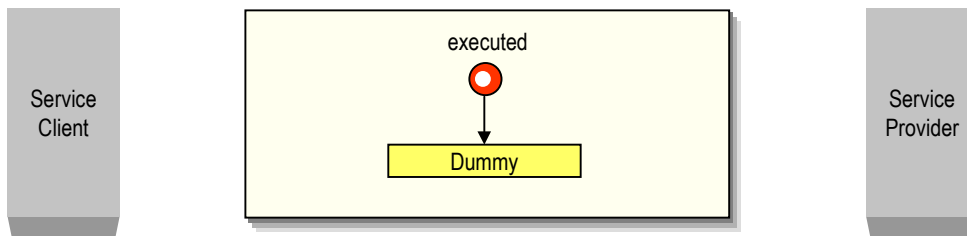


Figure 63 The ‘silent accept’ protocol pattern for the acceptance phase

- **Acceptance pattern: ‘accept / reject result’**

The acceptance phase starts when the service provider informs the service client of the completion of the execution and of the end state of the execution. The service client responds either by sending an ‘accept result’ message, hereby releasing the service provider of his contractual obligations, or he sends a ‘reject result’ message indicating he does not accept or partially accepts the result of the work executed by the service provider.

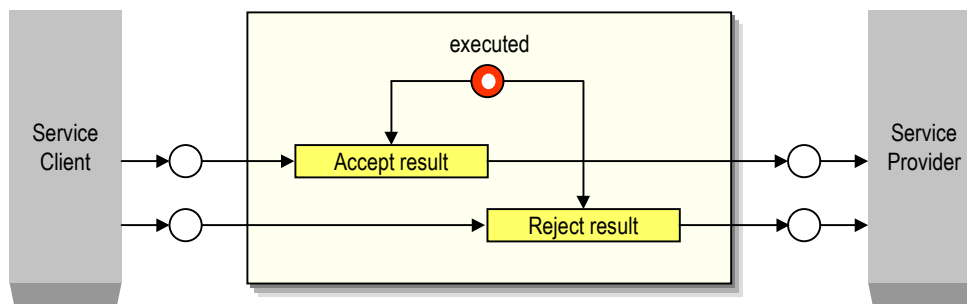
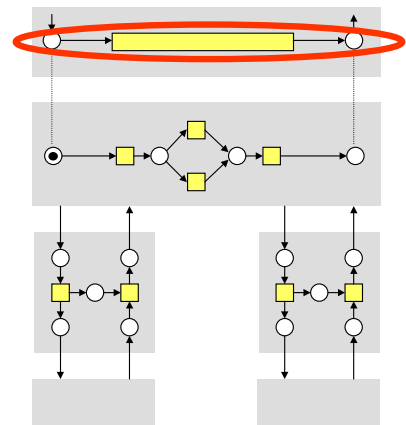


Figure 64 The ‘accept / reject result’ protocol pattern for the acceptance phase

2.5 Specification of contracting requirements

A service contracting process is executed for a business case in the business process of an organisation when this business process contains an outsourced task. Each case belongs to a case type and each case requires a specific set of services to be contracted. This section addresses the specification of contracting requirements that define which services must be contracted for a business case and how they must be contracted.



2.5.1 Definitions

We will use the term ‘contracting requirements’ for the information that defines *which* services must be contracted for an outsourced task in the business process of an organisation and *how* these services must be contracted. An important aspect of the contracting requirements is the possibility that a service contracting process involves more services than must actually be contracted for a specific business case. This can for instance be due to the fact that a specific service is only required for some business cases, depending on the value of the case data. Furthermore, it is possible that a service is required only if another (preferred) service could not be contracted (alternative service). These two examples illustrate that we can not simply refer to all services involved in a service contracting process as *required* services. Instead, we will use the term *candidate* services which is defined as follows.

Definition: *candidate service, candidate service type*

A *candidate service* is a service of a specific service type and with specific service data that is involved in the service contracting process for a specific business case. A *candidate service type* is a class of candidate services with similar characteristics defined for a case type.

Each candidate service type corresponds with one service type and therefore with one transaction protocol too. Although a transaction protocol defines *constraints* on the sequence of message types used in one business transaction, it still leaves a degree of freedom for the service client. The first example of this freedom is when the transaction protocol allows one of two different message types to be sent in a specific state, e.g. an ‘accept offer’ message or a ‘reject offer’ message. Furthermore, the service client has a free choice in creating the *contents* of outgoing messages, e.g. the contents of a ‘counter offer’ message as a response to a counter offer made by the service provider. Finally, when multiple service providers are available for one service type, a service client can have multiple concurrent business transactions for one required service. Although the transaction protocol defines constraints on the use of message types in one transaction, it does not impose constraints on the *interleaving* of concurrent transactions. Therefore, although a transaction protocol imposes constraints on the behaviour of a service client in a business transaction, the behaviour of the service client is not entirely determined by the transaction protocol, but involves choices to be made by him. Clearly, when we want to automate the service contracting process, we need a *decision rule* for every choice the service client is encountered with during a business transaction. We therefore introduce the notion of a *contracting strategy*, which defines the interleaving of concurrent business transactions, the rules according to which a service client chooses between alternative branches in a transaction protocol and the rules according to which the contents of outgoing messages is created. For each transaction protocol pattern we can define one or more contracting strategies.

Definition: *contracting strategy*

A *contracting strategy* defines the behaviour of a service client towards a service provider in a business transaction, within the constraints of the agreed transaction protocol.

We will use the term ‘contracting strategy’ (or just ‘strategy’) as a generic term for the decision rules that govern the entire business transaction (negotiation, execution and acceptance). If we refer to a specific part of a contracting strategy that applies to one contracting phase only, we will use the terms ‘negotiation strategy’, ‘execution strategy’ and ‘acceptance strategy’. An important part of a negotiation strategy is dealing with multiple service providers. At a high level of abstraction, there are two types of negotiation strategies.

- **Sequential exploration**

In this approach, the service client always negotiates with one service provider at a time. Before the negotiation starts, the service client ranks the available service providers according to his preference. He then starts a negotiation with the first service provider on the list. If this negotiation results in a contract, the process ends. Otherwise, a negotiation is started with the next service provider on the list. The process ends when a contract is established or when the list of available service providers is exhausted.

- **Parallel exploration**

In this approach, the service client negotiates with two or more service providers simultaneously. Clearly, since the negotiation task must end in exactly one service contract, this can

only be done in situations where a negotiation protocol is not based on a binding request by the service client.

Contracting strategies are described in plain English. An example of a negotiation strategy for the ‘multiple binding offer’ negotiation pattern is:

“Send a ‘request offer’ message to all available providers. Wait 60 seconds and rank the received ‘offer’ messages according to increasing value of the message attribute ‘Total-NetPrice’. If the value of the offer on top of the list is lower than the case attribute ‘MaximumPrice’, accept the offer and reject all other offers. Otherwise, reject all received offers.”

Clearly, this negotiation strategy is used for a very specific business situation. However, it is possible to distinguish a generic structure behind different strategies, which we will call the strategy type.

Definition: *strategy type, parameter type*
 A *strategy type* is a class of strategies with similar characteristics. It has the form of a strategy and uses one or more *parameter types* as placeholders for actual values. A strategy is derived from a strategy type by assigning a parameter value to each parameter type.

An example of a negotiation strategy type, which is the basis for the negotiation strategy example shown above is the following. The example uses three parameter types: P1, P2 and P3.

“Send a ‘request offer’ message to all available providers. Wait <P1> seconds and rank the received ‘offer’ messages according to increasing value of message attribute <P2>. If the value of the offer on top of the list is lower than the case attribute <P3>, accept the offer and reject all other offers. Otherwise, reject all received offers.”

2.5.2 A data model for contracting requirements

The relationship between the concepts ‘protocol pattern’, ‘strategy type’ and ‘parameter type’ is defined in Figure 65.

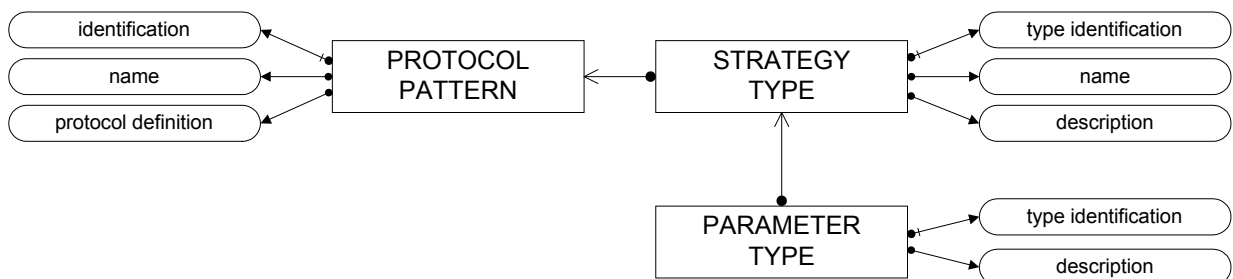


Figure 65 relationship between protocol patterns, strategy types and parameter types

Having defined the concepts of strategy and strategy type, we can define the contracting requirements in a complex of which the object model is given in Figure 66.

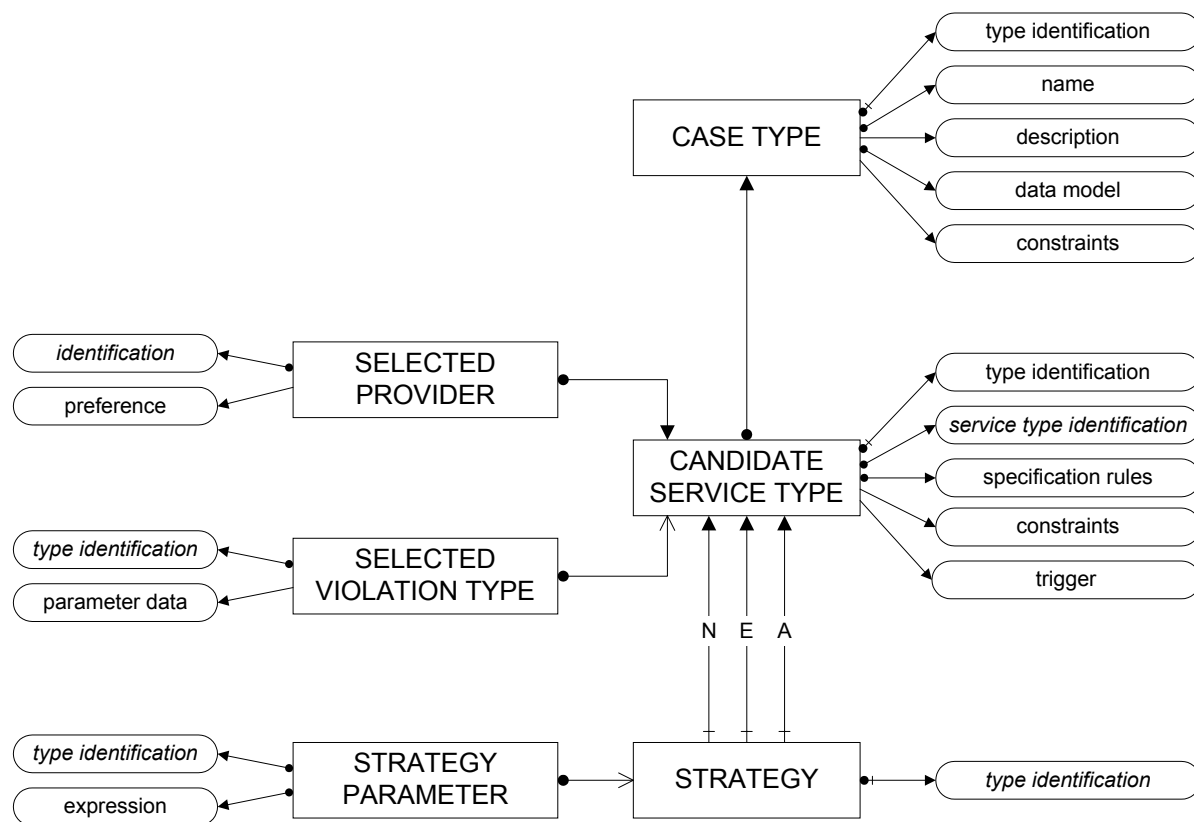


Figure 66 Object model of the complex class 'contracting requirements'

The basis for the contracting requirements parameter is a 'CASE TYPE' entity, for which we already defined an object model in Figure 26. As has been explained before, in order to keep the object model in Figure 66 simple, we modelled the hierarchic data model of the case data by a single attribute 'data model'. Furthermore, we added a 'constraints' attribute to model constraints on the case data that must be fulfilled by all business cases that are handled by a service contracting process. A 'CANDIDATE SERVICE TYPE' entity models a candidate service type involved in the service contracting process belonging to a case type. The 'type identification' attribute models the unique identification of the candidate service type. The 'service type identification' attribute models the service type on which the candidate service type is based and is a reference to a 'SERVICE TYPE' entity in the available services definition (Figure 46). The 'specification rules' attribute defines a data transformation operation of which the result is the service data with which the negotiation process starts. The 'constraints' attribute can be defined for a candidate service type that must be contracted only when one or more constraints are fulfilled. If there is a constraint on the order in which different candidate services must be contracted, the 'trigger' attribute is used to model the events that together form a precondition on the contracting process for candidate services belonging to the candidate service type.

The 'SELECTED PROVIDER' entity models a service provider to be used in the contracting process for candidate services belonging to the candidate service type. The 'identification' attribute is a reference to a 'SERVICE PROVIDER' entity in the available services definition (Figure 46). The 'preference' attribute is used to rank the selected service providers according to preference and is an integer from the range 1, 2, 3, ...

The 'SELECTED VIOLATION TYPE' entity models a violation type defined for the corresponding service type as relevant for the candidate service type. The 'type identification' attribute is a

reference to a ‘VIOLATION TYPE’ entity in the available services definition (Figure 46). The ‘*parameter data*’ attribute contains the values for parameters used in the constraints by which the violation type can be detected.

The ‘STRATEGY’ entity models a negotiation, execution or acceptance strategy for the candidate service. The ‘*type identification*’ attribute is a reference to a ‘STRATEGY TYPE’ entity (see Figure 65). The ‘STRATEGY PARAMETER’ entity models the value for a parameter type in the strategy type. The ‘*type identification*’ attribute is a reference to a ‘PARAMETER TYPE’ entity and the ‘*expression*’ attribute models the actual parameter value.

A grammar for the ‘*specification rules*’ attribute

As we have seen in Section 2.3.2, the first phase of a contracting process involves the specification of the service data of the required service. The object model in Figure 66 contains an attribute ‘*specification rules*’ that defines a data transformation function of which the result is the service data of the candidate service. We will now address the input of this transformation function and the consequences for the conditions under which the function can be applied. Input for the specification rules that generate service data is:

- case attributes;
- service attributes of already contracted services.

If a transformation function is expressed in terms of case attributes only, it is possible to specify the service data at the start of the service contracting process as a whole. If a specification rule uses one or more service attributes, it always concerns service attributes of which the value becomes known to the service client during the negotiation and/or the execution phase (otherwise, the information would be available as case attribute). Consequently, it is not possible to specify the service data at the start of the service contracting process. Instead, the service data can only be specified after the value of the service attributes used in the specification rules has become available. An example of the use of a service attribute in the specification rules involves booking a flight to London and a rental car at the airport of destination. Since London has different airports, the airport of destination depends on the flight that is selected and booked by the service client. However, the service data of the rental car service requires the airport where the car will be picked up. Therefore, the specification rules that are used to generate the service data of the rental car candidate service uses the ‘airport of destination’ service attribute from the flight service.

We will now define a grammar for the specification rules used to create the initial value of service data. Again, the presented grammar has a limited expressive power, but can be replaced by another grammar if required. A specification rule consists of one or more *assignments* in which the left-hand side is a reference to a service attribute and the right-hand side is an expression in which case attributes and service attributes of earlier contracted services for the same business case can be operands. The grammar uses the grammar for constraints in Figure 48 and Figure 49 with one extension. This extension is required because the input of the specification rules consists of multiple hierarchic data sets: the case data and the service data of other candidate services. Therefore, we prefix each attribute reference with a reference to the data set it comes from. Since each service contracting process relates to exactly one business case, we can use the prefix ‘CASE.’ to reference the case data in a service contracting process. Furthermore, since each candidate service type has its own unique identification, the service data of a candidate service can be referenced by using its type identification as prefix.

DataRef	:=	('CASE' Name) ':' (EntityType) ⁺ AttributeType
Assignment	:=	DataRef '=' Expr (' ' Expr)*
SpecRules	:=	Assignment*

Figure 67 Grammar for specification rules

We will give some examples to illustrate the use of the grammar. The simplest example is the assignment of a constant to a service attribute, e.g.:

Pre-carriage : [TRANSPORT] [PLACE OF LOADING] (name) = 'Industrieweg 34'
--

It is also possible to assign the value of a case attribute to a service attribute, e.g.:

Pre-carriage : [TRANSPORT] (date of delivery) = CASE : [ORDER] (requested delivery date)

The right hand side of the assignment can also be an *expression* of two or more case attributes, e.g. a string concatenation:

Pre-carriage : [PLACE OF DELIVERY] (address) = CASE : [ORDER] [CUSTOMER] (street) & ' ' & CASE : [ORDER] [CUSTOMER] (number)
--

An example of a specification rule in which a service attribute is created based on a service attribute in an earlier contracted service is:

Main-carriage : [TRANSPORT] (earliest date of loading) = Pre-carriage : [TRANSPORT] (planned date of delivery)

Finally, an assignment can contain the '|' operator. When an assignment has the form 'a | b | c | ...' the processor that executes it will evaluate the expressions a, b and c from left to right. It will return the value of the first expression that is not empty. This construct can for instance be used when the service data of a candidate service depends on an earlier contracted service, for which two or more alternative candidate service types are defined.

Main-carriage : [TRANSPORT] (earliest date of loading) = Pre-carriage1 : [TRANSPORT] (planned date of delivery) Pre-carriage2 : [TRANSPORT] (planned date of delivery)
--

A grammar for the 'constraints' attribute

The 'constraints' attribute of a 'CANDIDATE SERVICE TYPE' entity is used to model whether a candidate service is required or not for a specific business case depending on the case data of the business case. For example, a service of type 'insurance' may be required only for business cases of which the insured value exceeds a minimum value. Likewise, if external logistics is outsourced to service providers, a business case with a domestic delivery address may require different services than a business case with a foreign delivery address. So, the constraints on a

candidate service type can be used to contract different services depending on differences in case data. The grammar of the ‘*constraints*’ attribute is already defined in Figure 49.

A grammar for the ‘*trigger*’ attribute

The role of the ‘*trigger*’ attribute is to start the contracting process for a candidate service at the right moment, or maybe not at all. When multiple candidate services are involved in a service contracting process, the contracting process for each particular candidate service must be triggered at one of the following events, or a combination of these events:

- the start of the entire service contracting process;
- the negotiation for another service ended *with* a contract (committed);
- the negotiation for another service ended *without* a contract (failed);
- the execution of another service was completed (completed);
- the execution of another service was aborted (aborted);

A trigger is a Boolean expression in which the events listed above are operands. An event is the completion of a contracting phase with a certain end-state. It is defined by the type identification of the candidate service, the identification of the contracting phase and the identification of the end-state. Therefore, the grammar of a ‘*trigger*’ attribute is defined by the element ‘Trigger’:

```

Event          := ServiceId '=' ('COMMITTED' | 'FAILED' | 'COMPLETED' | 'ABORTED')
CompositeEvent := Event | '(' CompositeEvent Opr3 CompositeEvent ')'
Trigger        := ServiceId ':' ('NEGOTIATION' | 'EXECUTION')
                'AFTER' CompositeEvent

```

Figure 68 Grammar for the ‘*trigger*’ attribute

We will give a number of examples to illustrate the use of the grammar. The first example shows a trigger for a candidate service ‘transport’ where two candidate services ‘order1’ and ‘order2’ must have resulted in a service contract before the contracting process of the ‘transport’ candidate service starts.

```

Transport : NEGOTIATION AFTER
           (Order1 = COMMITTED AND Order2 = COMMITTED)

```

Another example shows a trigger for a candidate service type ‘order2’ which is an alternative for another candidate service ‘order1’:

```

Order2 : NEGOTIATION AFTER Order1 = FAILED

```

A ‘*trigger*’ attribute defines a sequence in the contracting phases of different candidate services for one business case. There are two main reasons for modelling a sequential relationship between two candidate service types A and B.

- **Data dependencies in specification rules**

The specification of candidate service B requires a service attribute of candidate service A that becomes available when the contract for candidate service A is concluded. A typical example of such an attribute is the ‘earliest start time’ of B which is equal to the ‘promised

completion time’ of A. In those cases, the negotiation of candidate services A and B can not be performed in parallel, but must be performed sequentially.

- **Data dependencies in *constraints***

A candidate service B is required under conditions defined by the ‘*constraints*’ attribute of the candidate service type. If this ‘*constraints*’ attribute uses a service attribute of another candidate service A that becomes known when the service contract for candidate service A is established, the ‘*trigger*’ attribute of candidate service B must indicate that candidate service A must have been contracted first.

We will now focus on the situation where candidate services must be executed consecutively and are therefore related by their start- and end-times. Clearly, a sequential approach in contracting these candidate services is required. There is however a choice in the sequence used.

- **Forward scheduling**

Forward scheduling starts with specification and negotiation of the service that must be executed first. The earliest start time is derived from the case data. During the negotiation phase or the execution phase, the planned start time and planned completion time is received from the service provider. This information is then used to specify the earliest start time of the second service, etc.

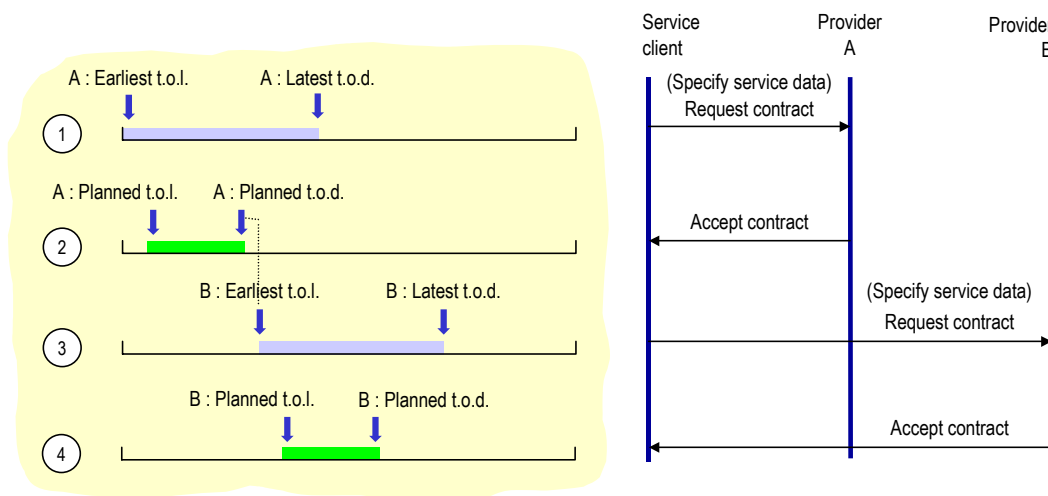


Figure 69 Example of forward scheduling

- **Backward scheduling**

Backward scheduling starts with specification and negotiation of the service that must be executed last. The latest completion time is derived from the case data. During the negotiation phase or the execution phase, the planned start time and planned completion time is received from the service provider. This information is then used to specify the latest completion time of the service that must be executed previously, etc.

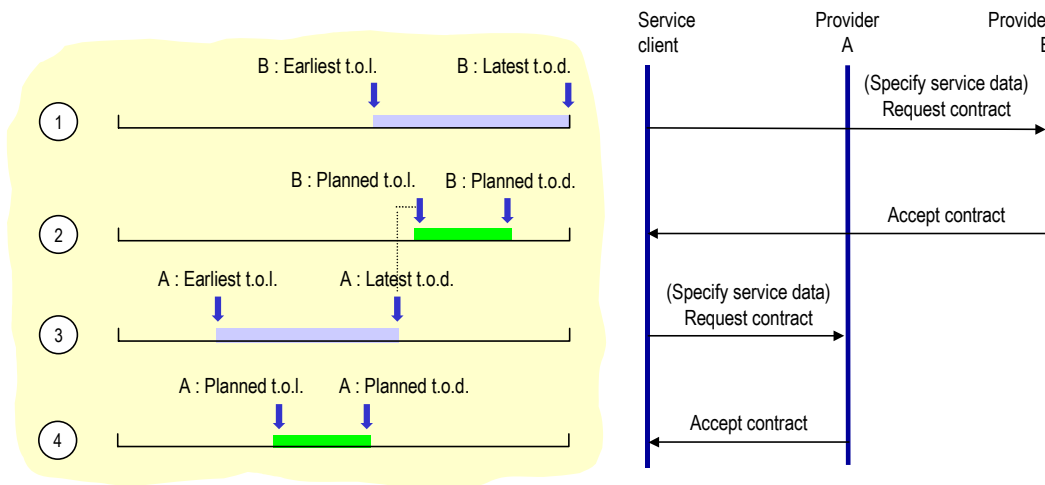
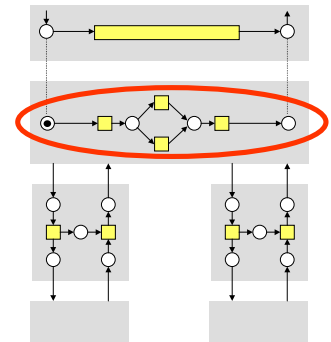


Figure 70 Example of backward scheduling

2.6 Construction of the contracting workflow

A contracting workflow defines the structure and behaviour of a service contracting process. We will use high level coloured Petri nets and functional data models as modelling techniques for contracting workflows. This section defines the structure of the contracting workflow, standard building blocks of which the contracting workflow can be composed and construction rules by which the contracting workflow can be derived from the available services (Figure 46) and the contracting requirements (Figure 66).



2.6.1 Basic operations in service contracting processes

As we have seen in Chapter 1, workflow management is about separation of execution and control. We will follow the same approach in this chapter, where we first define the state data of service contracting processes and a set of standard operations on that state data (execution) after which we use this result to define a workflow that invokes these standard operations in the right order and with the right parameters (control). The structure of the state data in a service contracting process is defined by Figure 71.

The basis of the state data is the entity ‘CASE’ that models a business case for which a service contracting process is executed. The ‘identification’ attribute of the ‘CASE’ entity is a unique identification for the business case. The case type to which a business case belongs is modelled by the attribute ‘type identification’ which is a reference to a ‘CASE TYPE’ entity. The ‘status’ attribute contains a value from the set {‘accepted’, ‘rejected’} and indicates whether the case data is valid and the service contracting process can be executed for this business case or not. The ‘date’ and ‘time’ attributes model the date and time at which the service contracting process started for the business case. Finally, the ‘data’ attribute contains the entire case data.

The ‘CANDIDATE SERVICE’ entity models a candidate service of a specific candidate service type involved in the service contracting process for a specific case. The ‘*type identification*’ attribute is a reference to the corresponding ‘CANDIDATE SERVICE TYPE’ entity. The ‘*provider identification*’ attribute is the unique identification of the service provider with which a contract is established, and is filled when the negotiation phase ends successfully. The ‘*status*’ attribute indicates whether the candidate service is required or must be skipped. Finally, the ‘*data*’ attribute models the candidate service data which is filled for required candidate services only.

The ‘AVAILABLE PROVIDER’ entity models a service provider from which the candidate service can be contracted. The ‘*identification*’ attribute is a reference to a ‘SELECTED PROVIDER’ entity related to the corresponding ‘CANDIDATE SERVICE TYPE’ entity. The ‘*preference*’ attribute defines a ranking between available providers and is an integer 1, 2, 3, ...

The ‘VIOLATION’ entity models a violation that occurred for a specific candidate service. The ‘*type identification*’ attribute refers to a ‘VIOLATION TYPE’ entity. The ‘*date*’ and ‘*time*’ attributes model the date and time at which the violation occurred.

The ‘TRANSACTION’ entity models a business transaction with the service provider identified by the related ‘AVAILABLE PROVIDER’ entity and for the candidate service identified by the related ‘CANDIDATE SERVICE’ entity. The ‘*identification*’ attribute is a unique identifier for the business transaction, assigned by the service client. The ‘*state*’ attribute models the state in the transaction protocol the business transaction is in.

The ‘MESSAGE’ entity models a message in a business transaction. The ‘*identification*’ attribute is a unique identifier for the message. The ‘*type identification*’ is a reference to the corresponding ‘MESSAGE TYPE’ entity. The ‘*date*’ and ‘*time*’ attributes model the date and time at which the message was created. The ‘*direction*’ attribute has the value ‘in’ if the message is inbound and the value ‘out’ if the message is outbound (from the perspective of the service client). The ‘*status*’ attribute has the value ‘accepted’ if the message data conforms to the message type data model and has the value ‘rejected’ otherwise. Finally, the ‘*data*’ attribute contains the entire message data.

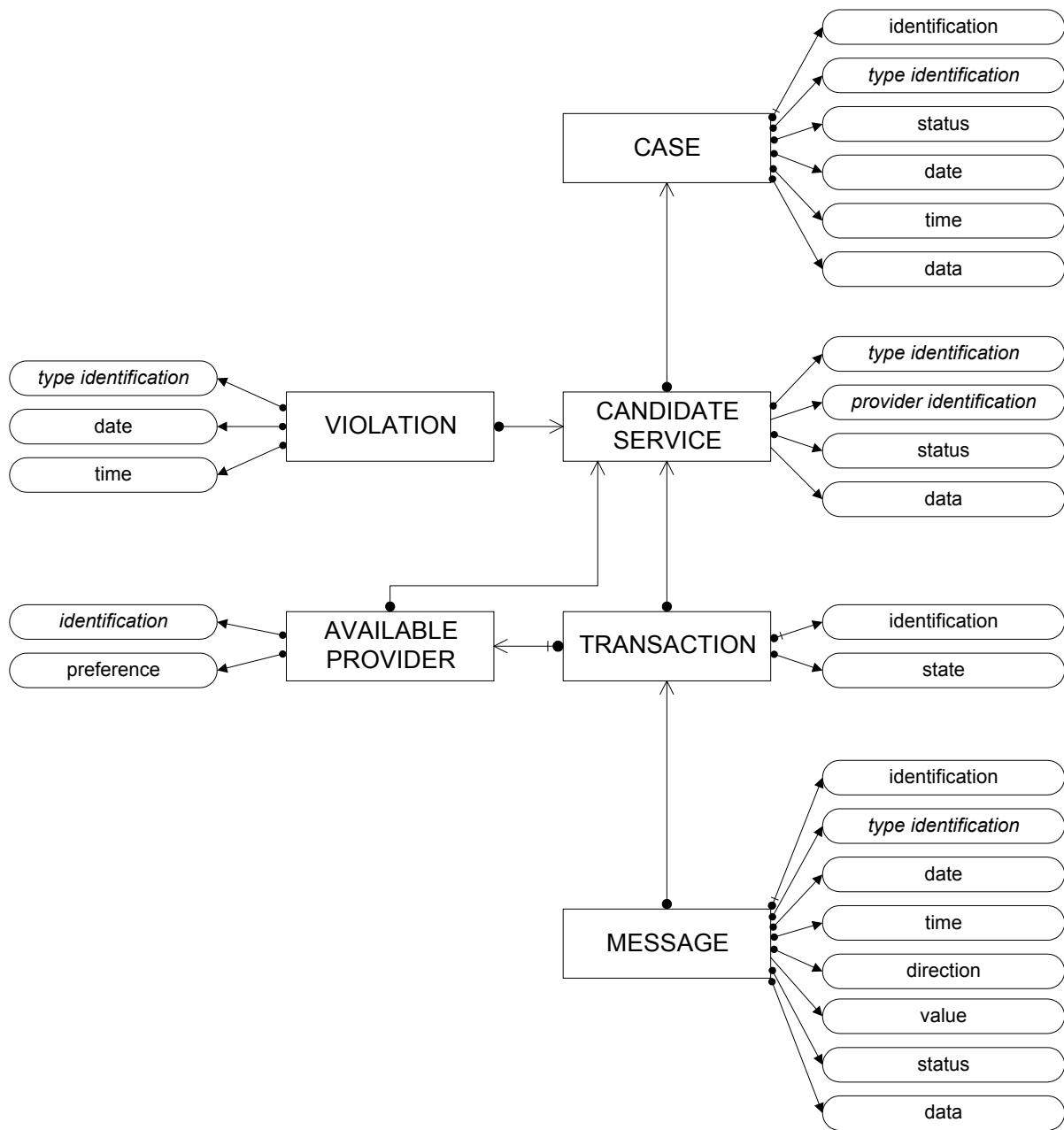


Figure 71 State data of a service contracting process

The structure of the configuration data used by the standard operations in a service contracting process is defined in Figure 72. The colour of this ‘*configuration*’ store is derived by joining the relevant information from the ‘*available services*’ token in Figure 46 and the ‘*contracting requirements*’ token in Figure 66.

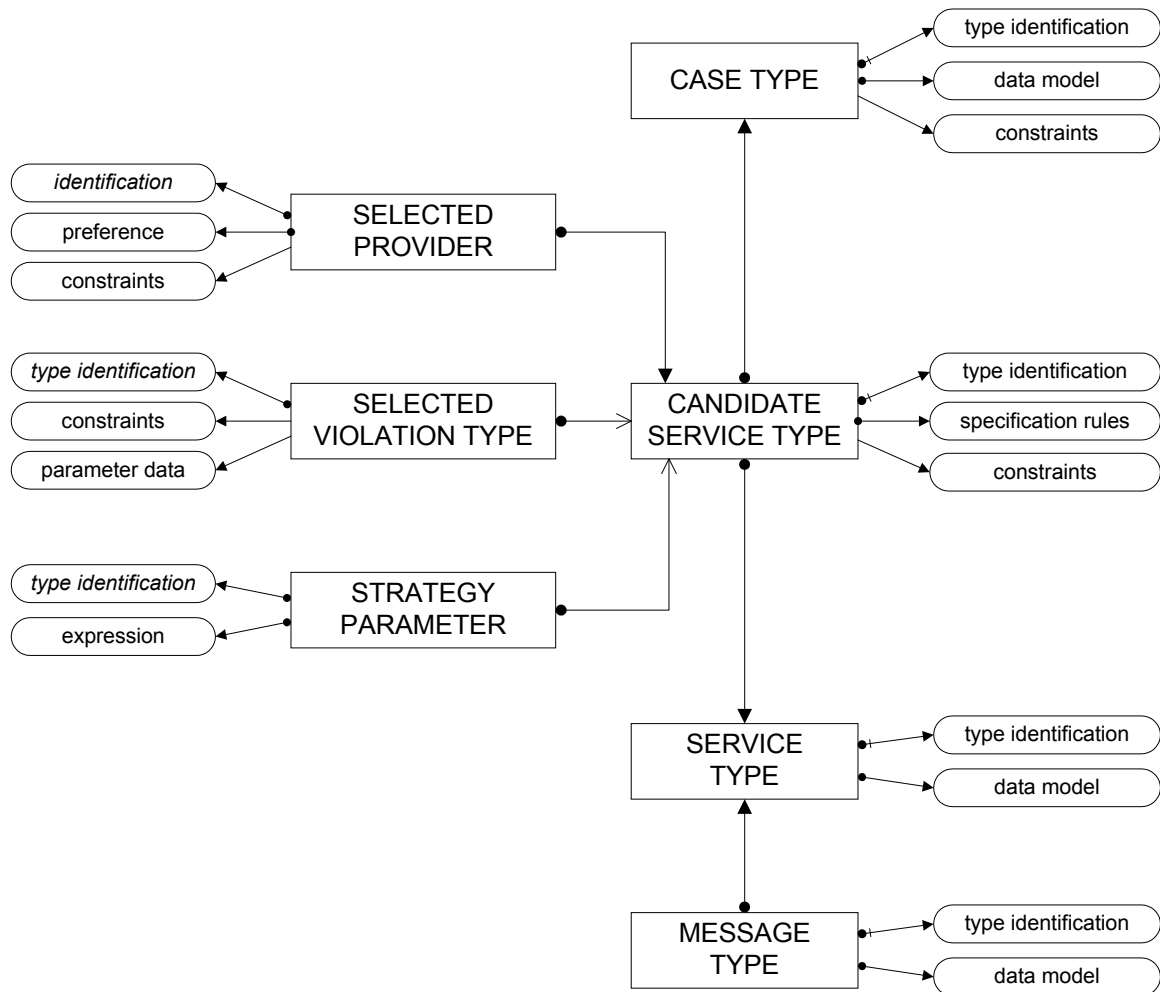


Figure 72 Configuration data used by the standard operations on the state data

We will now define the standard operations on the state data. Each operation has a name, one or more parameters and a description of the actions that are performed on the state data.

- Operation: **‘validate case’**

Parameters: P0: case identification

This operation is used to check the validity of a specific business case, before the actual service contracting starts. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Next, the entity ‘CASE TYPE’ entity of which the attribute ‘*type identification*’ equals the attribute ‘*type identification*’ in the selected ‘CASE’ entity is selected. If this ‘CASE TYPE’ entity does not exist, the value of the ‘*status*’ attribute in the selected ‘CASE’ entity is changed to ‘rejected’. Otherwise, the ‘*constraints*’ attribute in the selected ‘CASE TYPE’ entity is applied to the ‘*data*’ attribute of the selected ‘CASE’ entity. If the constraints are not fulfilled, the value of the ‘*status*’ attribute in the selected ‘CASE’ entity is changed to ‘rejected’. Otherwise the value of the ‘*status*’ attribute is changed to ‘accepted’.

- Operation: **‘determine candidate service status’**

Parameters: P0: case identification
P1: candidate service type identification

This operation is used to check whether a specific candidate service type is required for a specific business case or can be skipped. First, the ‘CASE’ entity of which the attribute ‘*identification*’ equals parameter P0 is selected. Next, the ‘CASE TYPE’ entity of which the attribute ‘*type identification*’ equals the attribute ‘*type identification*’ in the selected ‘CASE’ entity is selected. To perform the check, the operation applies the ‘*constraints*’ attribute in the ‘CANDIDATE SERVICE TYPE’ entity, of which the attribute ‘*type identification*’ equals parameter P1, to the ‘*data*’ attribute of the selected ‘CASE’ entity and the ‘*data*’ attribute of the already existing ‘CANDIDATE SERVICE’ entities related to the selected ‘CASE’ entity. Here after, a new ‘CANDIDATE SERVICE’ entity is created which is related to the selected ‘CASE’ entity and of which the ‘*type identification*’ attribute is equal to parameter P1. If the constraints are fulfilled, the ‘*status*’ attribute of the new ‘CANDIDATE SERVICE’ entity gets the value ‘required’. Otherwise, the ‘*status*’ attribute of the new ‘CANDIDATE SERVICE’ entity gets the value ‘skipped’.

- Operation: **‘specify candidate service data’**

Parameters: P0: case identification
P1: candidate service type identification

This operation is used to assign an initial value to the ‘*data*’ attribute of a specific ‘CANDIDATE SERVICE’ entity, which represents the service data with which the negotiation phase starts. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE TYPE’ entity of which the attribute ‘*type identification*’ equals parameter P1 is selected. Thereafter, the ‘CANDIDATE SERVICE’ entity related to the selected ‘CASE’ entity and the selected ‘CANDIDATE SERVICE TYPE’ entity is selected. The ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity is changed into the result of applying the ‘*specification rules*’ attribute of the selected ‘CANDIDATE SERVICE TYPE’ entity to the ‘*data*’ attribute of the selected ‘CASE’ entity and the ‘*data*’ attribute of all other ‘CANDIDATE SERVICE’ entities related to the selected ‘CASE’ entity.

- Operation: **‘determine available providers’**

Parameters: P0: case identification
P1: candidate service type identification

This operation is used to determine which service providers are available to start a negotiation with for a specific candidate service. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE TYPE’ entity of which the attribute ‘*type identification*’ equals parameter P1 is selected. Thereafter, the ‘CANDIDATE SERVICE’ entity related to the selected ‘CASE’ entity and the selected ‘CANDIDATE SERVICE TYPE’ entity is selected. For each ‘SELECTED PROVIDER’ entity related to the selected ‘CANDIDATE SERVICE TYPE’ entity the operation applies the ‘*constraints*’ attribute to the ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity. If the constraints are fulfilled, a new ‘AVAILABLE PROVIDER’ entity is created related to the selected ‘CANDIDATE SERVICE’ entity. The ‘*preference*’ attributes of the created ‘AVAILABLE PROVIDER’ entities form a sequence 1, 2, 3, ...

- Operation: **‘create transaction’**

Parameters: P0: case identification
 P1: candidate service type identification
 P2: provider identification

This operation is used to create a new business transaction for a specific candidate service and service provider. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE’ entity related to the selected ‘CASE’ entity and of which the attribute ‘*type identification*’ equals parameter P1 is selected. Finally, the ‘AVAILABLE PROVIDER’ entity of which the attribute ‘*identification*’ equals parameter P2 and which is related to the selected ‘CANDIDATE SERVICE’ entity, is selected. A new ‘TRANSACTION’ entity, related to the selected ‘CANDIDATE SERVICE’ entity and the selected ‘AVAILABLE PROVIDER’ entity, is created. The value of the ‘*identification*’ attribute is a unique identifier and the value of the ‘*state*’ attribute is ‘start’.

- Operation: **‘create outbound message’**

Parameters: P0: case identification
 P1: candidate service type identification
 P2: provider identification
 P3: message type

This operation is used to create a new outbound message in an existing business transaction. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE’ entity, related to the selected ‘CASE’ entity, and of which the ‘*type identification*’ attribute equals parameter P1 is selected. Next, the ‘TRANSACTION’ entity, related to the selected ‘CANDIDATE SERVICE’ entity, and related to the ‘AVAILABLE PROVIDER’ entity of which the ‘*identification*’ attribute equals parameter P2 is selected. Finally, the ‘MESSAGE TYPE’ entity of which the ‘*type identification*’ attribute equals parameter P3 is selected. A new ‘MESSAGE’ entity is then created, related to the selected ‘TRANSACTION’ entity, of which the ‘*identification*’ attribute is made equal to a unique identifier, the ‘*type identification*’ attribute is made equal to parameter P3, the ‘*direction*’ attribute is made equal to ‘out’ and the ‘*date*’ and ‘*time*’ attributes are made equal to the current system date and time. The ‘*data*’ attribute of the new ‘MESSAGE’ entity is derived by copying the entities and attributes referenced in the ‘*data model*’ attribute of the selected ‘MESSAGE TYPE’ entity from the ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity.

- Operation: **‘store inbound message’**

Parameters: P0: case identification
 P1: candidate service type identification
 P2: provider identification
 P3: message type
 P4: message data

This operation is used to store a received inbound message in its business transaction. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE’ entity, related to the selected ‘CASE’ entity, and of which the ‘*type identification*’ attribute equals parameter P1 is selected. Next, the ‘TRANSACTION’ entity, related to the selected ‘CANDIDATE SERVICE’ entity, and related to the ‘AVAILABLE PROVIDER’ entity of which the ‘*identification*’ attribute equals parameter P2 is selected. Finally, the ‘MESSAGE TYPE’ entity of which the ‘*type identification*’ attribute equals parameter P3 is selected. A new ‘MESSAGE’ entity is then created, related to the selected

‘TRANSACTION’ entity, of which the ‘*identification*’ attribute is made equal to a unique identifier, the ‘*type identification*’ attribute is made equal to parameter P3, the ‘*direction*’ attribute is made equal to ‘in’, the ‘*date*’ and ‘*time*’ attributes are made equal to the current system date and time and the ‘*data*’ attribute is made equal to parameter P4.

- Operation: **‘process inbound message’**

Parameters: P0: case identification
 P1: candidate service type identification
 P2: provider identification
 P3: message identification

This operation is used to update the service data of a specific candidate service with the message data of a specific inbound message. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE’ entity, related to the selected ‘CASE’ entity, and of which the ‘*type identification*’ attribute equals parameter P1 is selected. Finally, the ‘MESSAGE’ entity of which the attribute ‘*identification*’ equals parameter P3 is selected. Here after, the ‘*data*’ attribute of the selected ‘MESSAGE’ entity is used to update the value of the ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity.

- Operation: **‘check for violations’**

Parameters: P0: case identification
 P1: candidate service type identification

This operation is used to check if the service data of a specific candidate service indicates that a violation has occurred. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE TYPE’ entity of which the attribute ‘*type identification*’ equals parameter P1 is selected. Thereafter, the ‘CANDIDATE SERVICE’ entity related to the selected ‘CASE’ entity and the selected ‘CANDIDATE SERVICE TYPE’ entity is selected. For each ‘SELECTED VIOLATION TYPE’ entity, related to the selected ‘CANDIDATE SERVICE TYPE’ entity, the operation applies the ‘*constraints*’ attribute in the ‘SELECTED VIOLATION TYPE’ entity to the ‘*data*’ attribute in the selected ‘CANDIDATE SERVICE’ entity. If the constraints are not fulfilled, an entity ‘VIOLATION’ related to the selected ‘CANDIDATE SERVICE’ entity is created and of which the ‘*type identification*’ attribute is equal to the ‘*type identification*’ attribute of the ‘SELECTED VIOLATION TYPE’ entity.

- Operation: **‘determine message value’**

Parameters: P0: case identification
 P1: candidate service type identification
 P2: message identification
 P3: strategy parameter type identification

This operation is used to apply a function to the message data of an inbound message that yields a value which is used in the contracting strategy, e.g. to be able to compare offers in order to select the ‘best’ offer from a set of received offers. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE TYPE’ entity of which the attribute ‘*type identification*’ equals parameter P1 is selected. Thereafter, the ‘CANDIDATE SERVICE’ entity related to the selected ‘CASE’ entity and the selected ‘CANDIDATE SERVICE TYPE’ entity is selected. Next, the ‘MESSAGE’ entity of which the ‘*identification*’ attribute equals parameter P2 is selected. Finally, the ‘STRATEGY PARAMETER’ entity related to the selected ‘CANDIDATE SERVICE TYPE’ entity and of which the ‘*type identification*’ attribute equals parameter P3 is selected. The expression in attribute

‘*expression*’ in the selected ‘STRATEGY PARAMETER’ entity is then applied to the ‘*data*’ attribute of the selected ‘MESSAGE’ entity and the ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity. The resulting value is then stored in the attribute ‘*value*’ of the selected ‘MESSAGE’ entity.

- Operation: **‘adjust candidate service data’**

Parameters: P0: case identification
 P1: candidate service type identification
 P2: strategy parameter type identification

This operation is used to update the service data of a specific candidate service, in order to create data for an outbound message that is not present in the service data yet. This operation is for instance used when a counter offer must be made according to the contracting strategy. First, the ‘CASE’ entity of which the ‘*identification*’ attribute equals parameter P0 is selected. Second, the ‘CANDIDATE SERVICE TYPE’ entity of which the attribute ‘*type identification*’ equals parameter P1 is selected. Thereafter, the ‘CANDIDATE SERVICE’ entity related to the selected ‘CASE’ entity and the selected ‘CANDIDATE SERVICE TYPE’ entity is selected. Finally, the ‘STRATEGY PARAMETER’ entity, related to the selected ‘CANDIDATE SERVICE TYPE’ entity, and of which the ‘*type identification*’ attribute equals parameter P2 is selected. The expression in attribute ‘*expression*’ in the selected ‘STRATEGY PARAMETER’ entity is then applied to the ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity. The value of the ‘*data*’ attribute of the selected ‘CANDIDATE SERVICE’ entity is then replaced by the result of that transformation.

The create-read-update matrix in Figure 73 shows the relationship between the entities in the state data and configuration data on one hand and the standard operations on the other hand.

	CASE	CANDIDATE SERVICE	AVAILABLE PROVIDER	TRANSACTION	MESSAGE	VIOLATION	CASE TYPE	CANDIDATE SERVICE TYPE	SELECTED PROVIDER	STRATEGY PARAMETER	SELECTED VIOLATION TYPE	SERVICE TYPE	MESSAGE TYPE
validate case	RU						R						
determine candidate service status	R	CR					R	R					
specify candidate service data	R	RU					R	R				R	
determine available providers	R	R	C				R	R					
create transaction	R	R	R	C									
create outbound message	R	R	R	R	C							R	R
store inbound message	R	R	R	R	C							R	R
process inbound message	R	RU	R	R	R							R	R
check for violations	R	R				C	R				R	R	
determine message value	R	R			RU		R		R				R
adjust candidate service data	R	RU		R	R		R		R			R	

Figure 73 Create-Read-Update matrix of standard operations in the service contracting process

2.6.2 The contracting workflow

In the previous section we have defined the basic operations of which a service contracting process exists (execution). This section uses this basic operations to define a workflow that models an entire service contracting process, and in which the basic operations are *invoked* with the right parameters and in the right order (control).

We use high-level coloured Petri nets to model the contracting workflow, mainly because of the formal basis that allows analysis techniques to be applied. On the highest level of abstraction, a contracting workflow is a system with the input and output places as shown in Figure 74. The contracting workflow has two input places, ‘start’ and ‘message in’ and two output places, ‘end’ and ‘message out’. Because of this structure, the contracting workflow can never be a workflow net according to the definition given before. As a consequence, we can not apply standard analysis techniques to prove important properties like soundness. Clearly, this is a very undesirable situation. As we have stated in Section 1.4.4, one of the major issues in interoperability between workflows in different organisations is to check the correctness of the workflow, for example to prevent deadlocks. The solution to this problem is to consider the contracting workflow without the places ‘message in’ and ‘message out’ and without the connectors that connect transitions with these places. The resulting contracting workflow must then be a sound WF-net.

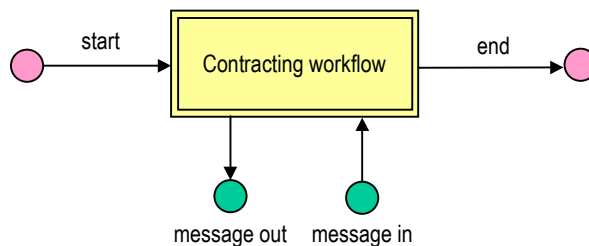


Figure 74 Input and output places of the contracting workflow

We will now define the colour of the places in the contracting workflow. First, the colour of places that model messages between service client and service provider (‘message in’ and ‘message out’) is a complex of which the object model is defined in Figure 31. The configuration data defined in Figure 72 is modelled by a store ‘*configuration*’ to which all transitions in the contracting workflow have access. For this reason, we do not draw the connectors with the ‘*configuration*’ store in order to keep the Petri nets simple. Finally, the case data is modelled as case tokens that flow through the contracting workflow. Each case is represented by one or more (in case of parallelism in the contracting workflow) case tokens, and each case token contains exactly one ‘CASE’ entity. The colour of a case token consists of a complex of which the object model is given in Figure 71 extended with an entity ‘CONTROL’ that models the control data used by the transitions in the contracting workflow for routing purposes. Therefore, the colour of a case token is a complex of which the object model is defined in Figure 75. The white entities and attributes are used to model the state data in a service contracting process: cases, candidate services, available providers, transactions, messages and violations. This information is persistent data and is relevant during the service contracting process and after the service contracting process has ended (e.g. for monitoring purposes). The grey entities and attributes are used in the contracting workflow as control information that is used only temporally. At the start of the service contracting process, the case token only contains a ‘CASE’ entity and a ‘CONTROL’ entity of which all attributes are empty. During the service contracting process, the state data is modified by applying the standard operations defined in Section 2.6.1.

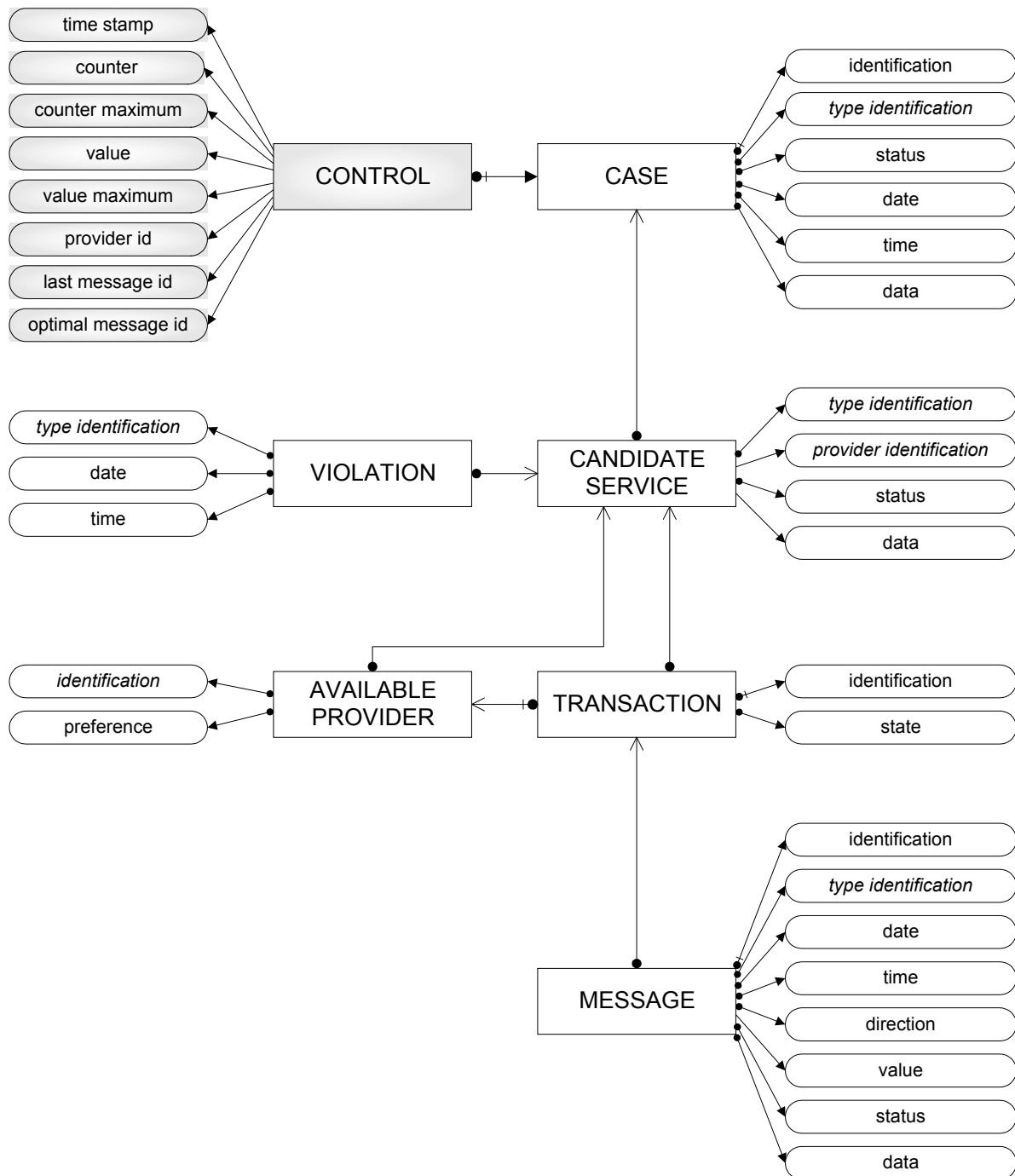


Figure 75 Object model of the case token in the contracting workflow

We will now describe the function of the attributes in the ‘CONTROL’ entity. The ‘time stamp’ attribute is used to model timeouts in real-time contracting processes. The ‘counter’ and ‘counter maximum’ attributes are used as control variables for iterations (loops). The ‘value’ attribute is used to store the result of the last ‘determine message value’ operation, whereas the ‘value maximum’ attribute is used to store the current optimum value in an optimisation process. The ‘provider id’ attribute is used in iterations on the list of available providers to store the identification of the current provider. Finally, the ‘last message id’ attribute is used to store the identification of the last received message, whereas the ‘optimal message id’ attribute is used to store the identification of the message that yielded the ‘value maximum’ attribute.

2.6.3 Standard transitions for the contracting workflow

Before discussing contracting workflows that involve two or more candidate services, we will describe a standard building block for contracting *one* candidate service. To define the structure of that building block, we will now concentrate on the consecutive steps in the process and the possible results of these steps. Step one is to evaluate the constraints defined for the candidate service type, to decide whether the candidate service is required for a specific business case or not. If a candidate service is required, the actual service contracting process is continued with four consecutive phases: specification, negotiation, execution and acceptance. The specification phase has one possible outcome: the specified service data for the required service. The negotiation phase has two possible outcomes, either the negotiation is completed successfully and ends with a contract or the negotiation fails and ends without a contract. The execution phase has also two possible outcomes: either the execution is completed (with or without violations) or the execution is aborted. Finally, the acceptance phase has exactly one possible outcome: the completion of the transaction (with or without claim). Concluding, the Petri net that models the contracting process for one candidate service type is given in Figure 76. The ‘*establish contract*’ system models the constraint checking, the specification phase and the negotiation phase, whereas the ‘*monitor contract*’ system models the execution and the acceptance phase.

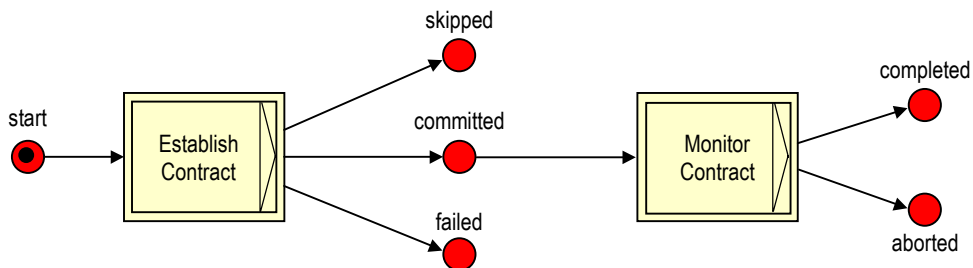


Figure 76 Petri net modelling of contracting one candidate service type

In the rest of this section we will define the structure of the ‘*establish contract*’ and ‘*monitor contract*’ systems. We will define a number of standard transitions, from which these systems can be composed. Two types of standard transitions are distinguished. The first type is formed by the standard transitions that represent a standard operation on the state data defined in Section 2.6.1. The second type is formed by the transitions that do not change the state data but are used for the routing of tokens in the contracting workflow only. A first decomposition of the systems ‘*establish contract*’ and ‘*monitor contract*’ is given in Figure 77.

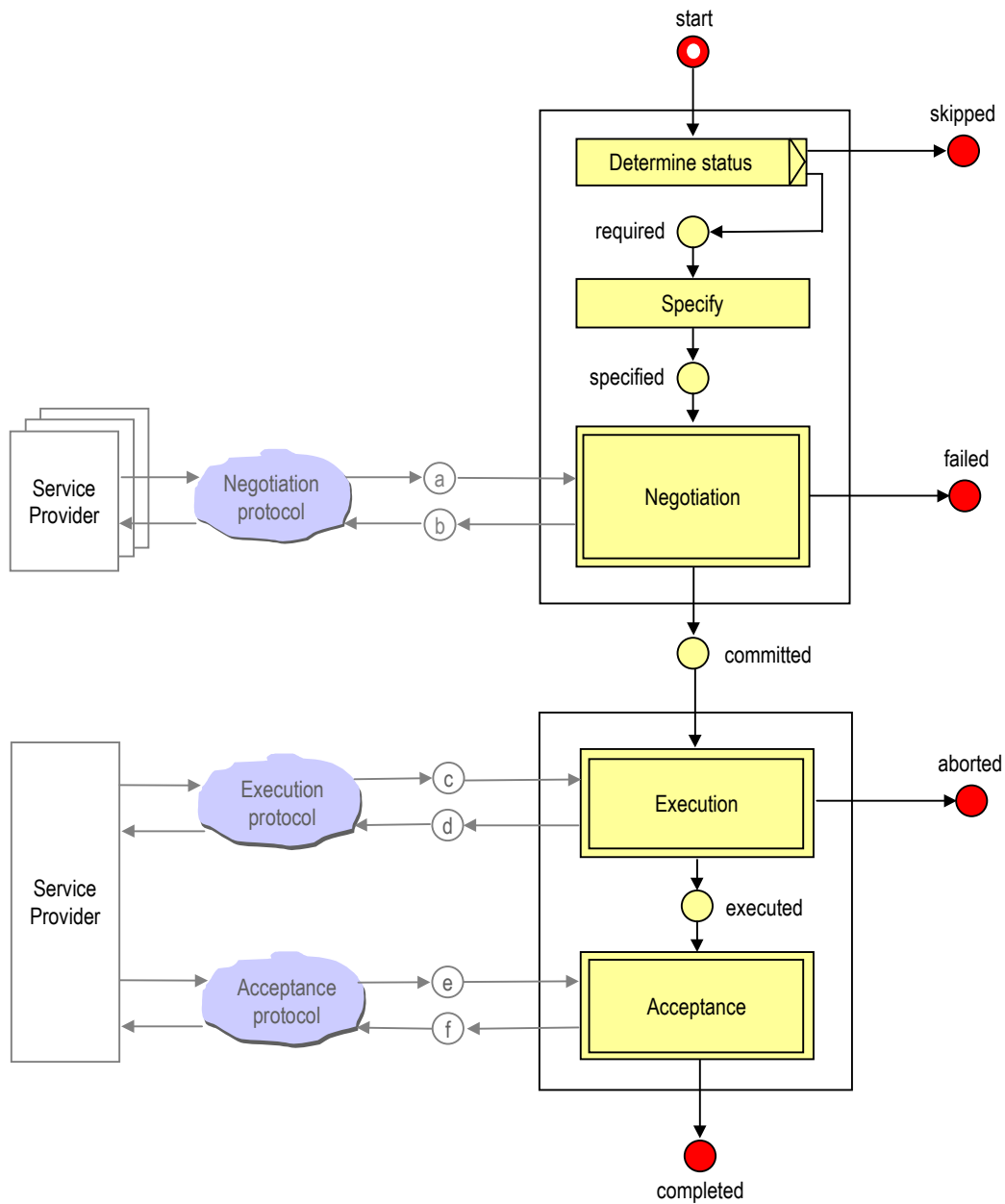


Figure 77 Petri net modelling of system 'establish contract' and 'monitor contract'

This Petri net in Figure 77 contains two standard transitions:

- **Standard transition 'determine status'**

This standard transition invokes the standard operation 'determine candidate service status' and has the input and output places as shown in Figure 78. The transition has no precondition. When the transition fires, the standard operation 'determine candidate service status' is invoked with the following parameters:

- P0: case identification → 'identification' in 'CASE'
- P1: candidate service type identification → literal

This operation creates a new 'CANDIDATE SERVICE' entity in the state data. If the 'status' attribute of the created 'CANDIDATE SERVICE' entity is equal to 'required', the case token is produced in place 'required'. Otherwise, the case token is produced in place 'skipped'.

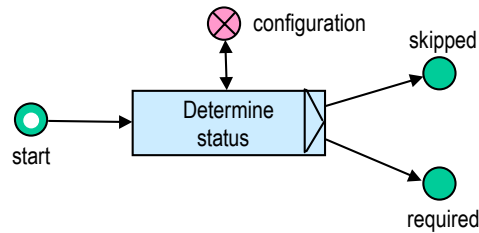


Figure 78 Standard transition 'determine status'

- **Standard transition 'specify'**

This standard transition invokes the standard operation 'specify candidate service data' and has the input and output places as shown in Figure 79. The transition has no precondition. When the transition fires, the standard operation 'specify candidate service data' is invoked with the following parameters:

P0: case identification → '*identification*' in 'CASE'
 P1: candidate service type identification → literal

The operation modifies the colour of the case token after which the case token is produced in place '*specified*'.

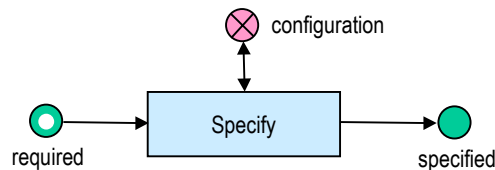


Figure 79 Standard transition 'specify'

In the following sections, we will discuss the structure of the transitions '*negotiation*', '*execution*' and '*acceptance*'. However, in order to be able to do this, we will first define standard transitions of which these larger '*negotiation*', '*execution*' and '*acceptance*' transitions can be composed. We will first define the standard transitions that correspond to a standard operation described in Section 2.6.1. There after, we will define standard transitions that do not change the state data, but are used for the control flow only.

- **Standard transition 'determine available providers'**

This standard transition invokes the standard operation 'determine available providers' and has the input and output places as shown in Figure 80. The transition has no precondition. When the transition fires, the standard operation 'determine available providers' is invoked with the following parameter values:

P0: case identification → '*identification*' in 'CASE'
 P1: candidate service type identification → literal

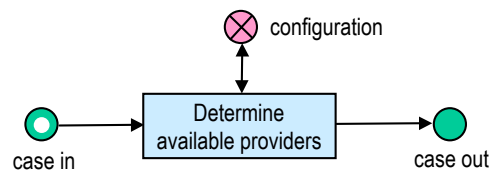


Figure 80 Standard transition 'determine available providers'

• **Standard transition ‘create transaction’**

This standard transition invokes the standard operation ‘create transaction’ and has the input and output places as shown in Figure 81. The transition has no precondition. When the transition fires, the standard operation ‘create transaction’ is invoked with the following parameter values:

- P0: case identification → ‘*identification*’ in ‘CASE’
- P1: candidate service type identification → literal
- P2: provider identification → ‘*provider id*’ in ‘CONTROL’

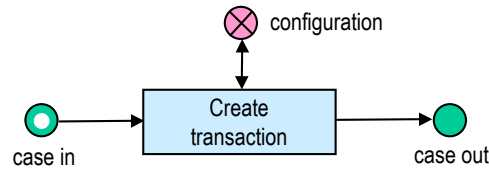


Figure 81 Standard transition ‘create transaction’

• **Standard transition ‘send <message type>’**

This standard transition invokes the standard operation ‘create outbound message’ and has the input and output places as shown in Figure 82. The transition has no precondition. When the transition fires, the standard operation ‘create outbound message’ is invoked with the following parameter values.

- P0: case identification → ‘*identification*’ in ‘CASE’
- P1: candidate service type identification → literal
- P2: provider identification → ‘*provider id*’ in ‘CONTROL’
- P3: message type → literal <message type>

After the operation is performed, the attributes of the ‘MESSAGE’ entity that has been created in the case token are used to create a message token that is produced in place <message type>.

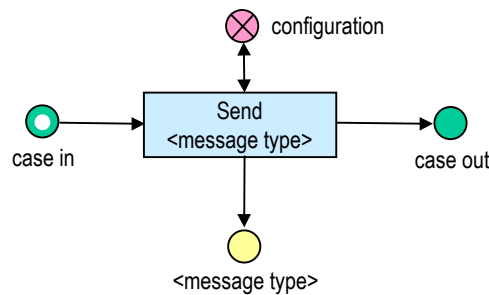


Figure 82 Standard transition ‘send message’

- **Standard transition: ‘receive <message type>’**

This standard transition invokes the standard operation ‘store inbound message’ and has the input and output places as shown in Figure 83. The transition has the following precondition: “the token in place ‘*case in*’ contains a ‘TRANSACTION’ entity of which the ‘*identification*’ attribute is equal to the ‘*transaction identification*’ attribute in the ‘MESSAGE’ entity of the token in place ‘<*message type*>’”. When the transition fires it consumes a token from place ‘*case in*’ and from place ‘<*message type*>’, after which the standard operation ‘store inbound message’ is invoked with the following parameter values.

P0: case identification	→	‘ <i>identification</i> ’ in ‘CASE’
P1: candidate service type identification	→	literal
P2: provider identification	→	‘ <i>sender identification</i> ’ in ‘MESSAGE’
P3: message type	→	‘ <i>type identification</i> ’ in ‘MESSAGE’
P4: message data	→	‘ <i>data</i> ’ in ‘MESSAGE’

After the operation is performed, the ‘*last message id*’ attribute in the ‘CONTROL’ entity is filled with the ‘*identification*’ attribute of the new ‘MESSAGE’ entity.

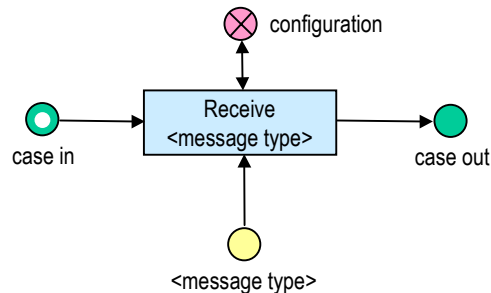


Figure 83 Standard transition ‘receive message’

- **Standard transition ‘process message’**

This standard transition invokes the standard operation ‘process inbound message’ and has the input and output places as shown in Figure 84. The transition has no precondition. When the transition fires, the standard operation ‘process inbound message’ is invoked with the following parameter values:

P0: case identification	→	‘ <i>identification</i> ’ in ‘CASE’
P1: candidate service type identification	→	literal
P2: message identification	→	‘ <i>last message id</i> ’ in ‘CONTROL’

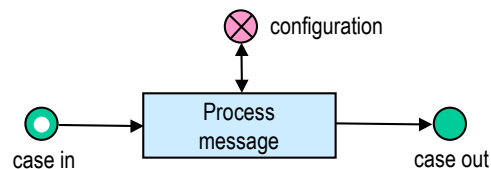


Figure 84 Standard transition ‘process message’

- **Standard transition ‘check for violations’**

This standard transition invokes the standard operation ‘check for violations’ and has the input and output places as shown in Figure 85. The transition has no precondition. When the

transition fires, the standard operation ‘check for violations’ is invoked with the following parameter values:

- P0: case identification → ‘*identification*’ in ‘CASE’
- P1: candidate service type identification → literal

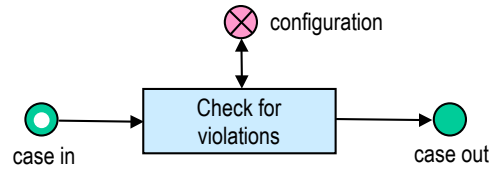


Figure 85 Standard transition ‘check for violations’

• **Standard transition ‘determine value’**

This standard transition invokes the standard operation ‘determine message value’ and has the input and output places as shown in Figure 86. The transition has no precondition. When the transition fires, the standard operation ‘determine message value’ is invoked with the following parameter values:

- P0: case identification → ‘*identification*’ in ‘CASE’
- P1: candidate service type identification → literal
- P2: message identification → ‘*last message id*’ in ‘CONTROL’
- P3: strategy parameter type identification → literal

After the operation is performed, the ‘*value*’ attribute in the ‘CONTROL’ entity is filled with the ‘*value*’ attribute of the ‘MESSAGE’ entity of which the ‘*identification*’ attribute equals the ‘*last message id*’ attribute in entity ‘CONTROL’. Finally, the case token is produced in place ‘*message values*’ and in place ‘*case out*’.

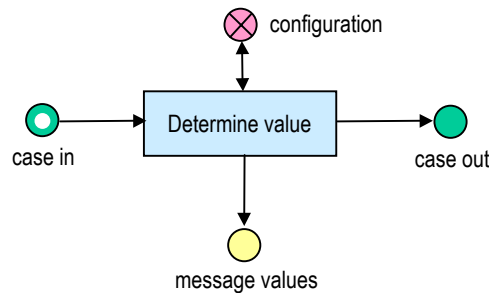


Figure 86 Standard transition ‘determine value’

• **Standard transition ‘adjust specification’**

This standard transition invokes the standard operation ‘adjust candidate service data’ and has the input and output places as shown in Figure 87. The transition has no precondition. When the transition fires, the standard operation ‘adjust candidate service data’ is invoked with the following parameter values:

- P0: case identification → ‘*identification*’ in ‘CASE’
- P1: candidate service type identification → literal
- P2: strategy parameter type identification → literal

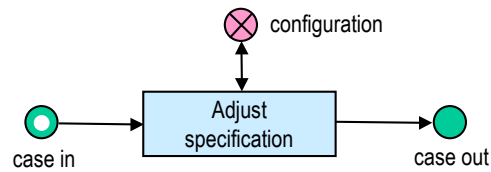


Figure 87 Standard transition 'adjust specification'

In addition to the standard transitions that create and change state data (candidate service, transaction, message, etc.), we will now present standard transitions that do not change the state data, but are used to control the flow of tokens only. These transitions modify the attributes of the 'CONTROL' entity that are used as parameters in consecutive transitions though. The next two standard transitions are used for an iteration on the available providers.

- **Standard transition 'select first service provider'**

This standard transition is used to fill the '*provider id*' attribute of the 'CONTROL' entity with the identification of the available provider with the highest preference. The input and output places of the transition are shown in Figure 88. The transition has no precondition and uses one parameter P1 with the identification of the candidate service type. When this transition fires, it consumes a case token from place '*case in*'. Here after, the 'AVAILABLE PROVIDER' entity of which attribute '*preference*' equals 1 and which is related to the 'CANDIDATE SERVICE' entity of which attribute '*type identification*' equals parameter P1 is selected. If the entity exists, the '*counter*' attribute of the 'CONTROL' entity is changed to 1 and the '*provider id*' attribute of the 'CONTROL' entity is changed to the '*identification*' attribute of the selected 'AVAILABLE PROVIDER' entity. Here after, the case token is produced in place '*available*'. If the searched 'AVAILABLE PROVIDER' entity does not exist, the case token is produced in place '*none available*'.

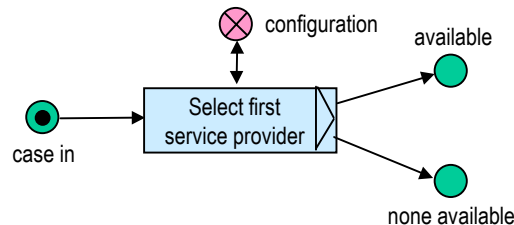


Figure 88 Standard transition 'select first service provider'

- **Standard transition: 'select next service provider'**

This standard transition is used to fill the '*provider id*' attribute of the 'CONTROL' entity with the identification of the next available provider relative to the provider identified by the '*provider id*' attribute in the 'CONTROL' entity. The input and output places of the transition are shown in Figure 89. The transition has no precondition and uses one parameter P1 with the identification of the candidate service type. When this transition fires, it consumes a case token from place '*case in*'. Here after, the 'AVAILABLE PROVIDER' entity of which attribute '*preference*' equals the attribute '*counter*' in the 'CONTROL' entity incremented by 1 and which is related to the 'CANDIDATE SERVICE' entity of which attribute '*type identification*' equals parameter P1 is selected. If the entity exists, the '*counter*' attribute of the 'CONTROL' entity is incremented by 1 and the '*provider id*' attribute of the 'CONTROL' entity is changed to the '*identification*' attribute of the selected 'AVAILABLE PROVIDER' entity. Here after, the case token is produced in place '*available*'. If the target 'AVAILABLE PROVIDER' entity does not exist, the case token is produced in place '*exhausted*'.

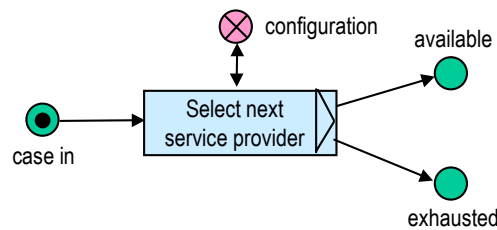


Figure 89 Standard transition 'select next service provider'

Another set of standard transitions is used for adding a time stamp to a case token and for signalling time outs. This is used to model the situation where the service client sends one or more messages and waits a period of time in which messages from service providers can be received.

- **Standard transition 'add time stamp'**

This standard transition is used to set the '*time stamp*' attribute of the 'CONTROL' entity to the current time. The transition has no precondition and uses no parameters. When the transition fires, it consumes a case token from place '*case in*'. Here after, the colour of the case token is changed by making the '*time stamp*' attribute of the 'CONTROL' entity equal to the current time. There after, the token is produced in place '*case out*'.

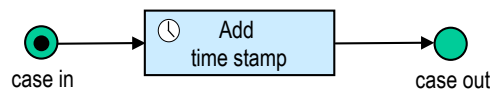


Figure 90 Standard transition 'add time stamp'

- **Standard transition 'timeout'**

This standard transition is used to model the expiration of a timer (time out). The transition uses one parameter P1 that models the length of the delay and has the precondition "the '*time stamp*' attribute of the 'CONTROL' entity of the token in place '*case in*', incremented with a delay P1, is equal to or earlier than the current time". When the transition fires, it consumes the token from place '*case in*' and produces it in place '*case out*'.

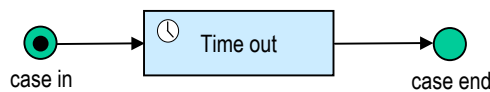


Figure 91 Standard transition 'timeout'

2.6.4 The 'negotiation' transition

The structure of the '*negotiation*' transition is not identical for all contracting processes. Instead it is determined by two factors: (i) the transaction protocol used in the negotiation phase and (ii) the negotiation strategy used in combination with that transaction protocol. In this section we will give examples of the structure of the '*negotiation*' transition for a number of protocol patterns presented before: 'implicit accept', 'binding request' and 'single binding offer'. Before we give the examples, we will discuss correctness criteria for the '*negotiation*' transition.

Correctness criteria

A ‘negotiation’ transition must suffice the following conditions in order to be used in a contracting workflow:

1. The transition has one input place ‘specified’ of which the colour is defined by the object model in Figure 75.
2. The transition has two output places ‘failed’ and ‘committed’ of which the colour is defined by the object model in Figure 75.
3. The transition has one input place for each inbound message type in the negotiation transaction protocol. The colour of the places is defined by the object model in Figure 31.
4. The transition has one output place for each outbound message type in the negotiation transaction protocol. The colour of the places is defined by the object model in Figure 31.
5. Create the workflow net as shown in Figure 92 by deleting the places that represent inbound and outbound messages from the ‘negotiation’ transition and by adding an extra place ‘end’ and an extra OR-join transition ‘end’. This workflow net must be sound.

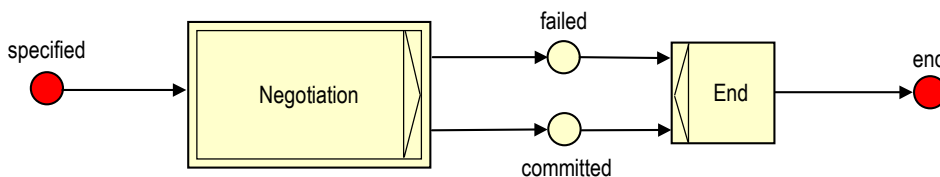


Figure 92 WF-net to prove correctness criterion 5

6. The behaviour of the ‘negotiation’ transition on its interface with service providers (places that model inbound and outbound messages) conforms to the corresponding negotiation transaction protocol. In other words, the ‘negotiation’ transition will never produce a token in a place that models an outbound message that will not be consumed by the protocol workflow. Similarly, a token produced by the protocol workflow in a place that models an inbound message will always be consumed by the ‘negotiation’ transition. The correctness criterion 6 can be proved analytically by constructing a workflow net like the one Figure 93 (if necessary with modifications depending on the structure of the negotiation protocol) and proving its soundness. If the WF-net is not sound, an empirical approach must be followed.

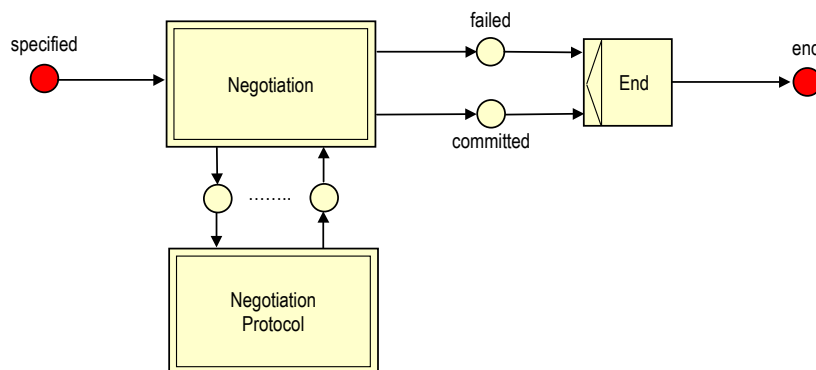


Figure 93 WF-net to prove correctness criterion 6

Rationale

We will now give the rationale behind these correctness criteria. First, criteria 1-4 are used to guarantee that the negotiation processes have a standardised interface (i.e. input places and output places) so that they can be used in the contracting workflow net seamlessly. Criterion 5 is used to prove that the *'negotiation'* transition behaves like an OR-split in the contracting workflow. This property of the *'negotiation'* transition can then be used later to prove that the entire contracting workflow is sound. As we will see, even when a *'negotiation'* transition behaves like an OR-split, it is not always possible to prove the soundness of the WF-net in Figure 92. A reason for this is for instance the use of a time stamp in preconditions. If this is the case, we can use an empirical approach where we consider the *'negotiation'* transition as a black box and observe its behaviour as we feed it with all possible responses of service providers in the transaction protocol. Finally, because each *'negotiation'* transition is designed for a specific negotiation protocol and negotiation strategy, we want to prove that the *'negotiation'* transition behaves according to the agreed negotiation protocol.

Example for negotiation pattern 'implicit accept'

An example of the structure of the *'negotiation'* transition in case of the 'implicit accept' negotiation pattern is given in Figure 94. All transitions are based on the standard transitions defined before.

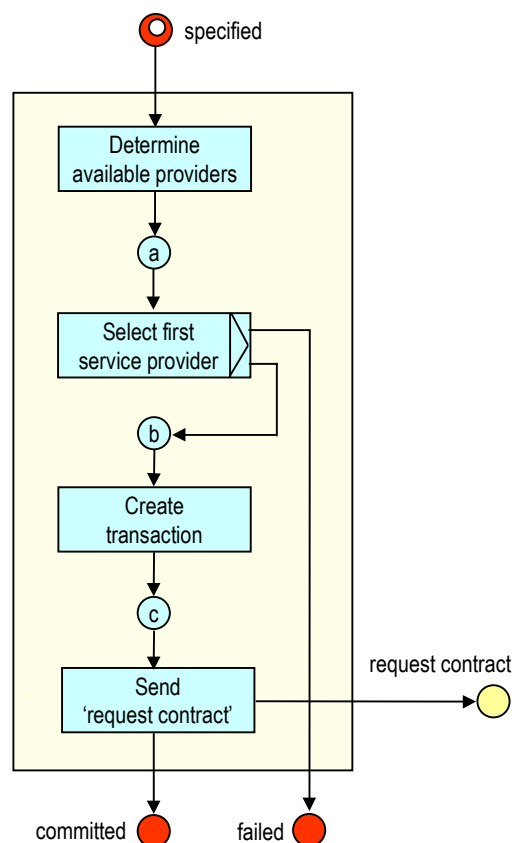


Figure 94 A 'negotiation' transition for the 'implicit accept' pattern

The transition suffices correctness criteria 1-4. In order to prove correctness criterion 5, we construct the workflow net shown in Figure 95 and analyse it with Woflan. The results of the analysis shows that the workflow is sound and the *'negotiation'* transition suffices correctness criterion 5.

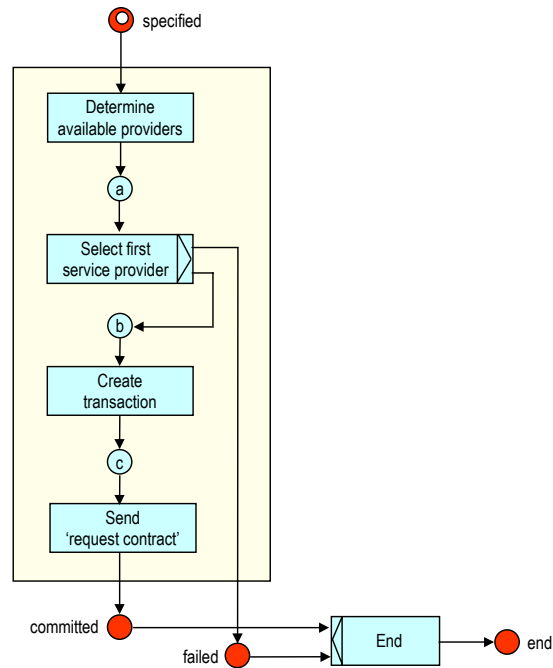


Figure 95 Structure of the WF-net to prove correctness criterion 5

In order to prove correctness criterion 6, we construct the workflow net shown in Figure 96 and analyse it with Woflan. The result of the analysis shows that the workflow is sound and the ‘negotiation’ transition suffices correctness criterion 6.

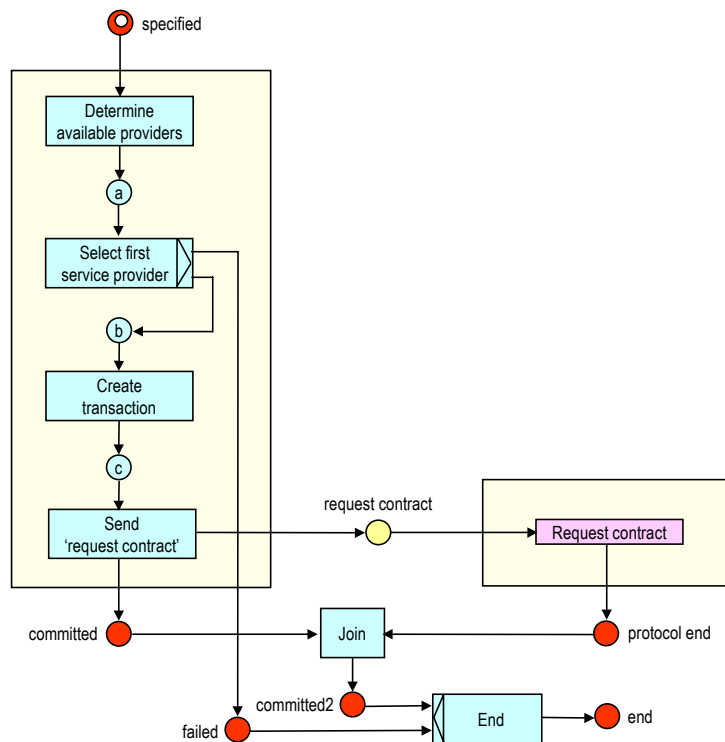


Figure 96 Structure of the WF-net to prove correctness criterion 6

Example for negotiation pattern ‘binding request’

An example of the structure of the ‘negotiation’ transition in case of the ‘binding request’ negotiation pattern is given in Figure 97. The process is based on a sequential approach and starts by sending a ‘request contract’ message to the first available provider on the list. If the service provider responds by sending a ‘reject contract’ message, or if the service provider does not answer within a predefined time interval (defined in the negotiation strategy), a ‘request contract’ message is sent to the next available service provider on the list. The process ends as soon as an ‘accept contract’ message is received or when the list of available providers is exhausted. All transitions are based on the standard transitions presented before.

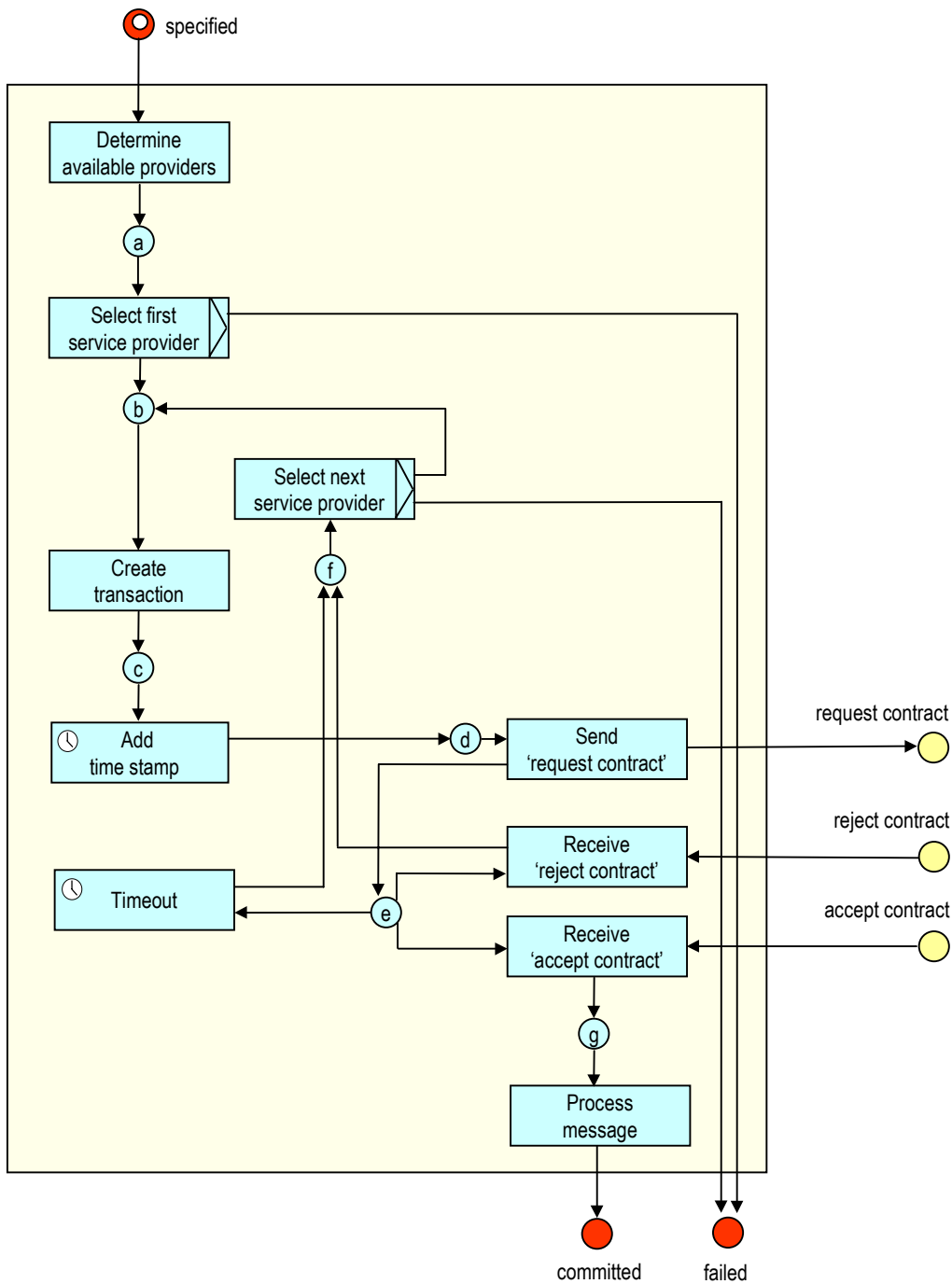


Figure 97 A ‘negotiation’ transition for the ‘binding request’ negotiation pattern

In order to prove correctness criterion 5 for this *'negotiation'* transition, we construct the WF-net shown in Figure 98 and check the soundness property with Woflan. The analysis shows that the WF-net is not sound. The reason for this is the *'timeout'* transition, for which we have defined a precondition that uses the real time clock. Since the analysis techniques do not take into account these preconditions, soundness can not be proven. However, if we delete the *'timeout'* transition, the WF-net is sound. With this result, we can reason that if the timeout period is longer than the maximum response time between the *'request contract'* message and the *'accept contract'* and *'reject contract'* messages, the *'negotiation'* transition still behaves like an OR-split as defined in correctness criterion 5.

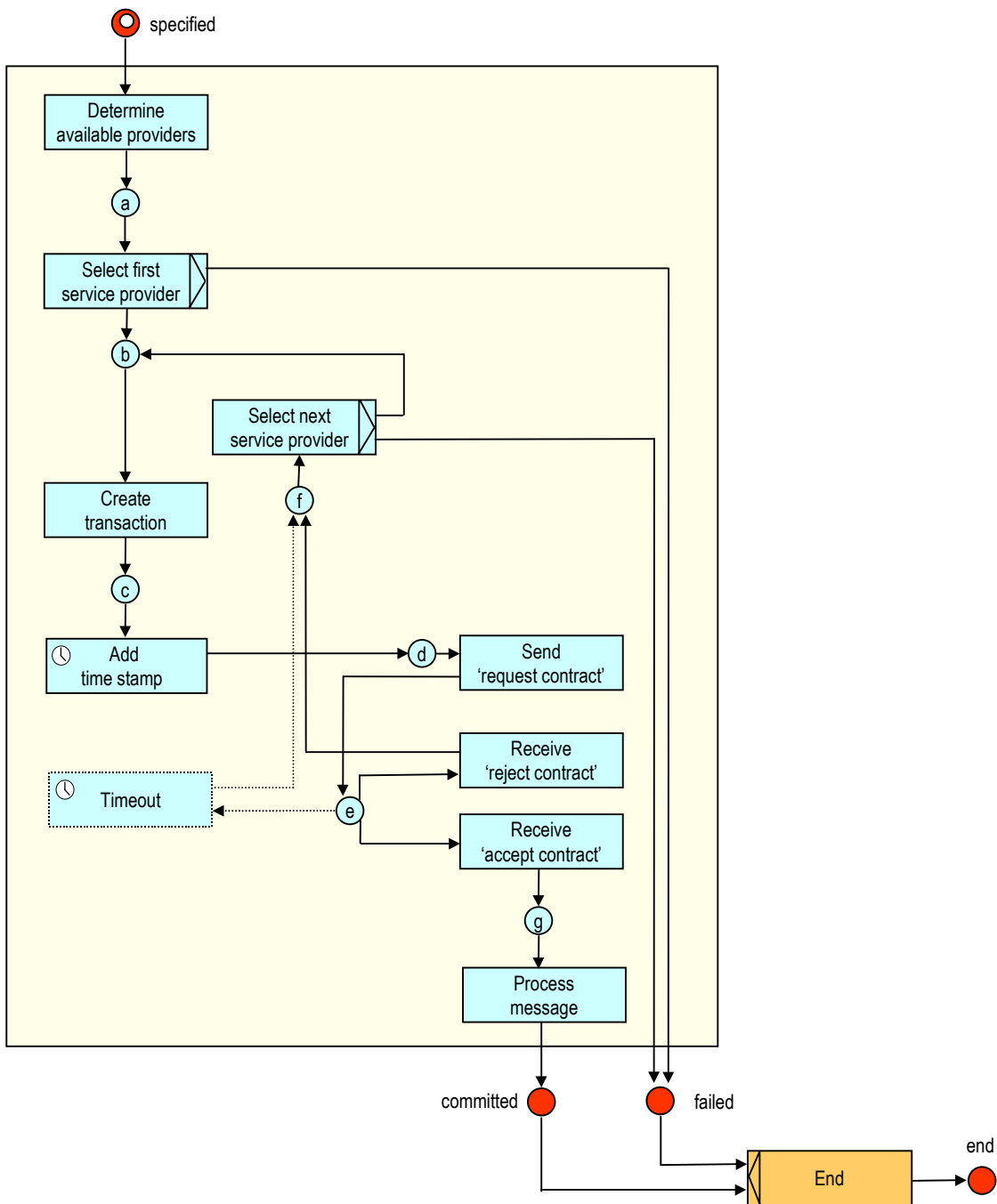


Figure 98 Structure of the WF-net to prove correctness criterion 5

In order to prove correctness criterion 6, we construct the workflow net shown in Figure 99 (transition *'timeout'* excluded) and analyse it with Woflan. The result of the analysis shows that the workflow is sound and the *'negotiation'* transition suffices correctness criterion 6.

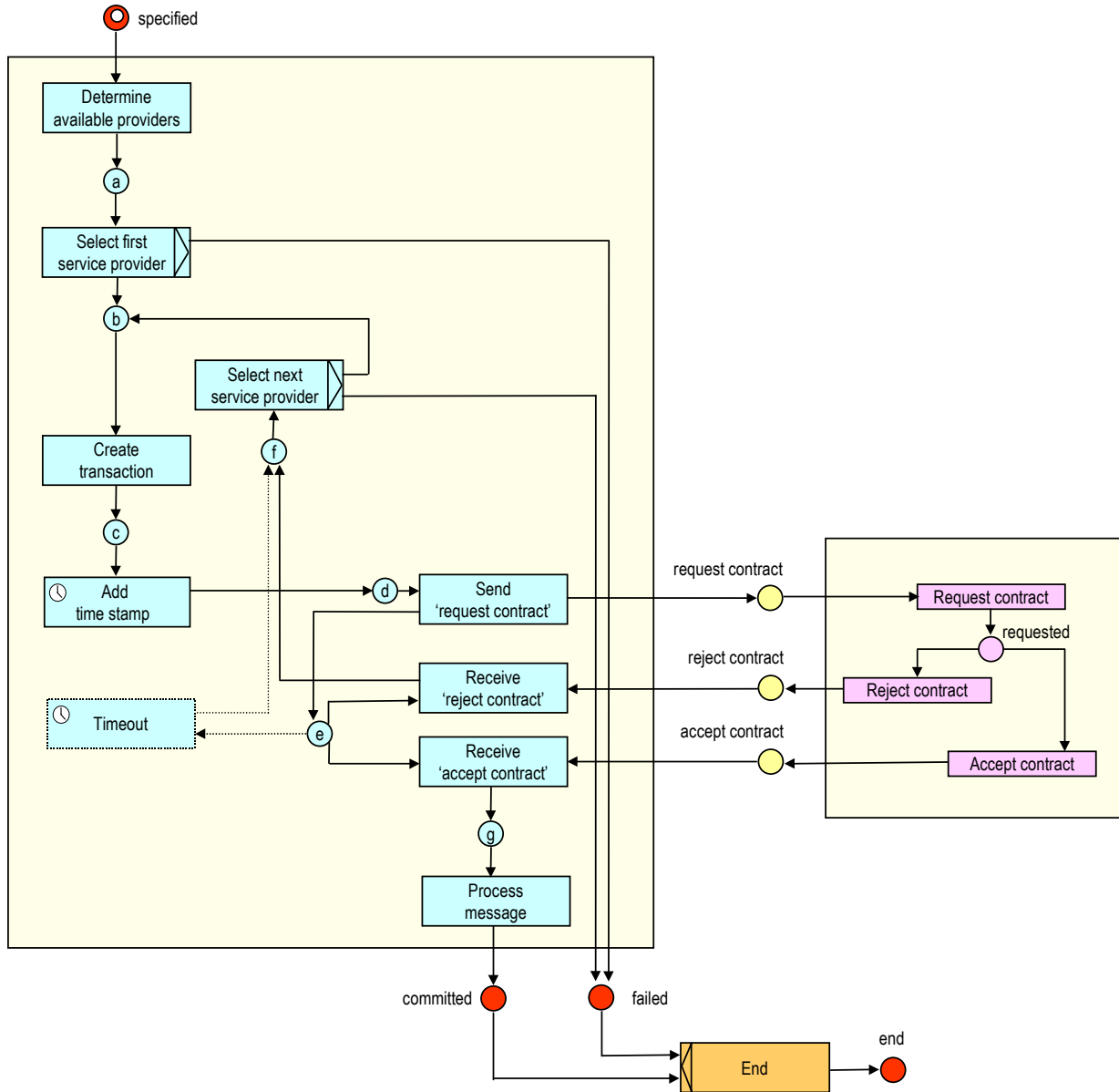


Figure 99 Structure of the WF-net to prove correctness criterion 6

Negotiation procedure for negotiation pattern ‘single binding offer’

When the negotiation process is based on the ‘single binding offer’ pattern, the structure of the *'negotiation'* transition is as given in Figure 100. The process starts by sending a ‘request offer’ message to all available providers. Here after, the responses received from the service providers are evaluated after which the ‘best’ offer is selected and accepted. The other received offers are then rejected. If an offer is accepted, the process ends by producing a token in place *'committed'*. If no offer is accepted (or when no offer is received), the process ends by producing a token in place *'failed'*. All transitions in Figure 100 are based on standard transitions, except transitions ‘choose response’ and ‘end’.

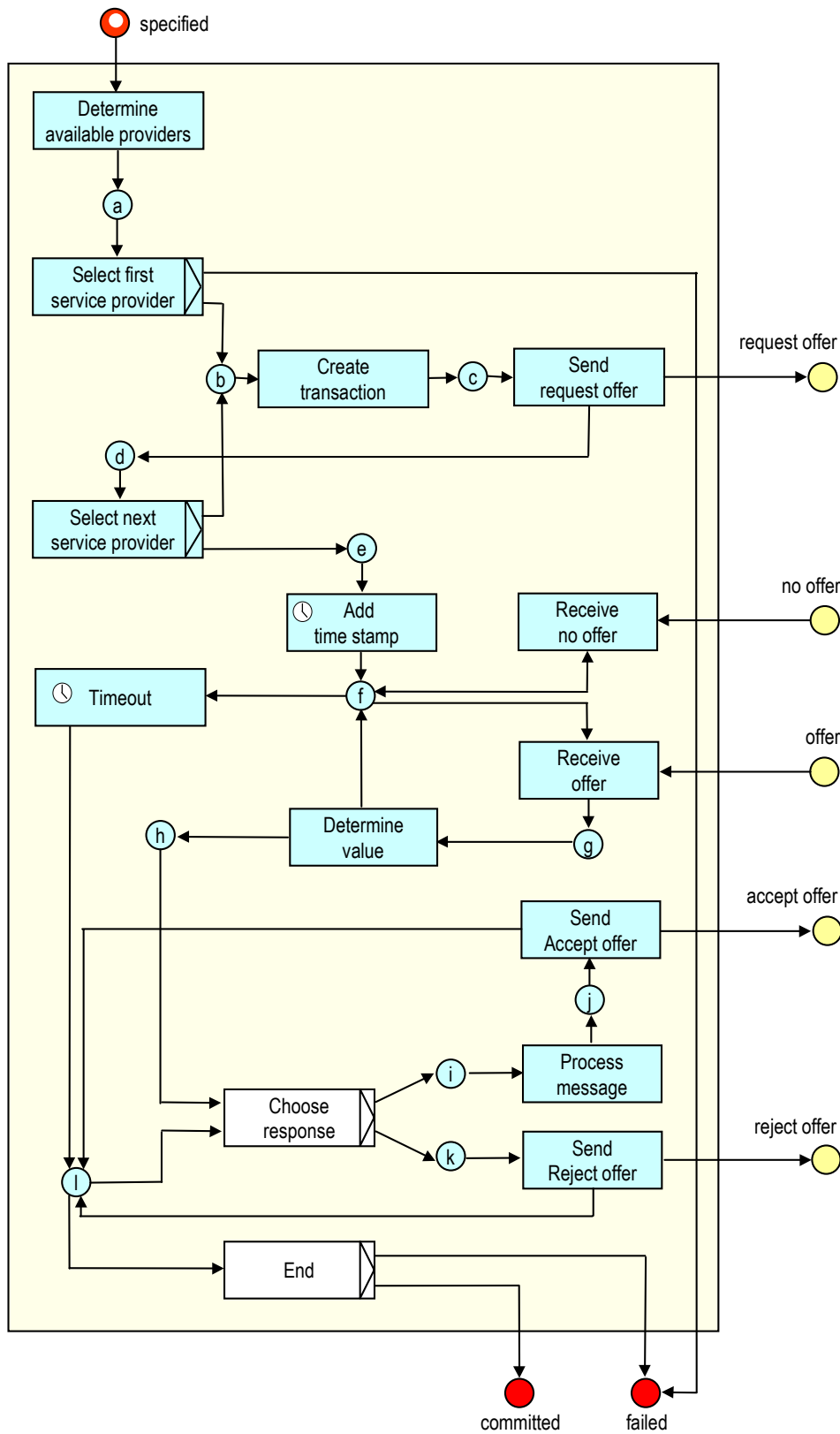


Figure 100 A 'negotiation' transition for the 'single binding offer' negotiation pattern

The ‘*negotiation*’ transition suffices correctness criteria 1-4. However, we can not prove the correctness criteria 5 and 6 analytically because the workflow nets for which we must prove the soundness property contains a number of transitions with a precondition in which a time stamp or a counter is used. It is therefore necessary to prove the characteristics in an empirical way.

2.6.5 The ‘*execution*’ transition

The structure of the ‘*execution*’ transition depends on (i) the execution protocol and (ii) the execution strategy used in combination with that execution protocol. In this section we will give an example of the structure of the ‘*execution*’ transition. First, we will discuss correctness criteria for ‘*execution*’ transitions.

Correctness criteria

An ‘*execution*’ transition must suffice the following conditions in order to be used in a contracting workflow:

1. The transition has one input place ‘*committed*’ of which the colour is defined by the object model in Figure 75.
2. The transition has two output places ‘*aborted*’ and ‘*executed*’ of which the colour is defined by the object model in Figure 75.
3. The transition has one input place for each inbound message type in the execution transaction protocol. The colour of the places is defined by the object model in Figure 31.
4. The transition has one output place for each outbound message type in the execution transaction protocol. The colour of the places is defined by the object model in Figure 31.
5. Create the workflow net as shown in Figure 101 by deleting the places that represent inbound and outbound messages from the ‘*execution*’ transition and by adding an extra place ‘*end*’ and an extra OR-join transition ‘*end*’. This workflow net must be sound.

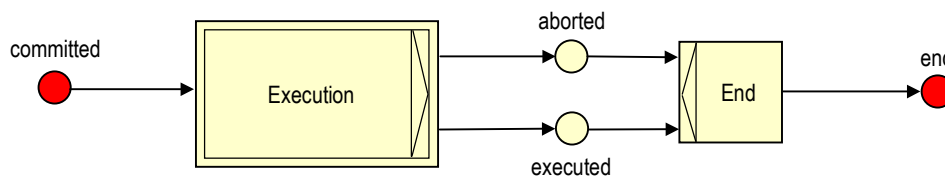


Figure 101 WF-net to prove correctness criterion 5

6. The behaviour of the ‘*execution*’ transition on its interface with service providers (places that model inbound and outbound messages) conforms to the corresponding execution transaction protocol. In other words, the ‘*execution*’ transition will never produce a token in a place that models an outbound message that will not be consumed by the execution protocol workflow. Similarly, a token produced by the execution protocol workflow in a place that models an inbound message will always be consumed by the ‘*execution*’ transition. The correctness criterion 6 can be proved analytically by constructing a workflow net like the one in Figure 102 (if necessary with modifications depending on the structure of the negotiation protocol) and proving its soundness. If the WF-net is not sound, an empirical approach must be followed.

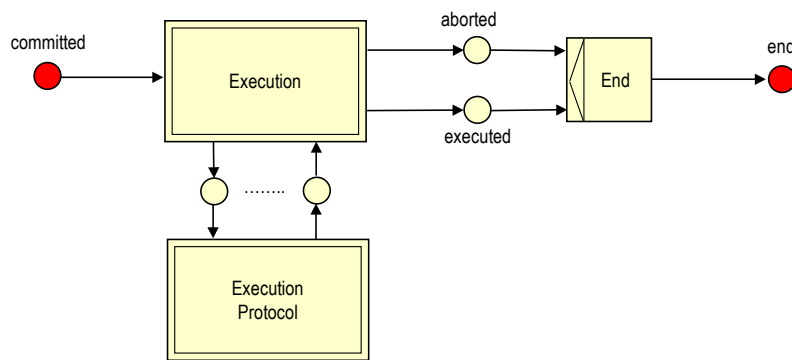


Figure 102 WF-net to prove correctness criterion 6

Example

When the ‘single phase execution’ pattern is used for the execution phase, the structure of the execution transition is as shown in Figure 103.

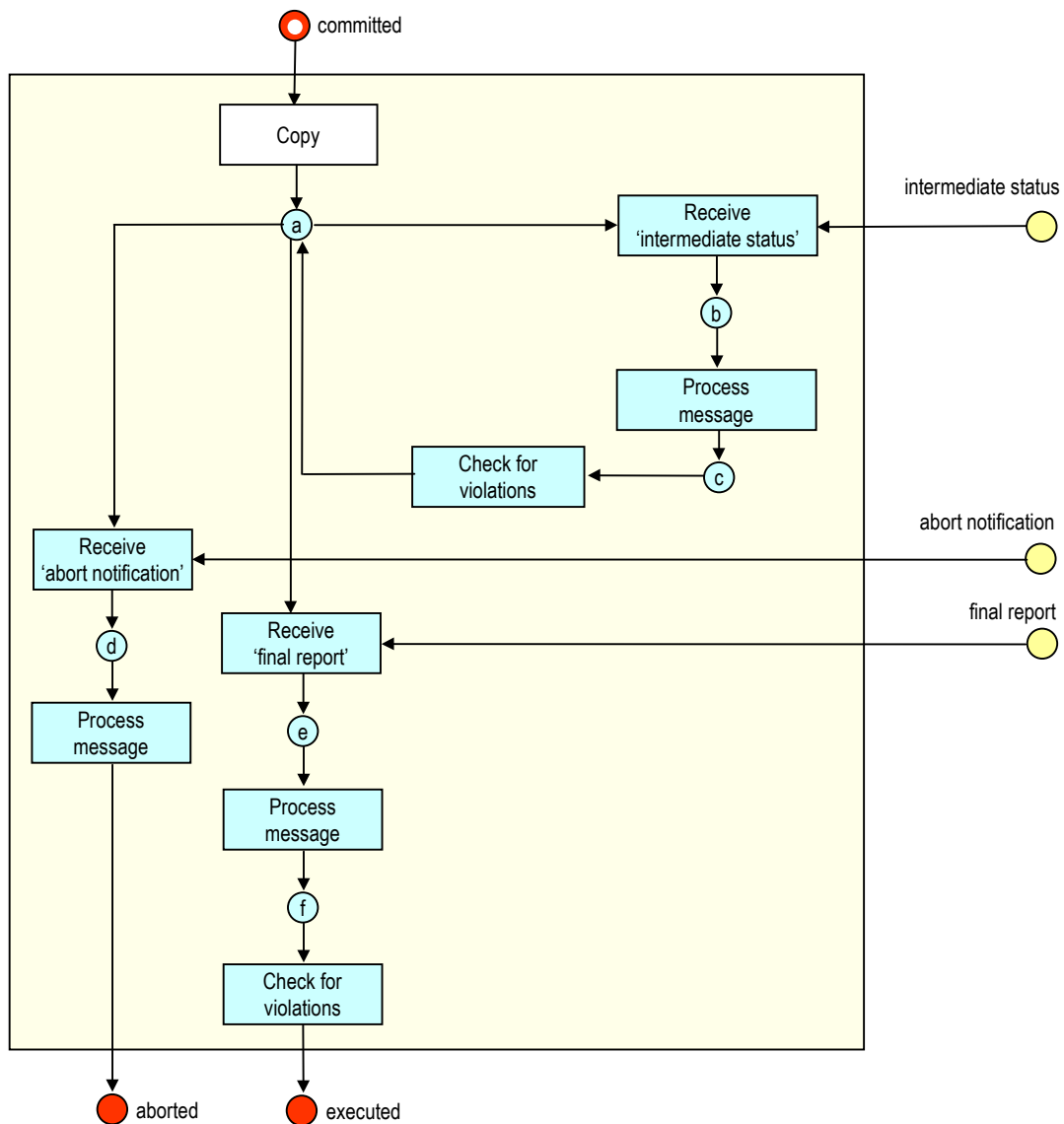


Figure 103 An ‘execution’ transition for the ‘two phase execution’ execution pattern

All transitions, except transition ‘copy’ are standard transitions. The ‘copy’ transition produces all tokens consumed from place ‘committed’ in place ‘a’ without changing the colour of the token. In order to prove correctness criterion 5 for this ‘execution’ transition, we construct the WF-net shown in Figure 104 and check the soundness property with Woflan. The analysis shows that the WF-net is sound.

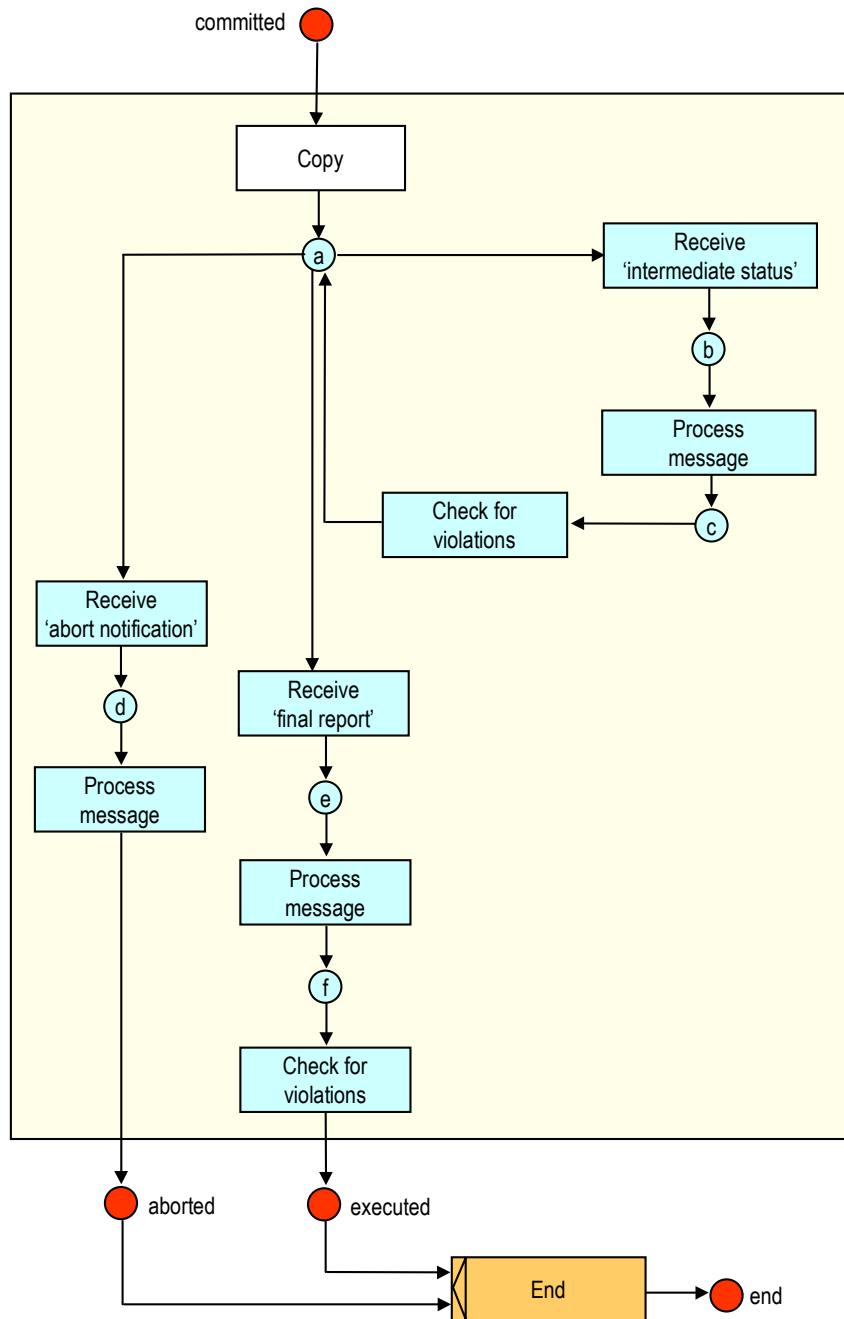


Figure 104 Structure of the WF-net to prove correctness criterion 5

In order to prove correctness criterion 6 of this ‘execution’ transition, we construct the WF-net shown in Figure 105 and check the soundness property with Woflan. The analysis shows that the WF-net is *not* sound. The reason for this is the possibility that a token is produced in place ‘intermediate status’ where after a token is produced in place ‘abort notification’ or ‘final report’ immediately. If the ‘execution’ transition does not consume the token in place ‘interme-

diate status' before the token in place *'abort notification*' or *'final report*' is produced, it is possible that either transition *'receive abort notification*' or *'receive final report*' fires. If this situation occurs, the token in place *'intermediate status*' will never be consumed. However, if we make an extra assumption that inbound messages are processed in the order in which they are created, the described anomaly will not occur.

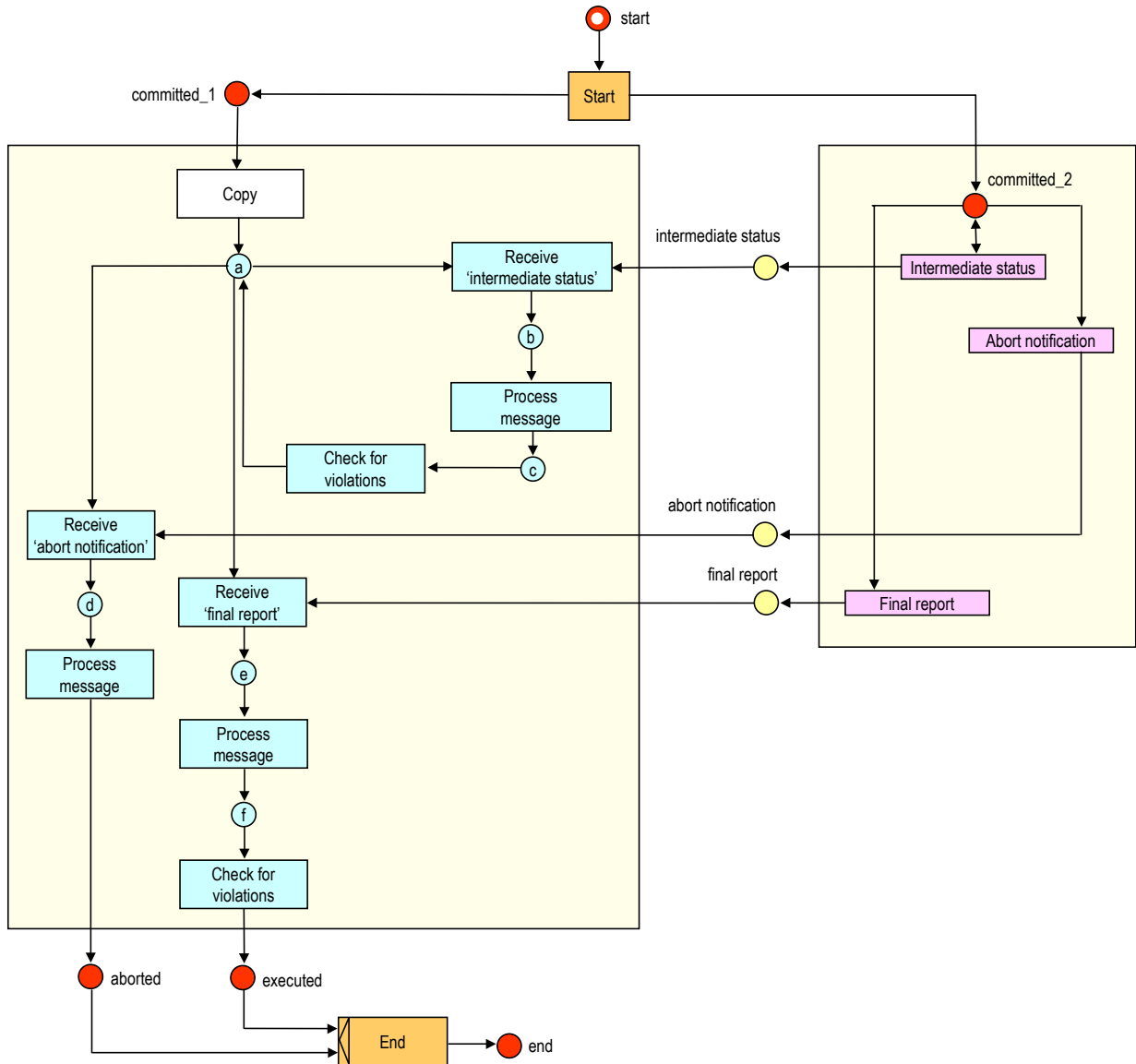


Figure 105 Structure of the WF-net which is checked for soundness

2.6.6 The ‘acceptance’ transition

The structure of the ‘*acceptance*’ transition depends on (i) the acceptance protocol and (ii) the acceptance strategy used in combination with that acceptance protocol. In this section we will give an example of the structure of the ‘*acceptance*’ transition. First, we will discuss correctness criteria for ‘*acceptance*’ transitions.

Correctness criteria

An ‘*acceptance*’ transition must suffice the following conditions in order to be used in a contracting workflow:

1. The transition has one input place ‘*executed*’ of which the colour is defined by the object model in Figure 75.
2. The transition has one output place ‘*completed*’ of which the colour is defined by the object model in Figure 75.
3. The transition has one input place for each inbound message type in the acceptance transaction protocol. The colour of the places is defined by the object model in Figure 31.
4. The transition has one output place for each outbound message type in the acceptance transaction protocol. The colour of the places is defined by the object model in Figure 31.
5. Create a workflow net by deleting all places representing inbound and outbound messages from the ‘*acceptance*’ transition. The workflow must be sound.
6. The behaviour of the ‘*acceptance*’ transition on its interface with service providers (places that model inbound and outbound messages) conforms to the corresponding acceptance transaction protocol. In other words, the ‘*acceptance*’ transition will never produce a token in a place that models an outbound message that will not be consumed by the acceptance protocol workflow. Similarly, a token produced by the acceptance protocol workflow in a place that models an inbound message will always be consumed by the ‘*acceptance*’ transition. The correctness criterion 6 can be proved analytically by constructing a workflow net like the one in Figure 106 (if necessary with modifications depending on the structure of the negotiation protocol) and proving its soundness. If the WF-net is not sound, an empirical approach must be followed.

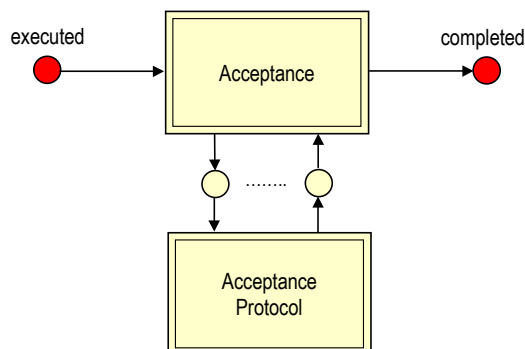


Figure 106 WF-net to prove correctness criterion 6

Example

When the ‘explicit accept’ pattern is used for the acceptance phase, the structure of the acceptance transition is as shown in Figure 107. All transitions, except transition ‘*evaluate result*’, are standard transitions.

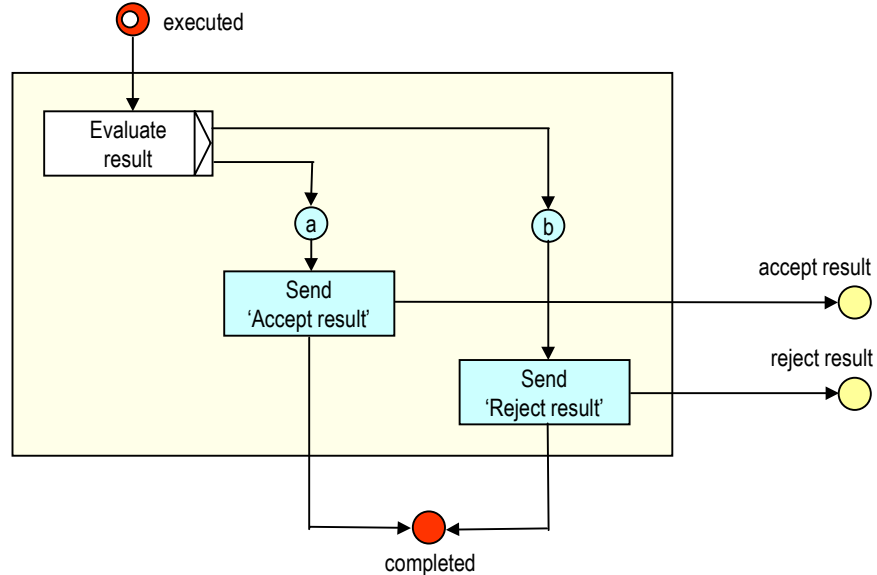


Figure 107 An ‘acceptance’ transition for the ‘explicit accept’ acceptance pattern

- **Firing rules for transition: ‘evaluate result’**

When the transition fires, it consumes a case token from place ‘*executed*’. When the token is consumed, the following attributes are read from the case token:

- the ‘*data*’ attribute of the corresponding ‘CANDIDATE SERVICE’ entity (here after referred to as service data).
- the ‘VIOLATION’ entities related to the ‘CANDIDATE SERVICE’ entity (here after referred to as violations).

The service data and violations are used to decide whether the result is accepted or rejected. In the former case, the case token is produced in place ‘*a*’. Otherwise, the case token is produced in place ‘*b*’.

In order to prove correctness criterion 5 for this ‘*acceptance*’ transition, we construct a WF-net by deleting the places ‘*accept result*’ and ‘*reject result*’ from the ‘*acceptance*’ transition in Figure 107. Here after we check the soundness property of this WF-net with Woflan. The analysis shows that the WF-net is sound.

In order to prove correctness criterion 6 for this ‘*acceptance*’ transition, we construct the WF-net shown in Figure 108 and check the soundness property with Woflan. The analysis shows that the WF-net is sound.

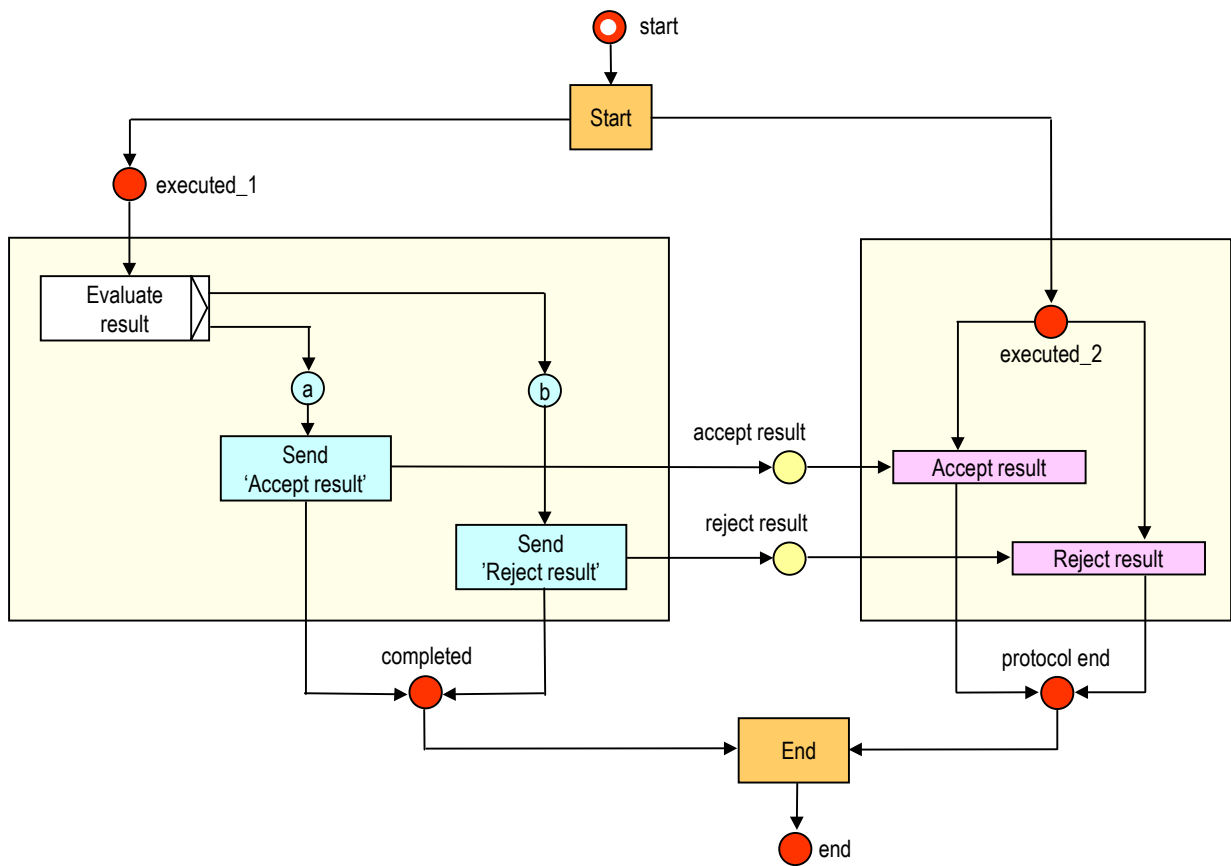


Figure 108 WF-net to prove correctness criterion 6

2.6.7 Composing the contracting workflow

This section discusses the rules according to which the contracting workflow is composed from the building blocks defined before and from the contracting requirements. As we have seen before, the contracting workflow must be a sound WF-net if we omit the places via which message tokens are exchanged with workflows of service providers. Therefore, the starting point for each contracting workflow is a source place ‘start’ and a sink place ‘end’.



Figure 109 Starting point for the contracting workflow

The next step in creating the contracting workflow is to add the transitions and places that model the contracting process for each individual candidate service type. Section 2.6.2 discussed this structure. If the successful completion of the negotiation phase of the candidate service type is *not* part of the trigger of another candidate service type, the following structure is used to represent the candidate service type in the contracting workflow.

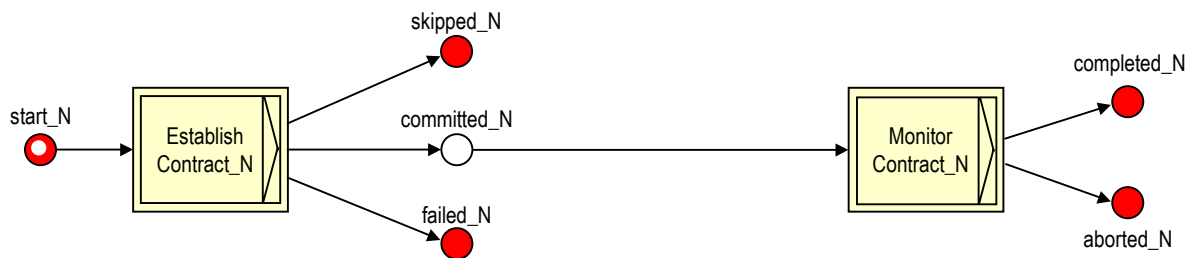


Figure 110 Structure of the basic building block 'contract service' of the contracting workflow (1)

If on the other hand the successful completion of the negotiation phase of the candidate service type *is* part of the trigger of another candidate service type, the following structure is used to represent the candidate service type in the contracting workflow. The extra AND-split is necessary to produce a token in place 'committed_N'.

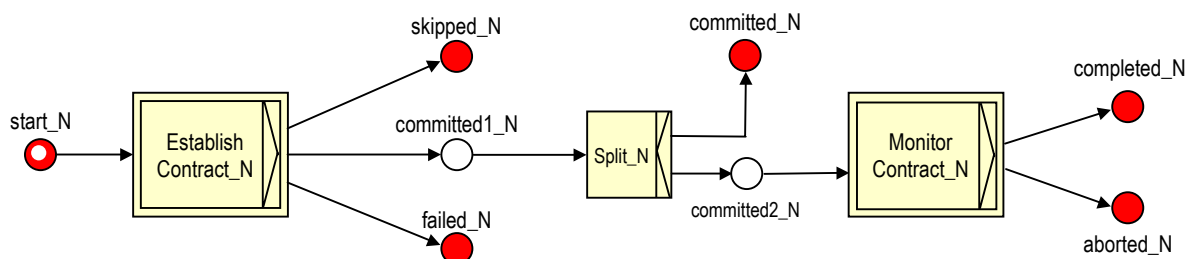


Figure 111 Structure of the basic building block 'contract service' of the contracting workflow (2)

Finally, if the execution phase does not start automatically when the negotiation phase ends with a commitment, but requires a trigger, the following structure is used to represent the candidate service type in the contracting workflow.

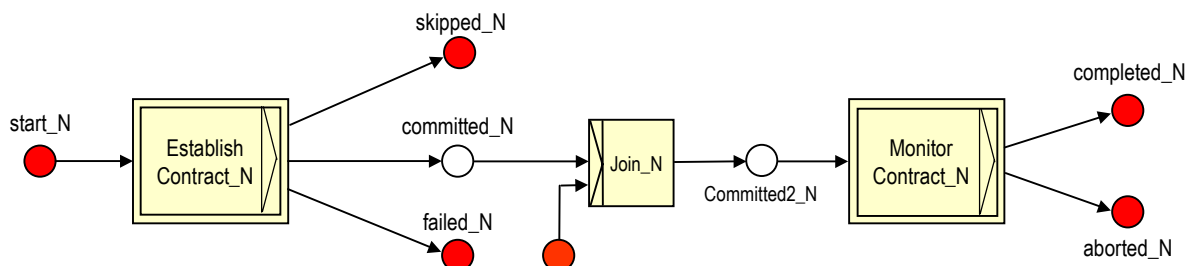


Figure 112 Structure of the basic building block 'contract service' of the contracting workflow (3)

At this point, we have a contracting workflow where transitions and places belonging to different candidate service types are not connected. One reason why these connections should exist is to represent the trigger defined for each candidate service type, modelled by the 'trigger' attribute in the 'CANDIDATE SERVICE TYPE' entity. The trigger of a candidate service type defines the conditions under which the service contracting process can start for a specific candidate service of that type. We will now show how to represent these triggers in the contracting workflow.

Trigger: none

The contracting process of a candidate service type for which no trigger is defined starts immediately when the entire service contracting process is started. The transition that implements this triggering mechanism is given in Figure 113 and consists of a trigger transition 'T0'.

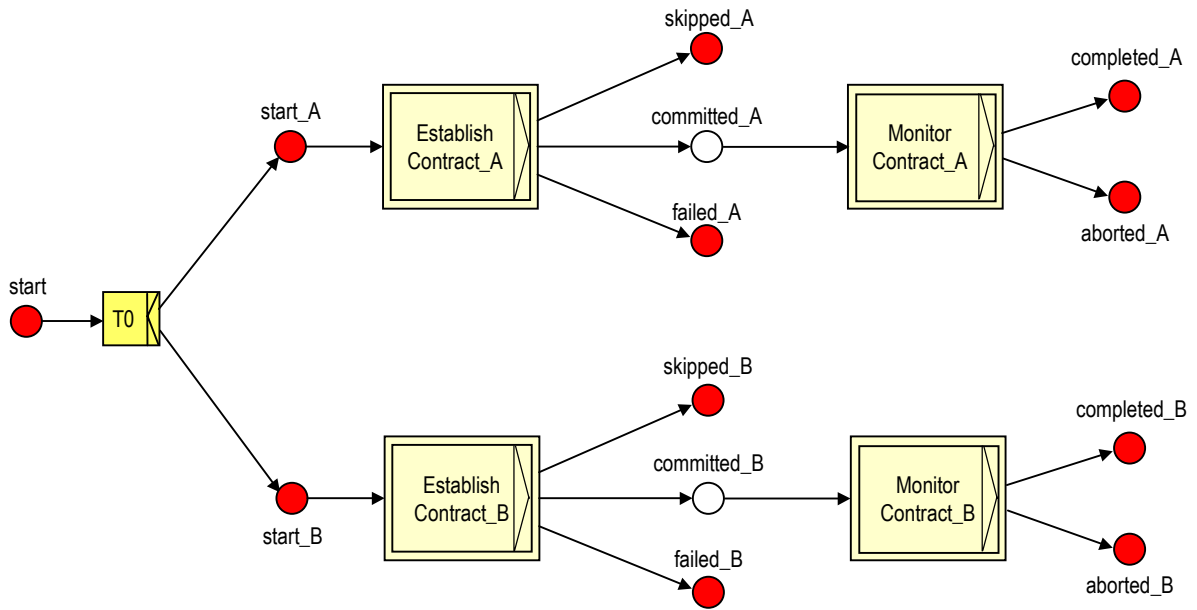


Figure 113 A and B start immediately (triggering mechanism)

Trigger: successful negotiation

A typical type of triggering is when the contracting for a candidate service B is started when a candidate service A has been contracted. Or, in the grammar presented in Figure 68, the trigger for candidate service B is:

B : NEGOTIATION AFTER A = COMMITTED

The transition ‘T1’ that implements the value of the ‘trigger’ attribute of the candidate service type B is given in Figure 114. A typical example of a situation in which this kind of triggering is used is when the ‘constraints’ attribute or the ‘specification rules’ attribute of the candidate service type B contains a service attribute of candidate service type A that becomes known to the service client only when the service contract is established.

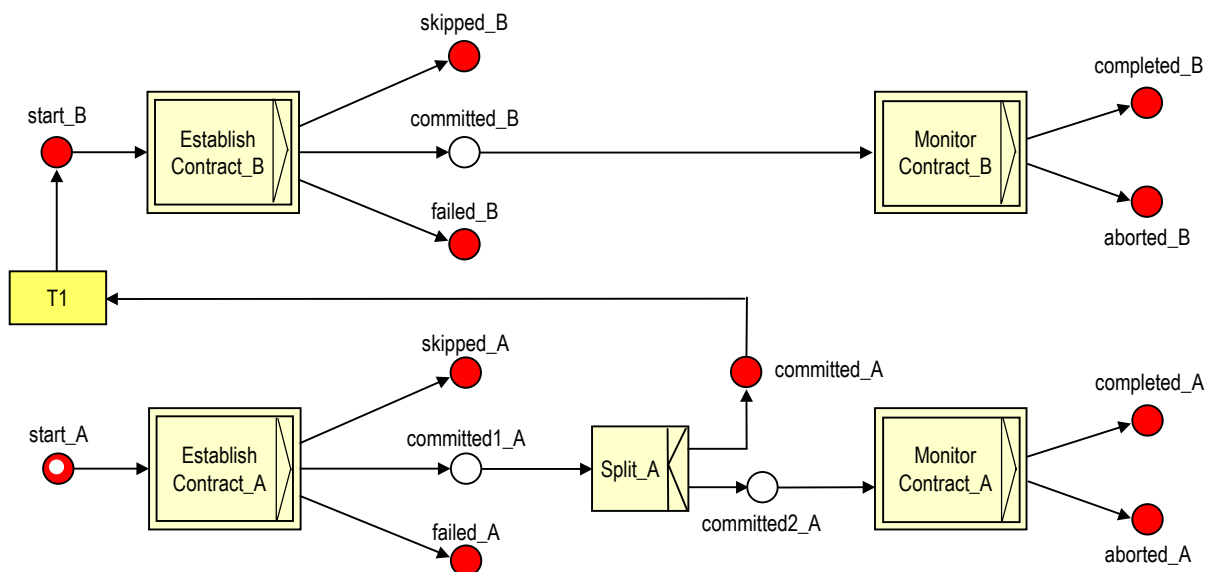


Figure 114 B starts when a contract for A is established (triggering mechanism)

Trigger: failed negotiation

In this type of relationship, candidate service B is an alternative for candidate service A. This means that the contracting process for candidate service B is started only when the contracting process for candidate service A failed. Or in the grammar presented before, the trigger for candidate service B is:

B : NEGOTIATION AFTER A = FAILED

The transition ‘T1’ that implements the ‘*trigger*’ attribute of the candidate service B is given in Figure 115.

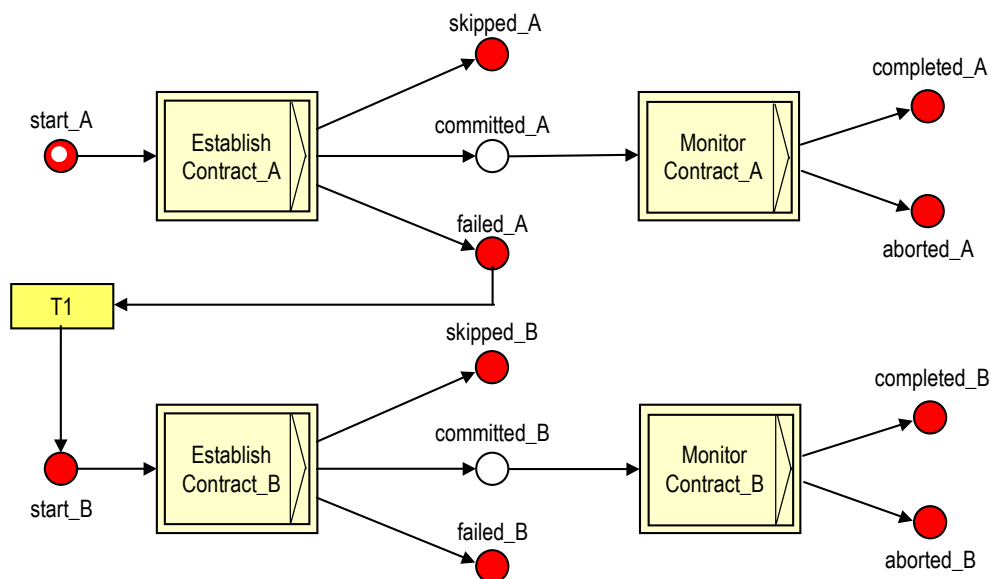


Figure 115 B starts when a contract for A could not be established (triggering mechanism)

Trigger: completion

Another typical example of triggering is when the contracting of a candidate service B starts when a candidate service A has been executed completely. Or, in the grammar presented before, the trigger for candidate service B is:

B : NEGOTIATION AFTER A = COMPLETED

The transition ‘T1’ that implements the ‘*trigger*’ attribute of the candidate service type B is given in Figure 116. A typical example of a situation in which this kind of triggering is used is when the ‘*constraints*’ attribute or the ‘*specification rules*’ attribute of the candidate service type B contains a service attribute of candidate service type A that becomes known to the service client only when the service provider informs the service client of the end-state of the execution phase. The triggering mechanism can also be used when the candidate services A and B must be executed consecutively and the completion time of candidate service A is not known in advance. The service client has to wait for a signal from the service provider before he starts contracting candidate service B.

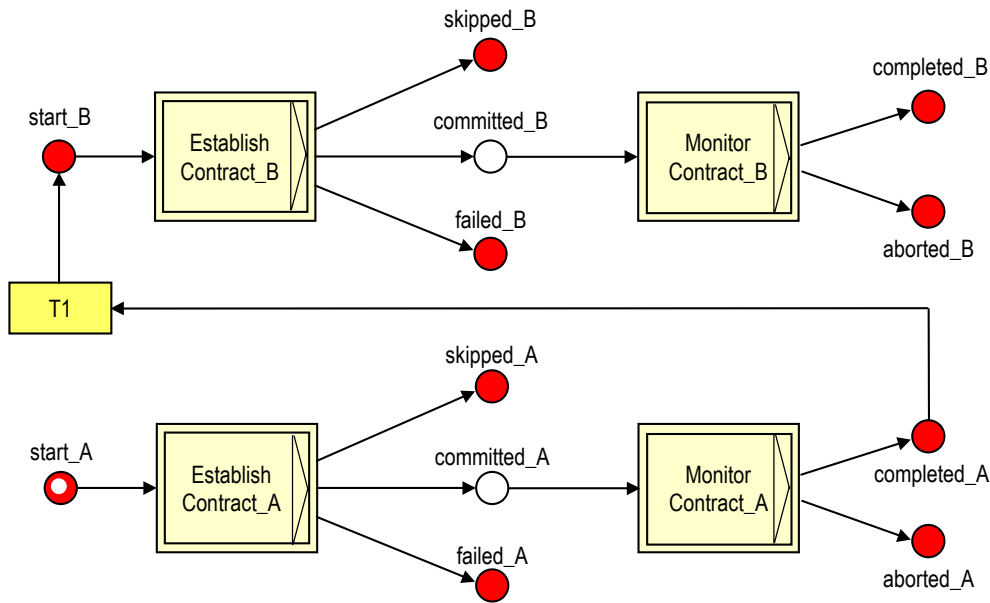


Figure 116 B starts when A is executed (triggering mechanism)

Trigger combination: commitment / completion

A more complex example of triggering is when the negotiation phase for a candidate service B is triggered by the *commitment* for candidate service A, and the execution phase for candidate service B is triggered by the *completion* of candidate service A. Or, in the grammar presented before, the trigger for candidate service B is:

B : NEGOTIATION	AFTER	A = COMMITTED
B : EXECUTION	AFTER	A = COMPLETED

The transitions ‘T1’ and ‘T2’ that implement the ‘trigger’ attribute of the candidate service type B are given in Figure 117.

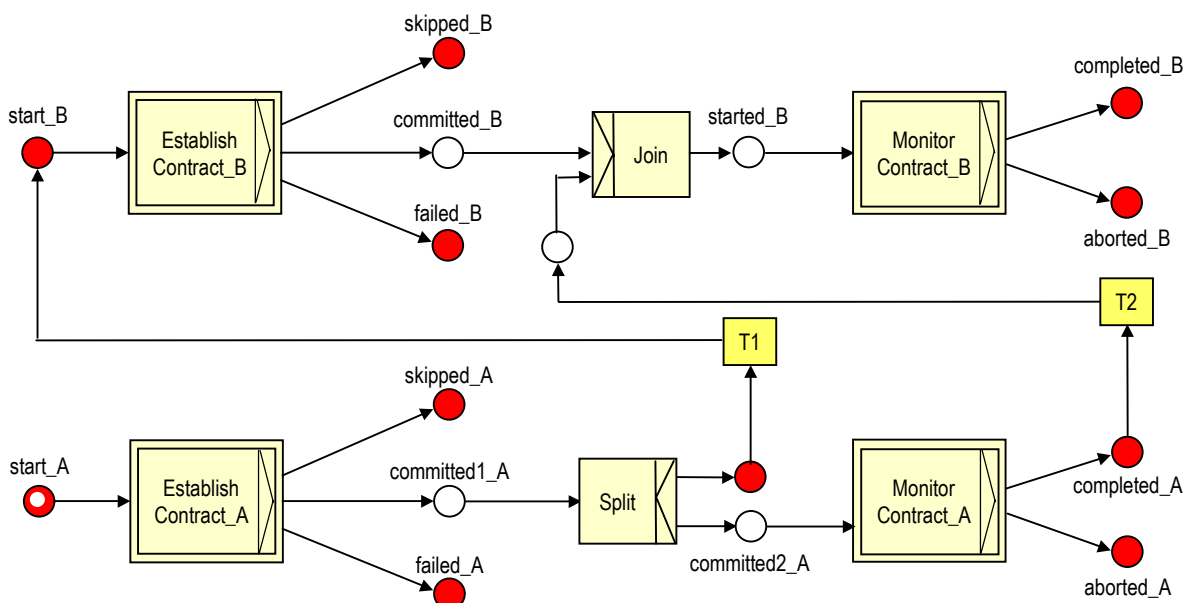


Figure 117 B starts when A is committed, B executes when A is completed (triggering mechanism)

Completion of the partial contracting workflow

At this point, we have created a *partial* contracting workflow in which the place ‘end’ is not connected to any transition yet. Furthermore, other places belonging to the basic building blocks defined in Figure 110, Figure 111 and Figure 112 do not have an outgoing connector. Clearly, this partial contracting workflow is not a sound WF-net. Therefore, the last action is to add those transitions and places to the partial contracting workflow that make the resulting contracting workflow a sound WF-net. This can be done by first constructing the partial contracting workflow and analysing the distribution of tokens in possible end-states. There after, new transitions and places are added in such a way that in each end-state the tokens that define the end-state are consumed and one token is produced in place ‘end’. An example of a partial contracting workflow is given in Figure 128. An example of a sound contracting workflow is given in Figure 129.

2.7 Use case ‘business trip’

2.7.1 Introduction

The use case is about a company in which employees have to travel frequently. Each business trip requires two flights to be booked (outbound and inbound). If the employee is not able to travel on one day, a hotel reservation has to be made. Finally, if the employee wants to, a rental car must be made available at the airport of arrival. The service providers (airline carriers, hotels and car rental organisations) offer a business-to-business electronic commerce interface by which their services can be contracted. The company has an Intranet web-application in which employees can enter the details of their business trip after which a service contracting process is started. The rest of this section consists of the following parts. First, we will define the case type (2.7.2), the service types (2.7.3-2.7.6) and the service providers (2.7.7). There after, we define the contracting requirements and the contracting workflow (2.7.8)

2.7.2 Case type ‘Trip’

The structure of the case data, entered via the intranet application, is defined by the hierarchic data model shown in the left part of Figure 118.

<i>name</i>	<i>format</i>	<i>example</i>
TRIP		
City of departure	an..35	'Amsterdam'
City of destination	an..35	'Seattle'
Latest date of arrival	n8	'25-09-2001'
Latest time of arrival	n4	'18:00'
Earliest date of departure	n8	'28-09-2001'
Earliest time of departure	n4	'10:00'
Rental car yes/no	a1 (1)	'Y'
EMPLOYEE		
Name	an..35	'van Dijk'
Initials	an..6	'A.'
Mr / mrs	a1	'M'
(1) Y	Yes	
N	No	

Figure 118 Case data structure in the ‘business trip’ use case

2.7.3 Service type ‘Book flight’

Booking a seat on a scheduled flight is offered by a number of airline carriers via the service type ‘book flight’. The transaction protocol is based on the ‘multiple non-binding offers’ negotiation pattern and no messages are exchanged in the execution and acceptance phase. The service client sends his travel preferences (departure city/date/time, arrival city/date/time) to an airline carrier who will respond by sending information about one or more flights that match the preferences and for which seats are still available. The service client responds either by booking a seat on one of the possible flights, or by cancelling the negotiation. If the service client books a flight, the airline carrier responds by sending a confirmation or a rejection (if the available seats have been booked in the mean time). In the latter case, the service client can book a seat on another possible flight, if one is still available.

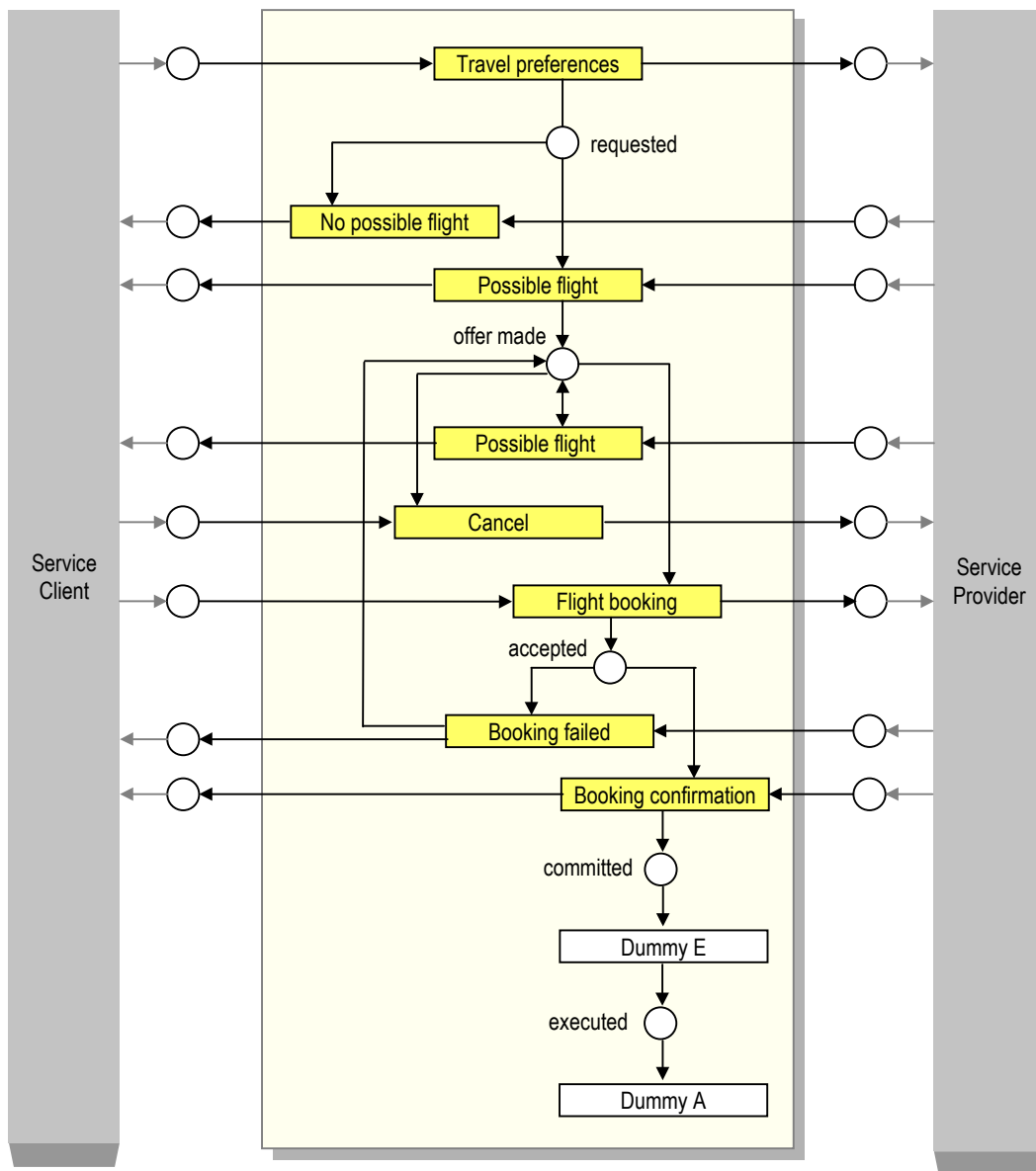


Figure 119 Transaction protocol for the ‘book flight’ service type

The data model of the service data, and the data model of the message types in the business transaction are defined in Figure 120.

<i>Travel preferences</i> ◆ <i>Possible flight</i> ◆ <i>No possible flight</i> ◆ <i>Cancel</i> ◆ <i>Flight booking</i> ◆ <i>Booking confirmation</i> ◆ <i>Booking failed</i> ◆			
MESSAGE	1x		
Message identification	an..20	R R R R R R R R	
Message type	an..20	R R R R R R R R	
Transaction identification	an..20	R R R R R R R R	
Sender identification	an..50	R R R R R R R R	
Receiver identification	an..50	R R R R R R R R	
FLIGHT	1x		
Code	an..6	- R - - R R R	
Class	a1	R R - - R R R	(1)
Price	n..6	- R - - - R -	
Currency	an3	O R - - - R -	(2)
Booking reference	an..10	- - - - - R -	
DEPARTURE	1x		
Airport	an3	O R - - R R -	(3)
City	an..35	R R - - - R -	
Earliest date	n8	R - - - - - -	
Earliest time	n4	O - - - - - -	
Latest date	n8	O - - - - - -	
Latest time	n4	O - - - - - -	
Schedule date	n8	- R - - R R -	
Schedule time	n4	- R - - - R -	
ARRIVAL	1x		
Airport	an3	O R - - R R -	(3)
City	an..35	R R - - - R -	
Earliest date	n8	O - - - - - -	
Earliest time	n4	O - - - - - -	
Latest date	n8	R - - - - - -	
Latest time	n4	O - - - - - -	
Schedule date	n8	- R - - R R -	
Schedule time	n4	- R - - - R -	
CLIENT	1x		
Name	an..50	- - - - R R -	
Initials	an..5	- - - - R R -	
Sex	a1	- - - - R R -	(4)
(1) E	Economy class	(2)	ISO 4217 currency codes
B	Business class		
(3)	IATA airport codes	(4)	M Male
			F Female

Figure 120 Service data and message data for the 'book flight' service type

2.7.4 Service type ‘Cancel flight’

The service type ‘Cancel flight’ is offered by the airline carriers that also offer the ‘Book flight’ service type. It can be used to cancel a flight that has been booked earlier. The transaction protocol is based on the ‘binding request’ pattern and has no message exchange in the execution and acceptance phases.

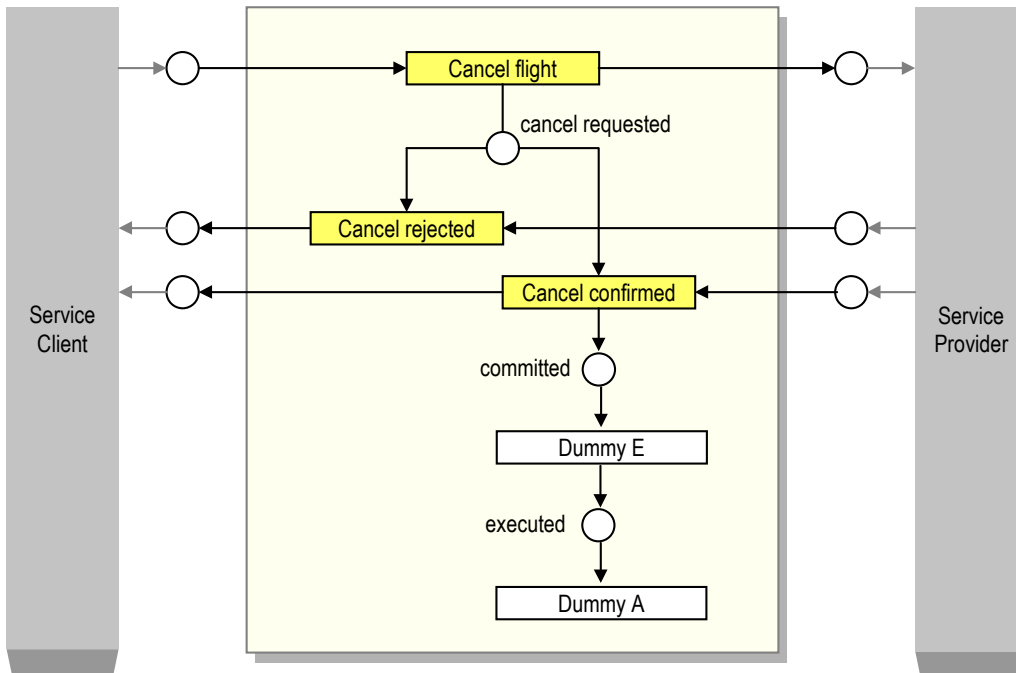


Figure 121 Transaction protocol for the ‘cancel flight’ service type

The data model of the service data, and the data model of the message types in the business transaction protocol is defined in Figure 122.

<i>Cancel flight</i> ◆ <i>Cancel confirmed</i> ◆ <i>Cancel rejected</i> ◆		
MESSAGE	1x	
Message identification	an..20	R R R
Message type	an..20	R R R
Transaction identification	an..20	R R R
Sender identification	an..50	R R R
Receiver identification	an..50	R R R
FLIGHT	1x	
Code	a1	R R R
Booking reference	an..10	R R R

Figure 122 Service data and message data for the ‘cancel flight’ service type

2.7.5 Service type 'Book hotel'

Reservation of a hotel room for one or more days is offered by a number of hotel chains via the 'Book hotel' service type. If a room is available for the specified period, the reservation will be confirmed and a contract is established. If no rooms are available, the client will be notified. After a contract is established, the hotel will send a notification when the client checked in. When the client checks out, the hotel will send a check out notification with a specification of the costs. The transaction protocol is based on the 'binding request' negotiation pattern and on the 'two phase execution' execution pattern. No messages are exchanged in the acceptance phase.

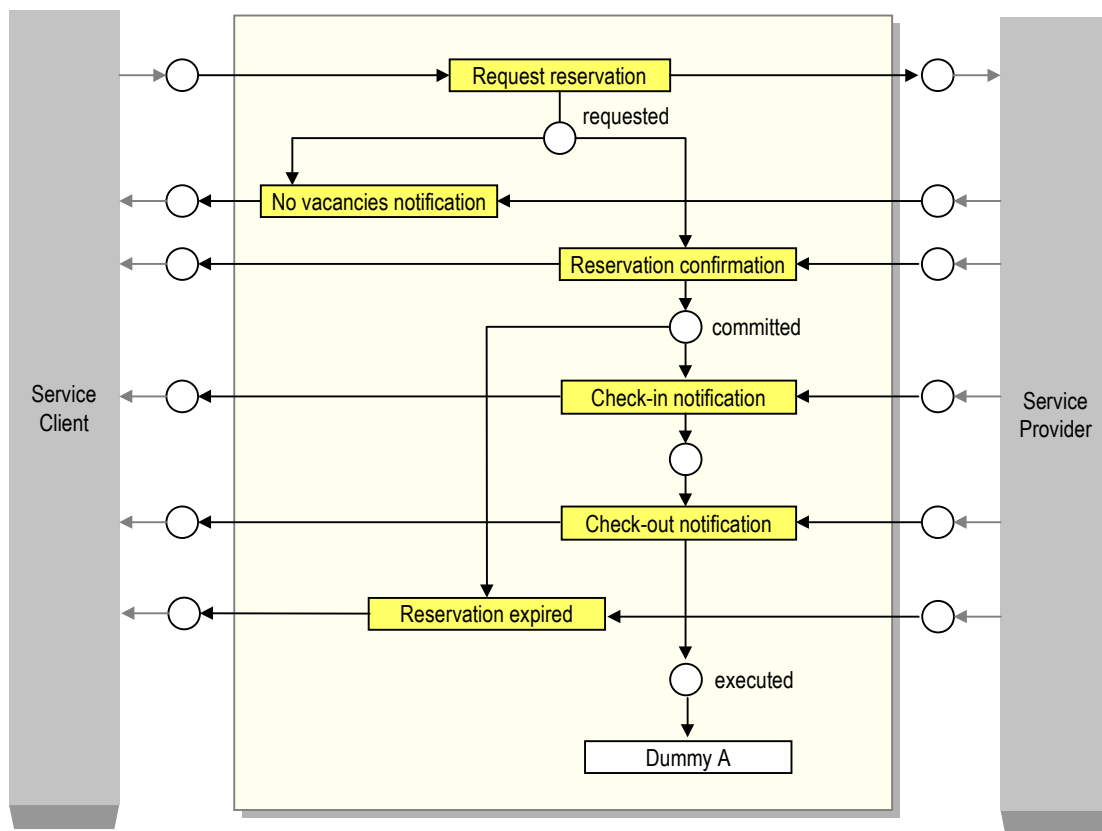


Figure 123 Transaction protocol for the 'book hotel' service type

The data model of the service data, and the data model of the message types in the business transaction is defined in Figure 124.

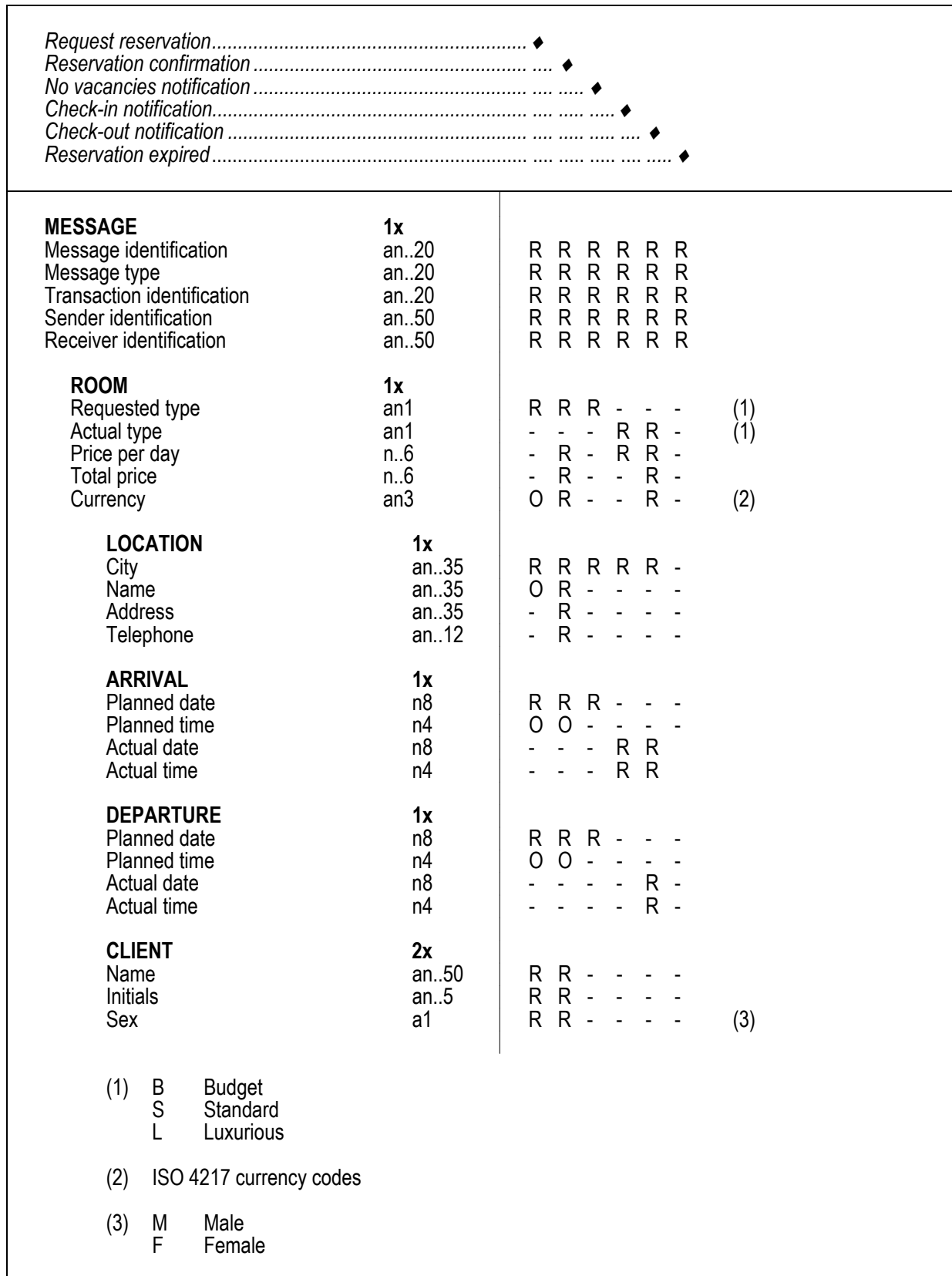


Figure 124 Service data and message data for the 'book hotel' service type

There is one violation type defined for this service type, which occurs when the agreed type of room is not available when the guest checks-in.

[ROOM] (requested type) <> [ROOM] (actual type)

2.7.6 Service type ‘Book rental car’

Reservation of a rental car for one or more days is offered by a number of service providers via the ‘Book rental car’ service type. A service client can request a reservation at a rental car organisation. If a car is available for the specified period, the reservation will be confirmed and a contract is established. If no car is available, the client will be notified. After a contract is established, the rental organisation will send a notification when the client picks up the car. When the client returns the car, the rental organisation will send a notification with a specification of the costs. The transaction protocol is based on the ‘binding request’ negotiation pattern and on the ‘two phase execution’ execution pattern.

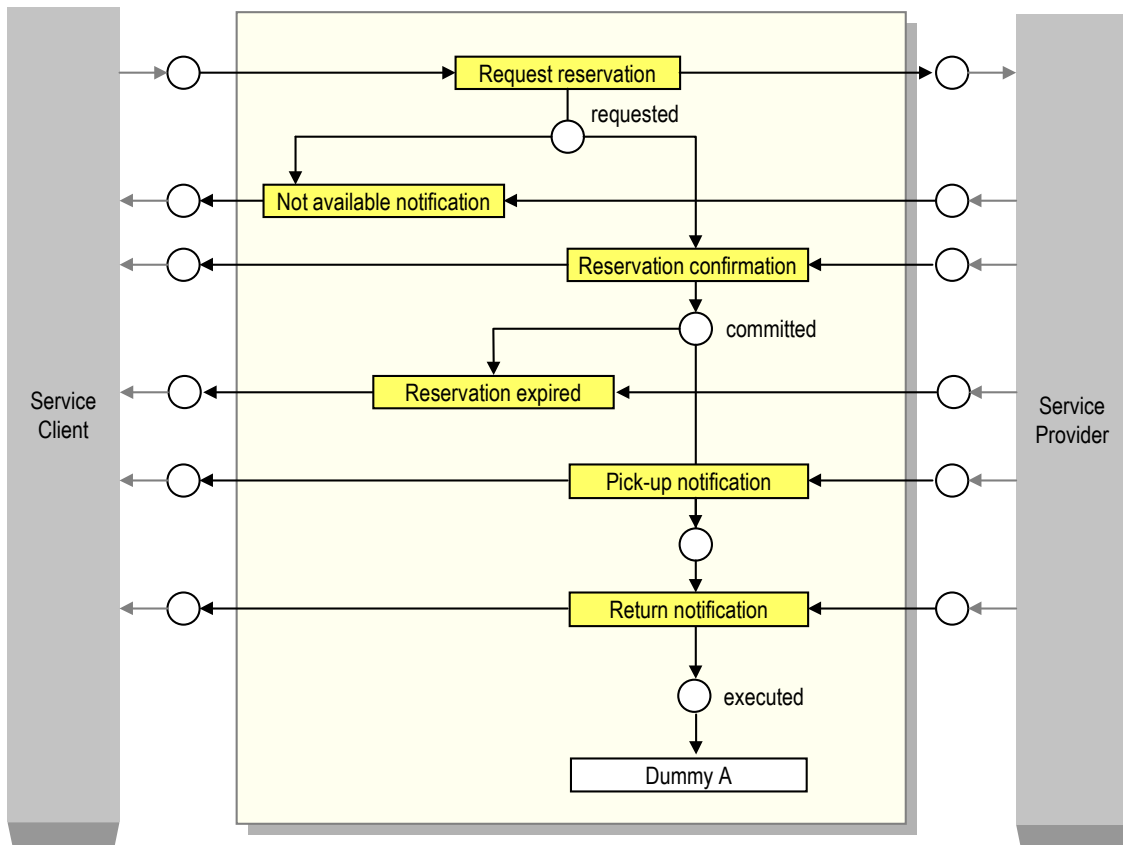


Figure 125 Transaction protocol for the ‘book rental car’ service type

The data model of the service data, and the data model of the message types in the business transaction protocol is defined in Figure 126.

<i>Request reservation.....◆</i> <i>Reservation confirmation.....◆</i> <i>Not available notification.....◆</i> <i>Pick-up notification.....◆</i> <i>Return notification.....◆</i> <i>Reservation expired.....◆</i>						
MESSAGE	1x					
Message identification	an..20	R	R	R	R	R
Message type	an..20	R	R	R	R	R
Transaction identification	an..20	R	R	R	R	R
Sender identification	an..50	R	R	R	R	R
Receiver identification	an..50	R	R	R	R	R
RENTAL CAR	1x					
Requested type	an1	R	R	-	-	-
Actual type	an1	-	-	-	R	-
Price per day	n..6	-	R	-	-	R
Extra costs	n..6	-	-	-	-	R
Total price	n..6	-	R	-	-	R
Currency	an3	-	R	-	-	R
						(1)
						(1)
						(2)
PICK-UP	1x					
Planned date	n8	R	R	-	-	-
Planned time	n4	R	R	-	-	-
Actual date	n8	-	-	-	R	-
Actual time	n4	-	-	-	R	-
Airport code	an3	R	R	-	R	-
Flight number	an..6	R	R	-	-	-
						(3)
RETURN	1x					
Planned date	n8	R	R	-	-	-
Planned time	n4	R	R	-	-	-
Actual date	n8	-	-	-	-	R
Actual time	n4	-	-	-	-	R
Airport code	an3	R	R	-	-	-
Flight number	an..6	R	R	-	-	-
Damage description	an..100	-	-	-	-	O
						(3)
CLIENT	1x					
Name	an..50	R	R	-	-	-
Initials	an..5	R	R	-	-	-
Sex	a1	R	R	-	-	-
						(4)
(1)	C	Compact size				
	M	Medium size				
	F	Full size				
	4	Four wheel drive				
(2)		ISO 4217 currency codes				
(3)		IATA airport codes				
(4)	M	Male				
	F	Female				

Figure 126 Service data and message data for the 'book rental car' service type

There are two violation types defined for this service type, which occur when the agreed type of car is not available when the car is picked up, or when the car is returned with damage.

[RENTAL CAR] (requested type) <> [RENTAL CAR] (actual type)
 [RENTAL CAR] [RETURN] (damage description) <>

2.7.7 Service providers

The four service types are offered by the following service providers.

	Book flight	Cancel flight	Book hotel	Book rental car
Airline 1	Yes	Yes	-	-
Airline 2	Yes	Yes	-	-
Hotel 1	-	-	Yes	-
Hotel 2	-	-	Yes	-
Car rental 1	-	-	-	Yes (*)
Car rental 2	-	-	-	Yes (*)

(*) The two car rental organisations are not located on each airport. This is modelled by constraints on the availability of the service types. The first car rental organisation is located at the 50 largest airports of the world and has the following *availability constraint*:

[RENTAL CAR] [PICK-UP] (Airport code) **IN** {'ATL', 'ORD', 'LAX', 'LHR', 'DFW', 'HND', 'FRA', 'CDG', 'SFO', 'DEN', 'AMS', 'MSP', 'DTW', 'MIA', 'LAS', 'EWR', 'PHX', 'SEL', 'IAH', 'JFK', 'LGW', 'STL', 'HKG', 'MCO', 'MAD', 'YYZ', 'SEA', 'BKK', 'BOS', 'SIN', 'NRT', 'ORY', 'FCO', 'LGA', 'PHL', 'HNL', 'CVG', 'SYD', 'CLT', 'MUC', 'ZRH', 'MEX', 'BRU', 'SLC', 'KIX', 'IAD', 'FUK', 'PMI', 'PIT', 'CTS' }

The second car rental organisation is located in the largest airports in the United States of America and Canada only, which is modelled by the following availability constraint:

[RENTAL CAR] [PICK-UP] (Airport code) **IN** {'ATL', 'ORD', 'LAX', 'DFW', 'SFO', 'DEN', 'MSP', 'DTW', 'MIA', 'LAS', 'EWR', 'PHX', 'IAH', 'JFK', 'STL', 'MCO', 'YYZ', 'SEA', 'BOS', 'LGA', 'PHL', 'HNL', 'CVG', 'CLT', 'SLC', 'IAD', 'PIT' }

2.7.8 Contracting requirements

A business trip always requires two flights (outbound and inbound) and optionally a hotel reservation and a rental car reservation. Cancelling a flight reservation is necessary when other required services can not be contracted after a flight reservation has been made. We model this by the following six candidate service types.

- A. ‘Outbound flight’ of service type ‘Book flight’
- B. ‘Inbound flight’ of service type ‘Book flight’
- C. ‘Cancel outbound flight’ of service type ‘Cancel flight’
- D. ‘Hotel’ of service type ‘Book hotel’
- E. ‘Medium car’ of service type ‘Book rental car’
- F. ‘Compact car’ of service type ‘Book rental car’

In order to keep the contracting workflow as simple as possible, we use the construction in the upper part of Figure 127 as equivalent for the construction in the lower part. The places and connectors associated with states ‘skipped’ and ‘aborted’ are dashed, because they do not apply to each candidate service type.

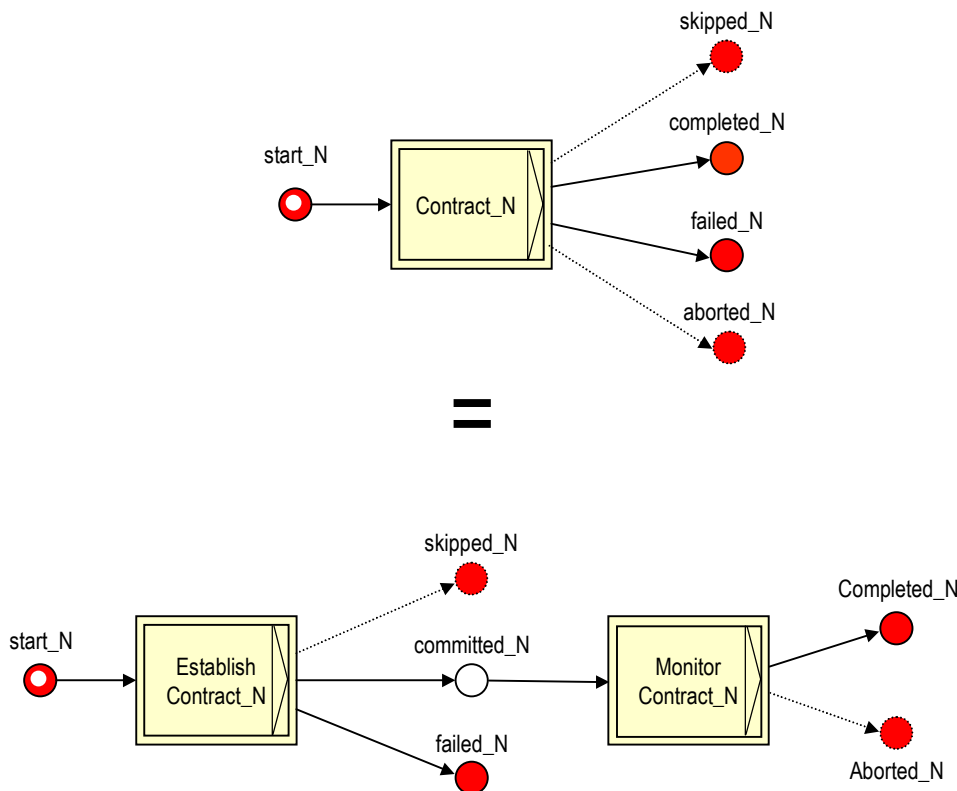


Figure 127 Basic building block used in the contracting workflow for the use case

The partial contracting workflow that contains the candidate service type transitions complemented with the transitions and places that model the triggers for candidate service types is given in Figure 128. The entire contracting workflow presented in Figure 129 is a sound WF-net.

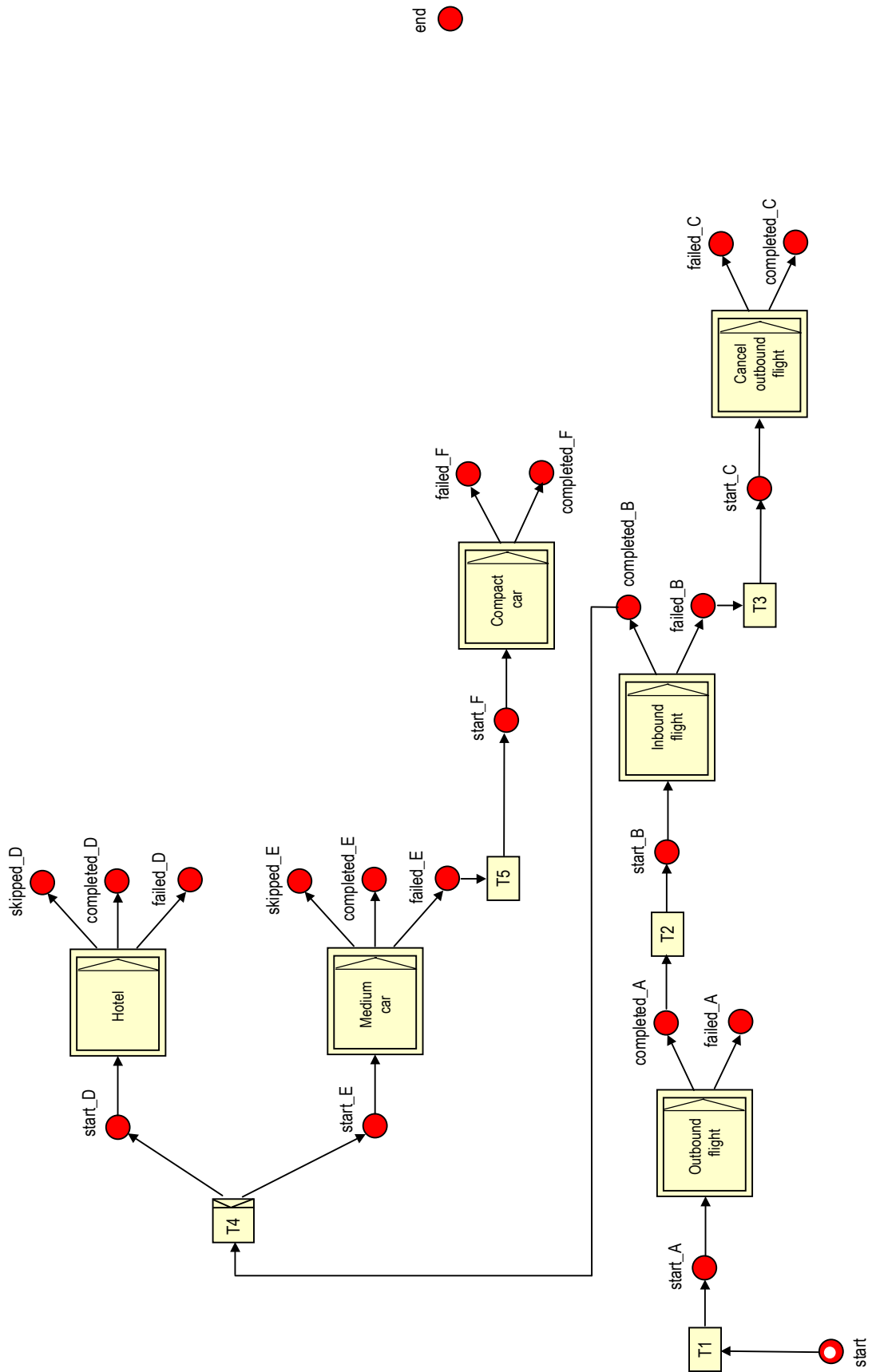


Figure 128 Partial contracting workflow corresponding to the business case

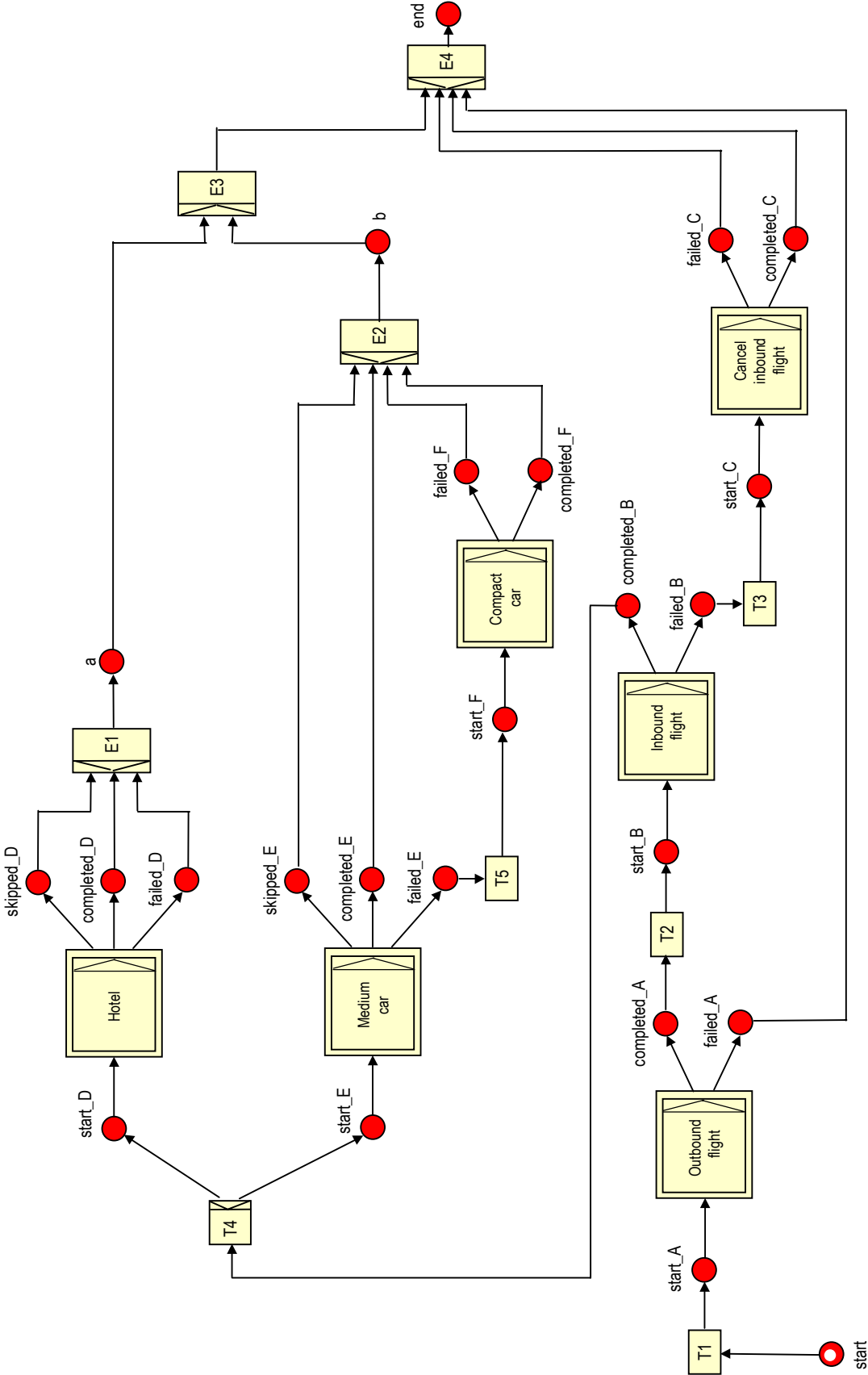


Figure 129 Sound contracting workflow corresponding to the business case

Candidate service type 'Outbound flight'

- **Specification rules:** the service data for the outbound flight is completely determined by the case data.

Outbound flight : [FLIGHT] (Class)	= 'E'
Outbound flight : [FLIGHT] [DEPARTURE] (City)	= CASE : [TRIP] (City of departure)
Outbound flight : [FLIGHT] [DEPARTURE] (Earliest date)	= CASE : [TRIP] (Latest date of arrival) - 1
Outbound flight : [FLIGHT] [ARRIVAL] (City)	= CASE : [TRIP] (City of destination)
Outbound flight : [FLIGHT] [ARRIVAL] (Latest date)	= CASE : [TRIP] (Latest date of arrival)
Outbound flight : [FLIGHT] [ARRIVAL] (Latest time)	= CASE : [TRIP] (Latest time of arrival)
Outbound flight : [FLIGHT] [CLIENT] (Name)	= CASE : [TRIP] [EMPLOYEE] (Name)
Outbound flight : [FLIGHT] [CLIENT] (Initials)	= CASE : [TRIP] [EMPLOYEE] (Initials)
Outbound flight : [FLIGHT] [CLIENT] (Sex)	= CASE : [TRIP] [EMPLOYEE] (Mr / Mrs)

- **Trigger:** none (contracting starts immediately).
- **Constraints:** none (the candidate service type is always required)
- **Negotiation strategy:** The corresponding service type (Book flight) is controlled by a transaction protocol in which the negotiation phase is based on the 'multiple non-binding offers' pattern. Therefore, it is possible to follow a parallel negotiation approach by sending the flight preferences to all available airlines. There are two optimisation criteria: (i) select the flight with lowest price from the set of possible flights that arrive no later than the latest arrival date/time and no earlier than 4 hours before the latest arrival date/time. If there are no flights that match this criterion, the following optimisation criterion is applied: (ii) select the flight with the latest arrival date/time that arrives no later than the latest arrival date/time.

Candidate service type 'Inbound flight'

- **Specification rules:** the service data for the inbound flight is also completely determined by the case data.

Inbound flight : [FLIGHT] (Class)	= 'E'
Inbound flight : [FLIGHT] [DEPARTURE] (City)	= CASE : [TRIP] (City of destination)
Inbound flight : [FLIGHT] [DEPARTURE] (Earliest date)	= CASE : [TRIP] (Earliest date of departure)
Inbound flight : [FLIGHT] [DEPARTURE] (Earliest time)	= CASE : [TRIP] (Earliest time of departure)
Inbound flight : [FLIGHT] [ARRIVAL] (City)	= CASE : [TRIP] (City of departure)
Inbound flight : [FLIGHT] [CLIENT] (Name)	= CASE : [TRIP] [EMPLOYEE] (Name)
Inbound flight : [FLIGHT] [CLIENT] (Initials)	= CASE : [TRIP] [EMPLOYEE] (Initials)
Inbound flight : [FLIGHT] [CLIENT] (Sex)	= CASE : [TRIP] [EMPLOYEE] (Mr / Mrs)

- **Trigger:** the inbound flight can be booked after the outbound flight has been booked.

Inbound flight : NEGOTIATION AFTER Outbound flight = COMPLETED
--

- **Constraints:** none (the candidate service type is always required)
- **Negotiation strategy:** the negotiation strategy for this candidate service type is similar to the one of the candidate service type 'Outbound flight'. The earliest date/time of departure is used instead of the latest date/time of arrival.

Candidate service type ‘Cancel outbound flight’

- **Specification rules:** the service data is completely determined by the service data of candidate service type ‘Outbound flight’.

Cancel outbound flight : [FLIGHT] (Code)	= Outbound flight : [FLIGHT] (Code)
Cancel outbound flight : [FLIGHT] (Booking reference)	= Outbound flight : [FLIGHT] (Booking reference)

- **Trigger:** the outbound flight must be cancelled when the inbound flight could not be booked.

Cancel outbound flight : NEGOTIATION AFTER Inbound flight = FAILED
--

- **Negotiation strategy:** the negotiation protocol does not require any additional decisions.

Candidate service type ‘Hotel’

- **Specification rules:** The service data for the overnight stay in the hotel is partially determined by the case data and partially determined by the details of the contracted flights.

Hotel : [ROOM] (Requested type)	= ‘S’
Hotel : [ROOM] [LOCATION] (City)	= CASE : [TRIP] (City of destination)
Hotel : [ROOM] [ARRIVAL] (Planned date)	= Outbound flight : [FLIGHT] [ARRIVAL] (Schedule date)
Hotel : [ROOM] [ARRIVAL] (Planned time)	= Outbound flight : [FLIGHT] [ARRIVAL] (Schedule time)
Hotel : [ROOM] [DEPARTURE] (Planned date)	= Inbound flight : [FLIGHT] [DEPARTURE] (Schedule date)
Hotel : [ROOM] [DEPARTURE] (Planned time)	= Inbound flight : [FLIGHT] [DEPARTURE] (Schedule time)
Hotel : [ROOM] [CLIENT] (Name)	= CASE : [TRIP] [EMPLOYEE] (Name)
Hotel : [ROOM] [CLIENT] (Initials)	= CASE : [TRIP] [EMPLOYEE] (Initials)
Hotel : [ROOM] [CLIENT] (Sex)	= CASE : [TRIP] [EMPLOYEE] (Mr / Mrs)

- **Trigger:** this candidate service type can only be contracted when the inbound flight and the outbound flight have been contracted because the specification rules and the constraints use service attributes of the inbound flight and outbound flight.

Hotel : NEGOTIATION AFTER (Inbound flight = COMPLETED AND Outbound flight = COMPLETED)

- **Constraints:** This candidate service type must be contracted only if the arrival date of the outbound flight is not equal to the departure date of the inbound flight.

Outbound flight : [FLIGHT] [ARRIVAL] (Schedule date) <> Inbound flight : [FLIGHT] [DEPARTURE] (Schedule date)
--

- **Negotiation strategy:** the transaction protocol is based on the ‘binding request’ pattern. A sequential approach is therefore required for this candidate service type.

Candidate service type 'Medium car'

- **Specification rules:** the service data for the preferred rental car service is partially derived from the case data and partially derived from the contracted inbound flight. This is because the pick-up location of the rental car is equal to the airport of arrival.

Medium car : [CAR] (Type of car)	= 'M'
Medium car : [CAR] [PICK-UP] (Planned date)	= Outbound flight : [FLIGHT] [ARRIVAL] (Schedule date)
Medium car : [CAR] [PICK-UP] (Planned time)	= Outbound flight : [FLIGHT] [ARRIVAL] (Schedule time)
Medium car : [CAR] [PICK-UP] (Airport code)	= Outbound flight : [FLIGHT] [ARRIVAL] (Airport code)
Medium car : [CAR] [PICK-UP] (Flight number)	= Outbound flight : [FLIGHT] (Flight number)
Medium car : [CAR] [RETURN] (Planned date)	= Inbound flight : [FLIGHT] [DEPARTURE] (Schedule date)
Medium car : [CAR] [RETURN] (Planned time)	= Inbound flight : [FLIGHT] [DEPARTURE] (Schedule time)
Medium car : [CAR] [RETURN] (Airport code)	= Inbound flight : [FLIGHT] [DEPARTURE] (Airport code)
Medium car : [CAR] [RETURN] (Flight number)	= Inbound flight : [FLIGHT] (Flight number)
Medium car : [CAR] [CLIENT] (Name)	= CASE : [TRIP] [EMPLOYEE] (Name)
Medium car : [CAR] [CLIENT] (Initials)	= CASE : [TRIP] [EMPLOYEE] (Initials)
Medium car : [CAR] [CLIENT] (Sex)	= CASE : [TRIP] [EMPLOYEE] (Mr / Mrs)

- **Trigger:** this candidate service type can only be contracted when the inbound flight and the outbound flight have been contracted, because the specification rules uses service attributes from the inbound flight and outbound flight candidate service types.

Medium car : NEGOTIATION AFTER (Inbound flight = COMPLETED AND Outbound flight = COMPLETED)
--

- **Constraint:** this candidate service must be contracted only if the case attribute 'rental car yes/no' equals 'Y'.

CASE : [TRIP] (Rental car yes/no) = 'Y'
--

- **Negotiation strategy:** the transaction protocol is based on the 'binding request' pattern. A sequential approach is therefore required for this candidate service.

Candidate service type 'Compact car'

- **Specification rules:** the service data for the alternative rental car service is identical to the service data of the 'Medium car' candidate service type, except for the attribute 'Requested type'.

Compact car : [CAR] (Requested type)	= 'C'
Compact car : [CAR] [PICK-UP] (Planned date)	= Outbound flight : [FLIGHT] [ARRIVAL] (Schedule date)
Compact car : [CAR] [PICK-UP] (Planned time)	= Outbound flight : [FLIGHT] [ARRIVAL] (Schedule time)
Compact car : [CAR] [PICK-UP] (Location)	= Outbound flight : [FLIGHT] [ARRIVAL] (Airport code)
Compact car : [CAR] [PICK-UP] (Flight number)	= Outbound flight : [FLIGHT] (Flight number)
Compact car : [CAR] [RETURN] (Planned date)	= Inbound flight : [FLIGHT] [DEPARTURE] (Schedule date)
Compact car : [CAR] [RETURN] (Planned time)	= Inbound flight : [FLIGHT] [DEPARTURE] (Schedule time)
Compact car : [CAR] [RETURN] (Location)	= Inbound flight : [FLIGHT] [DEPARTURE] (Airport code)
Compact car : [CAR] [RETURN] (Flight number)	= Inbound flight : [FLIGHT] (Flight number)
Compact car : [CAR] [CLIENT] (Name)	= CASE : [TRIP] [EMPLOYEE] (Name)
Compact car : [CAR] [CLIENT] (Initials)	= CASE : [TRIP] [EMPLOYEE] (Initials)
Compact car : [CAR] [CLIENT] (Sex)	= CASE : [TRIP] [EMPLOYEE] (Mr / Mrs)

- **Trigger:** this candidate service type will only be contracted when the candidate service type 'Medium car' could not be contracted.

Compact car : NEGOTIATION AFTER Medium car = FAILED

- **Constraint:** none (already covered by constraint on 'Medium car')
- **Negotiation strategy:** the transaction protocol is based on the 'binding request' pattern. A sequential approach is therefore required for this candidate service type.

3. Logical architecture of the Contracting Agent

This chapter defines the logical architecture of a software component for the class of service contracting processes of which the concepts have been given in Chapter 2. We will refer to this software component as the ‘Contracting Agent’ from now on. First, we address the relation of the logical architecture to the conceptual framework presented in Chapter 2. Because the term ‘architecture’ is used in many different ways, we continue by giving our definition of the term ‘software architecture’ used in this research (3.1.2) and the design goals by which the architecture is judged (3.1.3). Thereafter we give the logical architecture of the Contracting Agent according to this definition (3.2 – 3.4).

3.1 Introduction

3.1.1 Relation to the conceptual framework

Chapter 2 addressed service contracting processes on a *conceptual* level in which the entire world (service client, service providers, contracting processes) was viewed as a Petri net. Because of this abstraction, we were able to define the conceptual framework for expressing a service contracting process as a sound workflow net. In this chapter, we will decrease the level of abstraction and view the world as a network of interacting software components. Therefore, we need to map the concepts to software components. As we have stated in Section 1.6.3, we have chosen to design the software component according to the principles of *workflow management*: separation of execution and control. The former is housed in one or more application components for data storage and data transformation, whereas the latter is implemented in a workflow engine that triggers the earlier mentioned application components. The workflow engine is configured by an explicit model of the service contracting process, which we will call the *implementation contracting workflow*. Although the conceptual domain and the implementation domain both involve a contracting workflow, they are not identical. In the conceptual contracting workflow, the entire state data is modelled as tokens flowing through the net and all transformation on the state data is performed by the transitions. In the implementation domain however, state data is stored *outside* the contracting workflow as much as possible and transformations are performed by application components *outside* the contracting workflow. The function of the implementation contracting workflow is to *trigger* the right transformation of the state data at the right moment and with the right parameters.

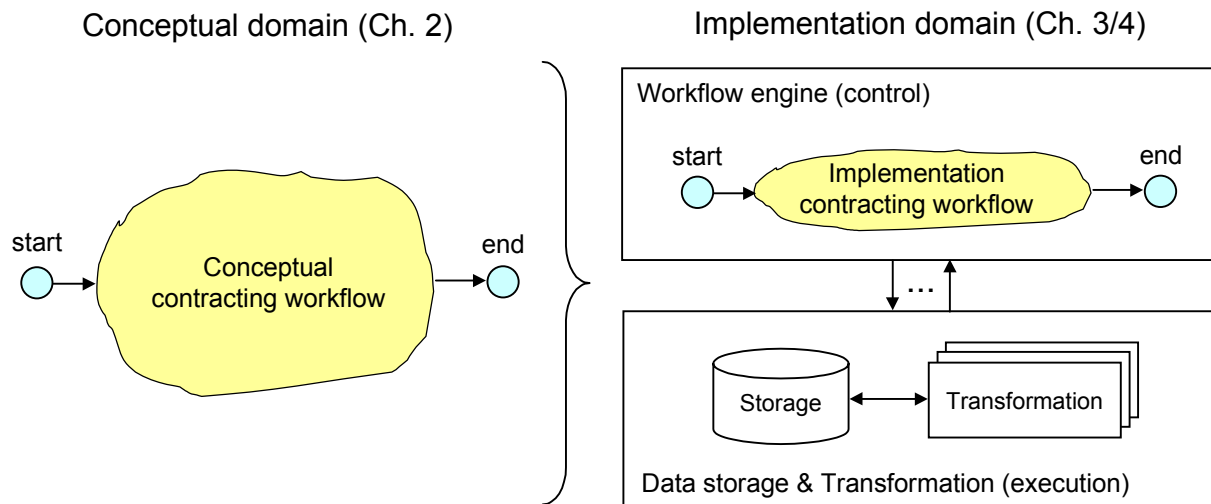


Figure 130 Relation of the Conceptual domain versus the Implementation domain

3.1.2 Definition of the term architecture

The term ‘software architecture’ is used in many different ways and there is no universally accepted definition of it, nor is the purpose of a software architecture well defined. However, there is a consensus that a software architecture is not a complete specification of a system but an abstraction of it, capturing the essentials of a software system and omitting information that is not relevant for the purpose of the architecture (which we will define below). In this research, we will use the following definition of software architecture.

Definition: *software architecture*

The high-level specification of the organisation of a software system in terms of its components, the distribution of functionality and data among the components, the externally visible properties of the components and the collaboration between components. Externally visible properties of a component encompasses both structural (static) and behavioural (dynamic) aspects.

In our view, the purpose of a software architecture is to demonstrate that the system - if organised according to the architecture - satisfies the requirements set to the system. These requirements cover aspects as functionality, flexibility, performance, scalability, reliability, etc. (see Section 3.1.3). Furthermore, since conflicting requirements often occur, the architecture of a system can be used in the design phase as a means of communication between stakeholders to the system. Finally, being a high-level specification of a software system, the software architecture is the starting point for a detailed specification of the entire software system. Major design-decisions are always made in the software architecture first, and have their effect in the detailed specification thereafter.

A ‘good’ software architecture is always the right balance between too little information and too much information. On one hand, the architecture must provide enough information for analyses and decision making. On the other hand, the architecture must abstract away from all details that are irrelevant for this purpose. Clearly, an architecture is more than a diagram consisting of boxes connected by lines. Although these drawings give an idea of the sub-

components in a system, they fail to capture the structure and behaviour of components and therefore miss out essential aspects of the system.

In this chapter, we will present a *logical* architecture of the Contracting Agent system. The prefix ‘logical’ before the term ‘architecture’ means that we abstract from the physical implementation of the components. We will use the modelling techniques of high-level coloured Petri nets and functional data modelling to define the logical architecture. In the next sections, we will define the sub-components of the Contracting Agent by the following information.

- **Distribution of functionality over components**

The first part of an architecture identifies the components and gives a high-level specification of the functionality of each component. We will use informal drawings (box-and-line drawings) and plain English for this purpose.

- **Structure of interfaces and persistent data**

After having identified the components, we define the structure of the interfaces exposed by each software component and the structure of the persistent data maintained by each component. We use the formalism of high-level Petri nets and functional data models as modelling techniques. A component is modelled as a *system*. The interfaces of a component are modelled as *places* connected to the system. Persistent data in a component is modelled as a *store* in the system. The static aspects of the interfaces and persistent data are defined by the colour of the tokens in the places, which defines the *semantics* of the data. We will use functional data models as modelling technique for this purpose.

- **Behaviour on the interfaces**

Finally, we define the behaviour of each component on its interfaces and specify the collaboration between the components via their interfaces. The behaviour of a component is defined by a Petri net, which we will call the behaviour net, and contains just as many transitions as are required to model the *externally visible* behaviour of the component.

3.1.3 Design goals

This section defines the design goals according to which the logical architecture presented in the next sections is judged.

- **Functionality**

The first design goal is to create a logical architecture that brings the functionality of executing service contracting processes according to the conceptual framework in Chapter 2. Furthermore, the following additional requirements must be fulfilled too:

- *Multiple client applications*

A client application is an information system that delegates the execution of service contracting processes to the Contracting Agent. It must be possible to use one Contracting Agent by different client applications simultaneously.

- *Multiple service contracting processes*

The Contracting Agent must allow an arbitrary number of service contracting processes to be executed simultaneously and independently. The situation where one service contracting process influences another service contracting process is outside the scope of this research.

- **Maximum use of domain knowledge**

The second design goal is to make maximum use of domain knowledge for service contracting processes in order to minimise the effort to configure the Contracting Agent. The conceptual framework presented in Chapter 2 is the basis for our domain knowledge and gives us standard building blocks (protocol patterns and contracting strategies). Furthermore, the conceptual framework provides us with a set of rules on how to construct a sound contracting workflow from these standard building blocks.

- **Extensibility**

The third design goal is to create a logical architecture that brings maximal flexibility in adapting to changes in the functionality of the Contracting Agent. This goal can be obtained by housing functionality that logically belongs together and is subject to change in a separate sub-component with a standardised interface. The conceptual framework contains several parts that are subject to change:

- *Constraint syntax*

Constraints must be expressed in a syntax. The conceptual framework in Chapter 2 presented a simple syntax for constraints. Since we do not claim that this syntax can be used to express all possible constraints to hierarchic data sets, the architecture must allow different constraint syntaxes to coexist in a Contracting Agent. Furthermore, the architecture must support adding a new constraint syntax to the Contracting Agent in a ‘plug-and-play’ kind of way.

- *Schema syntax*

Data models must be expressed in a syntax. The conceptual framework in Chapter 2 presented an object model for data models in Chapter 2. From now on, we will use the term ‘schema’ instead of the term ‘data model’. Since we do not claim that the object model in Chapter 2 can be used to capture all possible characteristics of a hierarchic data set, the architecture must allow different schema syntaxes to coexist in a Contracting Agent. Furthermore, the architecture must support adding a new schema syntax to the Contracting Agent in a ‘plug-and-play’ kind of way.

- *Transformation syntax*

Transformation functions (e.g. specification rules) must be expressed in a syntax. The conceptual framework in Chapter 2 presented a simple syntax for transformation functions. Since we do not claim that this syntax can be used to express all possible transformation functions, the architecture must allow different transformation syntaxes to coexist in a Contracting Agent. Furthermore, the architecture must support adding a new transformation syntax to the Contracting Agent in a ‘plug-and-play’ kind of way.

- *Protocol patterns*

Each service type has a transaction protocol for which we have presented a number of patterns in Chapter 2. These patterns are used as domain knowledge in the Contracting Agent to define transaction protocols more efficiently. Since we do not claim that the presented protocol patterns cover all situations, the architecture must allow new transaction protocol patterns to be added in a ‘plug-and-play’ kind of way.

- *Contracting strategies*

The conceptual framework defined the service contracting process as a workflow consisting of ‘negotiation’, ‘execution’ and ‘acceptance’ transitions that have a standard interface to their environment. Each of these transitions implements a negotiation, execu-

tion or acceptance strategy. Since we do not claim that the presented strategies cover all possible situations, the architecture must allow new contracting strategies to be added in a ‘plug-and-play’ kind of way.

- **Specialisation**

By dividing the entire functionality of a system into smaller, autonomous sub-components, we allow these sub-components to be developed by different actors. A user is then able to select the best-of-breed sub-components for his particular situation.

- **Re-use**

A component that implements a generic function can be used in more than one system. Re-using an existing component has a number of advantages: time, quality and costs.

- **Maintainability**

By dividing a complex information system into multiple less-complex components, we contribute to the maintainability of the system as a whole. The components should be designed such that each component supports a well-defined function of a complexity that is significantly smaller than the complexity of the entire system.

3.2 Architecture of the ‘Contracting Agent’ component

3.2.1 Distribution of functionality over components

The Contracting Agent consists of three components, as illustrated in Figure 132 in which we use the symbols shown in Figure 131. We distinguish two types of actors: humans and software components. A program interface is an interface by which a software component interacts with another software component. A user interface is an interface by which a software component interacts with a human.

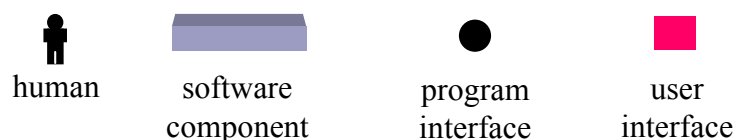


Figure 131 Legend of symbols used

- **The Server component**

The Server component is responsible for executing service contracting processes on behalf of one or more client applications. For that purpose, it has a program interface by which it interacts with these client applications. The Server component has no user interface and uses configuration parameters in which the structure of the service contracting process is defined. It offers a program interface by which its configuration parameters can be modified. Finally, the Server component offers a program interface via which its state data can be queried.

- **The Configurator component**

The Configurator component is responsible for creation and modification of the configuration parameters used by the Server component. It offers a user interface by which the configuration parameters can be created, inspected and modified. It uses the program interface of a Repository component to load information about service types, service providers, pro-

to col patterns and contracting strategies. Finally, it uses the program interface of the Server component to submit new configuration parameters to the Server component, after having checked them for completeness and correctness.

- **The Monitor component**

The Monitor component is responsible for querying the state data of the Server component and presenting the results to the user. The functionality of a Monitor user interface may be tailored to one or more specific types of service contracting processes. There is however a minimum set of functions that must be supported by every Monitor component. This minimum set of functions is to get an overview of all business cases for which a service contracting process is executing or has been executed. Furthermore, for each instance of a service contracting process the user must be able to view the entire state data.

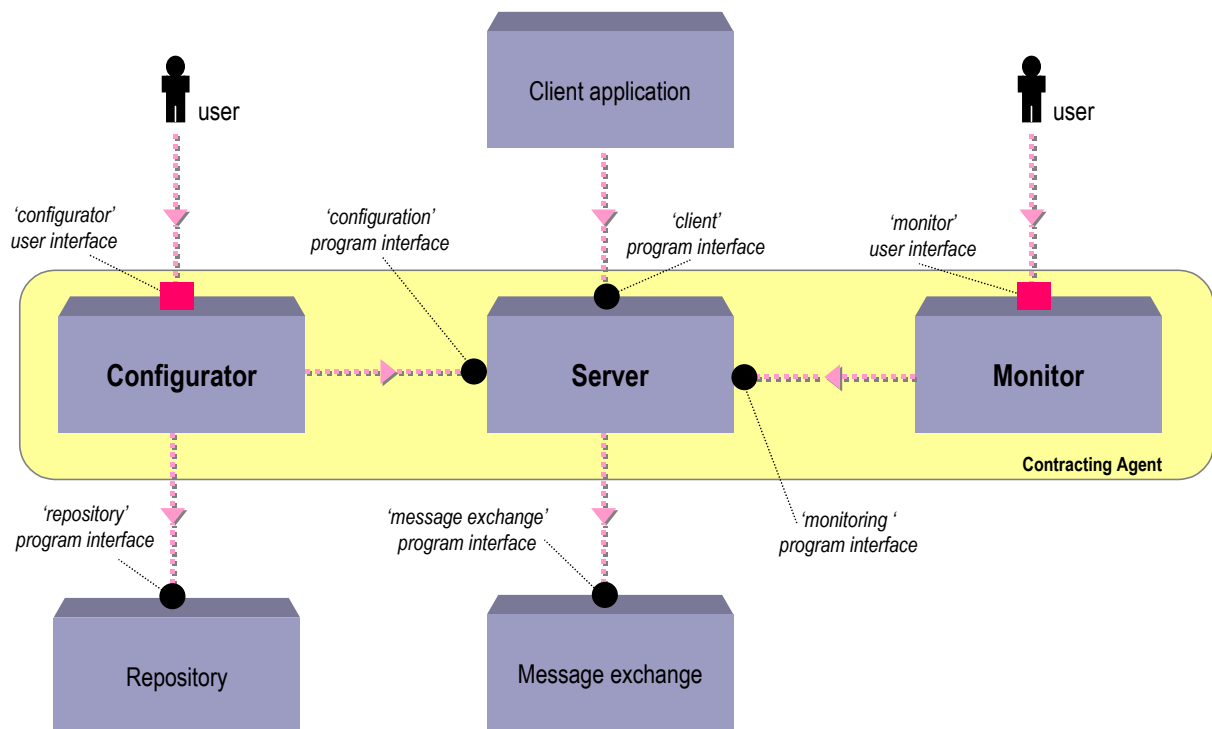


Figure 132 Main components and interfaces of the Contracting Agent

3.2.2 Structure of interfaces

We have defined the Contracting Agent as composed of three components: Server, Configurator and Monitor. Furthermore, the Contracting Agent interfaces with at least two external components: client applications and message exchange applications. A formal model of the interfaces between client application, Server, Configurator, Monitor and message exchange application is given in Figure 133.

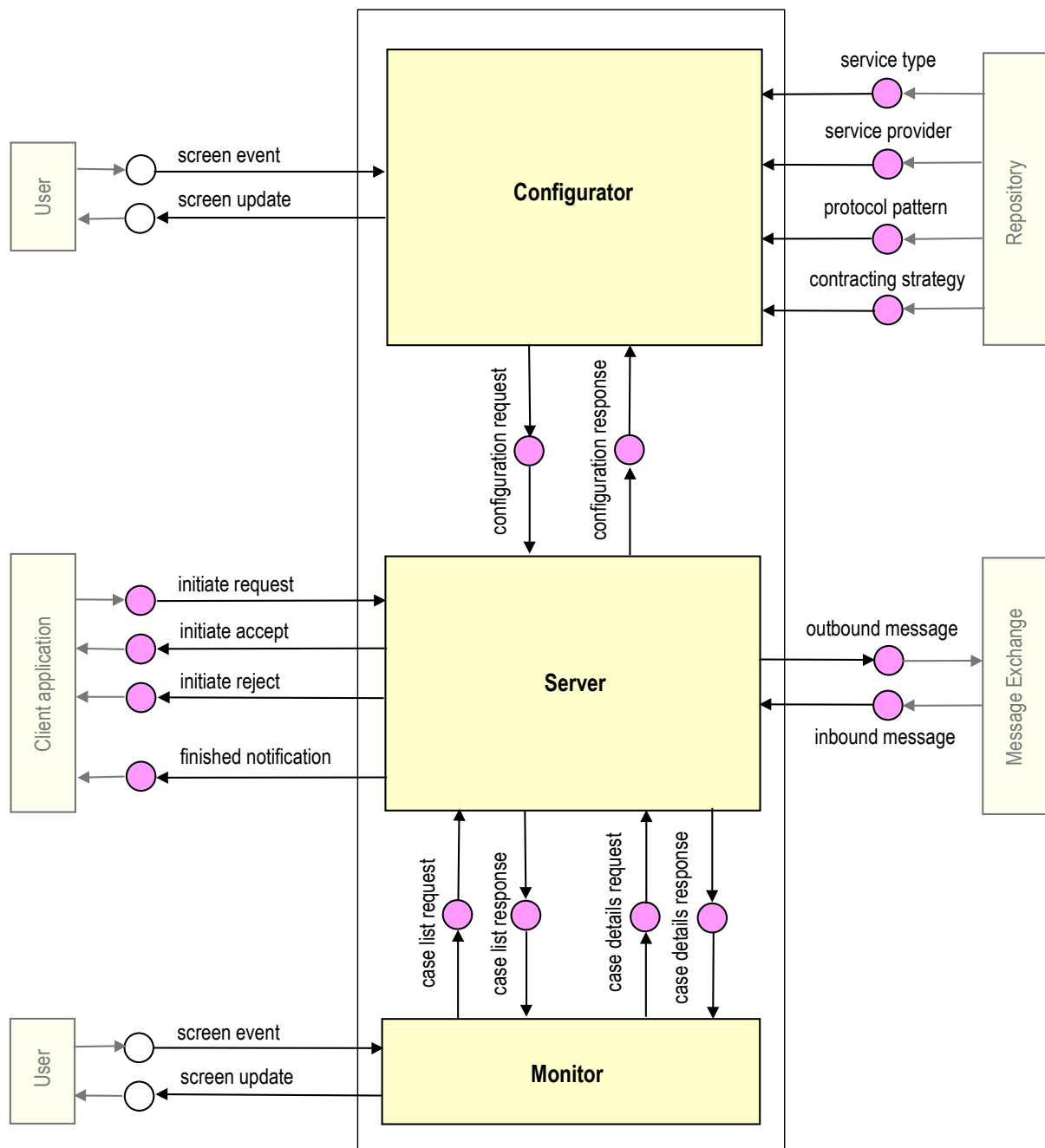


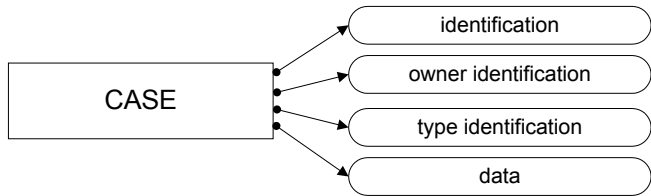
Figure 133 Interfaces between sub components in the Contracting Agent

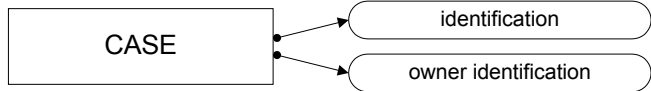
We will now define the structure of the following program interfaces:

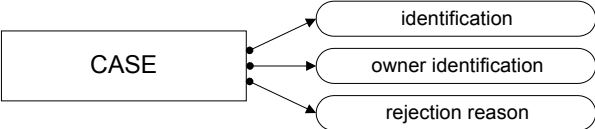
- The 'client' program interface;
- The 'message exchange' program interface;
- The 'monitoring' program interface;
- The 'configuration' program interface;
- The 'repository' program interface.

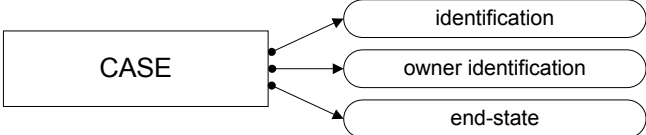
Structure of the Client program interface

A client application must be able to initiate the execution of a service contracting process in the Contracting Agent. For that purpose, a client application passes a business case to the Contracting Agent, who will perform a validity check, after which the business case is either accepted or rejected. If the business case is rejected, the reason of rejection must be passed to the client application. If the business case is accepted, the service contracting process will be started after which the client application will receive a confirmation of the initiation. When the service contracting process ends, the client application receives a notification with the end-state. This interface consists of four places of which we will now define the colour.

	Place 'initiate request'
Role	A token in this place models a request from a client application to the Server component to start a service contracting process for a business case.
Type	<p>The colour of the place is a complex of which the object model is given in Figure 134. The 'identification' attribute models the identification of the business case as assigned by the client application. The 'owner identification' attribute models the unique identification of the client application making the request. The combination of the attributes 'identification' and 'owner identification' must be unique. The case type and case data are modelled by the 'type identification' attribute and the 'data' attribute respectively.</p> 
	<p>Figure 134 Object model for the colour of the 'initiate request' place</p>

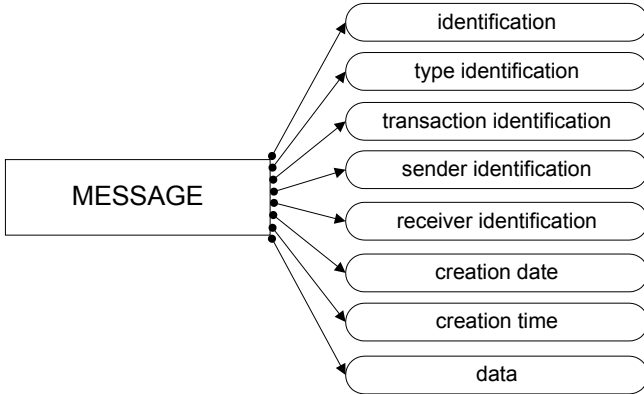
	Place 'initiate accept'
Role	A token in this place models the positive response of the Server component to an 'initiate request' message and notifies the client application that the service contracting process has started.
Type	<p>The colour of the place is a complex of which the object model is given in Figure 135. The 'identification' and 'owner identification' attributes identify the business case uniquely.</p> 
	<p>Figure 135 Object model for the colour of the 'initiate accept' place</p>

Role	Place 'initiate reject'
	A token in this place models the negative response of the Server component to an 'initiate request' message and notifies the client application that the service contracting process could not be started due to a specified error.
Type	<p>The colour of the place is a complex of which the object model is given in Figure 136. The 'identification' and 'owner identification' attributes identify the business case uniquely. The 'rejection reason' attribute models the reason of the rejection, which can be:</p> <ul style="list-style-type: none"> - 'case identification missing' - 'case identification is not unique' - 'case type missing' - 'case type invalid or unknown' - 'case data missing' - 'case data invalid: ' <error description>
	
<p>Figure 136 Object model for the colour of the 'initiate reject' place</p>	

Role	Place 'finished notification'
	A token in this place models the notification of the Server component to the client application that a service contracting process has ended.
Type	<p>The colour of the place is a complex of which the object model is given in Figure 137. The 'identification' and 'owner identification' attributes identify the business case uniquely. The 'end-state' attribute models the state in which the process ended and indicates whether all required services have been contracted and whether violations have occurred. Further details of the contracting process can be retrieved by the client application via the Monitor interface.</p>
	
<p>Figure 137 Object model for the colour of the 'finished notification' place</p>	

Structure of the Message exchange program interface

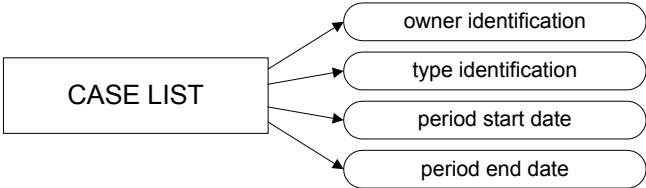
A Server component sends messages to service providers and receives messages from service providers. The exchange of messages requires well-known functions like data-conversion, communication, authentication, etc. Because message exchange is a generic function for which standard software components are available, we choose not to include the function as part of the Contracting Agent but to interface with one or more standard message exchange components. This interface consists of two places of which we will now define the colour.

Role	Place 'outbound message'
	A token in this place models a message from service client to service provider passed by the Server component to the message exchange component for processing.
Type	<p>The colour of this place is a complex of which the object model is given in Figure 138. The 'identification' attribute models the unique identification of the message in combination with the 'sender identification'. The 'transaction identification' attribute models the identification of the business transaction to which the message belongs. The 'sender identification' and 'receiver identification' attributes identify the actors between which the message is exchanged. The 'type identification' attribute is used to identify the message type. The 'creation date' and 'creation time' attributes identify the date and time at which the service client created the message. Finally, the 'data' attribute is used to contain the entire message data.</p>  <p style="text-align: center;"><i>Figure 138</i> Object model for the colour of the 'outbound message' place</p>

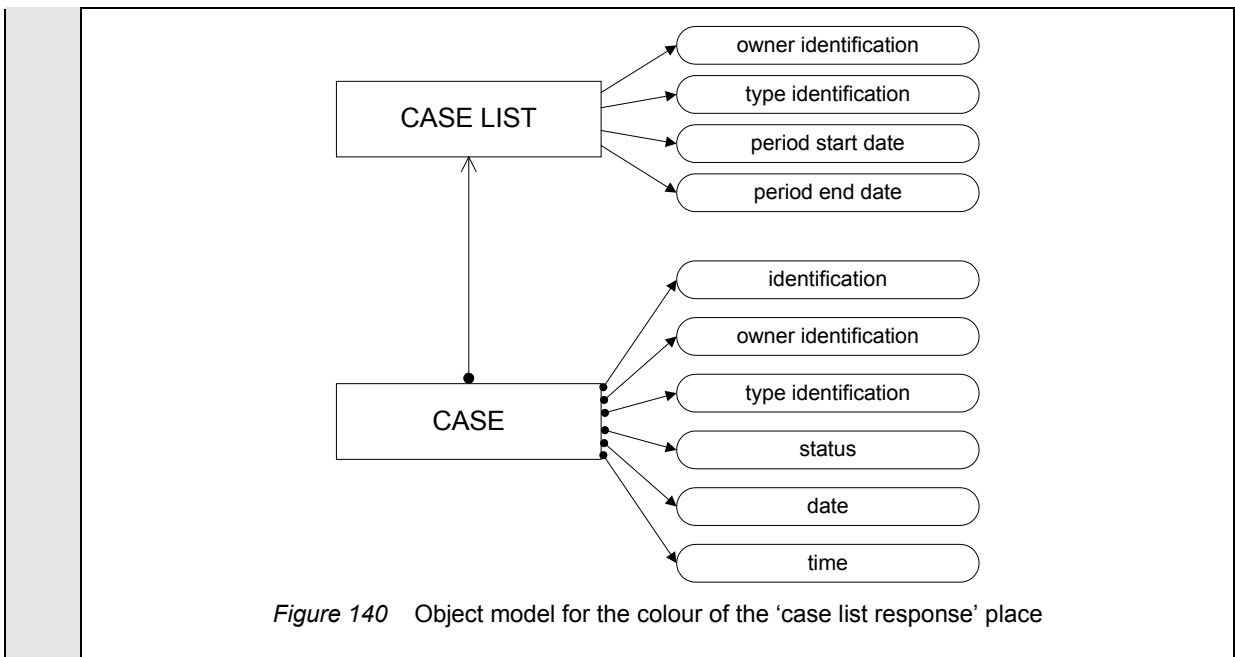
Role	Place 'inbound message'
	A token in this place models a message from service provider to service client passed to the Server component by the message exchange component.
Type	Identical to place 'outbound message'.

Structure of the Monitoring program interface

The Monitor component is used to query the state of the Server component and present the information to the user in a suitable way. To retrieve the state data from the Server component, the Server component offers two functions. The first function is used to retrieve a list of business cases for which a service contracting process is running, or has been completed. The second function is used to retrieve the entire state data associated with the service contracting process of one specific business case. This interface consists of four places of which we will now define the colour.

Place 'case list request'	
Role	A token in this place models a request to the Server component to retrieve a list of cases stored in the state data that match certain selection criteria.
Type	<p>The colour of this place is a complex of which the object model is given in Figure 139. The attributes of the complex are used as selection criteria. The 'owner identification' attribute is used to limit the response to cases from one owner. The 'type identification' attribute is used to limit the response to cases from one type. The 'period start date' and 'period end date' attributes are used to limit the response to cases for which the service contracting process started in a specific date interval.</p>  <p style="text-align: center;"><i>Figure 139</i> Object model for the colour of the 'case list request' place</p>

Place 'case list response'	
Role	A token in this place models the response from the Server component to a request for a list of cases stored in the state data.
Type	<p>The colour of this place is a complex of which the object model is given in Figure 140. The 'CASE LIST' entity is copied from the case list request and has a relation to zero, one or more 'CASE' entities with the following attributes. The 'identification' attribute identifies the business case uniquely in combination with the 'owner identification' attribute. The 'type identification' attribute contains the identification of the case type. The 'status' attribute contains a code that identifies the state of the service contracting process uniquely and is an item from the set {'rejected', 'executing', 'completed'}. The 'date' and 'time' attributes contain the date and time at which the case was received from the client application by the Server component and created in the state data.</p>



Place 'case details request'	
Role	A token in this place models a request to the Server component for the entire state data associated with the service contracting process for one specific business case.
Type	The colour of this place is a complex of which the object model is given in Figure 141. The ' <i>identification</i> ' attribute identifies the business case uniquely in combination with the ' <i>owner identification</i> ' attribute.
<i>Figure 141</i> Object model for the colour of the 'case details request' place	

Place 'case details response'	
Role	A token in this place models the response from the Server component to a case details request.
Type	The colour of this place is identical to the colour of the state data store in Figure 152. A token in this place contains zero or one 'CASE' entities.

Structure of the Configuration program interface

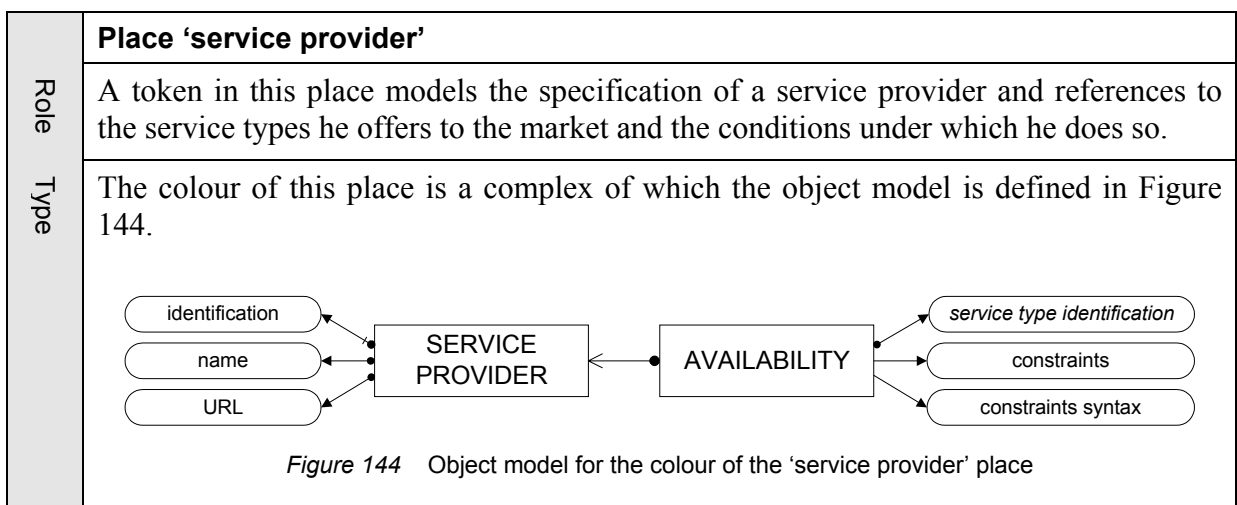
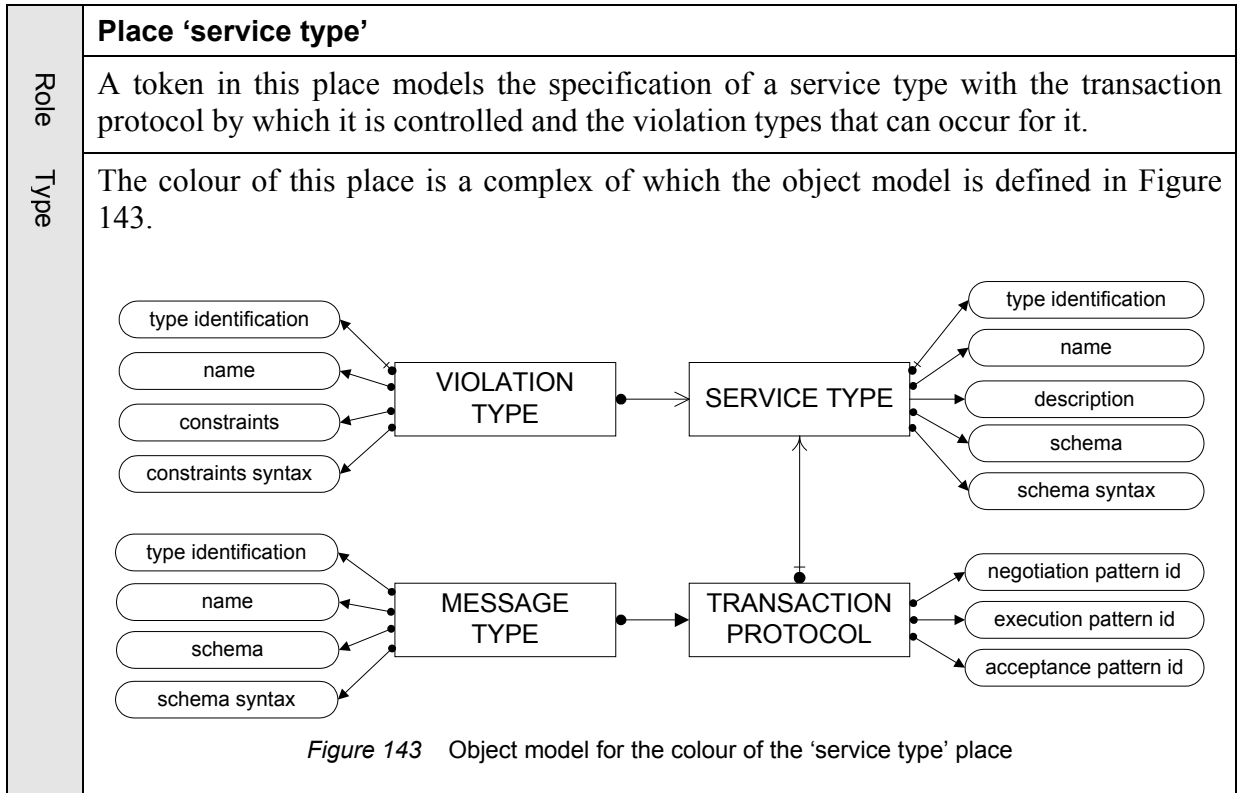
The Configurator component is used to maintain the configuration parameters used by the Server component. However, the Configurator component can not modify the configuration parameters of the Server component directly because of temporally inconsistencies that will exist inevitably during modification of the configuration parameters. Therefore, the Configurator component and the Server component each have their own copy of the configuration parameters. The Configurator component operates on its own copy of the configuration parameters, without affecting the working of the Server component. The Server component offers an interface by which the Configurator component can submit its own copy of the configuration parameters to update the existing configuration parameters of the Server component. This interface consists of two places of which we will now define the colour.

Place 'configuration request'	
Role	A token in this place models the request to the Server component to update its set of configuration parameters with new configuration parameters which are valid within a specified period by setting the start date and end date.
Type	The colour of this place is a complex of which the object model is equal to the object model of the configuration data in the Server component, which is defined in Figure 153, extended with an attribute ' <i>workflow definition</i> ' of entity 'CONFIGURATION'.

Place 'configuration response'	
Role	A token in this place models the response from the Server component to a previous request to update its configuration parameters with new configuration parameters.
Type	<p>The colour of this place is a complex of which the object model is given in Figure 142. The '<i>identification</i>' attribute refers to the identification of a prior configuration request, whereas the '<i>status</i>' attribute indicates whether the update request was executed successfully or not. The value of the attribute is an element from the set {'accepted', 'rejected'}.</p> <div style="text-align: center;"> <pre> graph LR CONFIGURATION[CONFIGURATION] --> identification([identification]) CONFIGURATION --> status([status]) </pre> </div>
<p><i>Figure 142</i> Object model for the colour of the 'configuration response' place</p>	

Structure of the Repository program interface

The Configurator component is used to maintain the configuration parameters of the Server component. Because the offerings of service providers are relevant for many companies, it is likely to have an external Repository from which a service client can import service types, service providers, protocol patterns and contracting strategies. The Repository program interface defines the structure of the information exchanged from Repository to Configurator. This interface consists of four places of which we will now define the colour.



	<p>Place 'protocol pattern'</p>
<p>Role</p>	<p>A token in this place models the specification of a protocol pattern, consisting of a protocol definition and the generic name of each message type used in the protocol definition.</p>
<p>Type</p>	<p>The colour of this place is a complex of which the object model is defined in Figure 145.</p>
<div style="text-align: center;"> <pre> classDiagram class PATTERN_MESSAGE_TYPE { type_identification generic_name } class PROTOCOL_PATTERN { identification name protocol_definition } PATTERN_MESSAGE_TYPE --> PROTOCOL_PATTERN </pre> </div> <p><i>Figure 145</i> Object model for the colour of the 'protocol pattern' place</p>	

	<p>Place 'contracting strategy'</p>
<p>Role</p>	<p>A token in this place models the specification of a strategy type for a specific protocol pattern with its parameter types.</p>
<p>Type</p>	<p>The colour of this place is a complex of which the object model is defined in Figure 146.</p>
<div style="text-align: center;"> <pre> classDiagram class PARAMETER_TYPE { type_identification name } class STRATEGY_TYPE { type_identification protocol_pattern_identification name description } PARAMETER_TYPE --> STRATEGY_TYPE </pre> </div> <p><i>Figure 146</i> Object model for the colour of the 'contracting strategy' place</p>	

3.2.3 Behaviour on the component interfaces

The behaviour of the Server component on the interfaces it exposes to client applications and the message exchange component is defined in Figure 147. A token in place ‘*initiate request*’ is followed by either a token in place ‘*initiate accept*’ or in place ‘*initiate reject*’. After a token is produced in place ‘*initiate accept*’, messages can be exchanged with service providers. When a token is produced in place ‘*finished notification*’, the process is in an end-state and no messages can be exchanged with service providers any more.

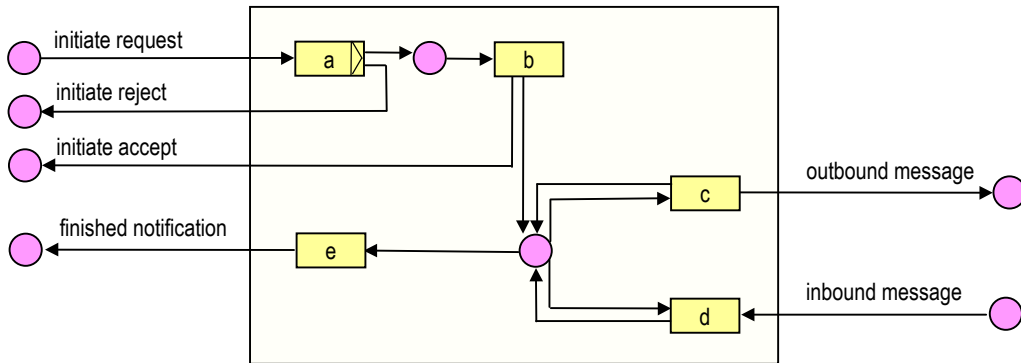


Figure 147 Behaviour of the Server component on the client application interface

The behaviour of the Server component on the interface it exposes to the Configurator component follows a simple request/response protocol defined in Figure 148.

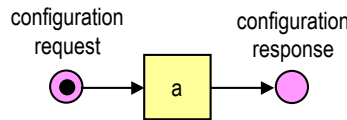


Figure 148 Behaviour of the Server component on the configuration program interface

The behaviour of the Server component on the interface it exposes to the Monitor component follows a simple request/response protocol defined in Figure 149.

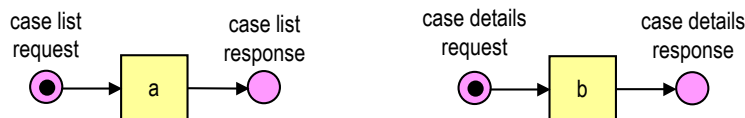


Figure 149 Behaviour of the Server component on the monitor program interface

3.3 Architecture of the ‘Server’ component

3.3.1 Distribution of functionality over components

The Server component is responsible for executing service contracting processes on behalf of client applications. A conceptual model of service contracting processes has been given in Chapter 2, where we modelled the entire service contracting process in a conceptual contracting workflow. Because we viewed the service contracting process at a conceptual level, the conceptual contracting workflow combined both execution and control. The entire state data of

the service contracting process was modelled as case tokens flowing through the workflow net and operations on the state data were performed by the firing of transitions that consumed and produced case tokens. This section addresses the mapping of the conceptual model on a software architecture. The first architectural decision is to separate execution from control. There will at least be three types of components: (i) storage components for state data and configuration data, (ii) processing components that perform the necessary operations on the state data and (iii) a workflow engine, that triggers the right operations on the right moment and with the right parameters. An important decision that has to be made next is the type of processing components that will be used. One option is to implement each standard operation (see Section 2.6.3) to the state data in a separate processing component, which accesses the storage components directly. However, since these standard operations are very specific for service contracting processes, they will almost certain not be suited for re-use. Another option is to implement a generic function (e.g. transformation) in a processing component, that is invoked with parameters and therefore needs no direct access to the storage components. A major advantage of this approach is the ability to re-use these processing components later, or to use existing processing components. Furthermore, since there is no direct access of the processing components to the storage components, storage and processing components have become more independent of each other. A drawback is that in order to perform one standard operation on the state data (e.g. ‘create outbound message’), one has to perform several queries on the storage components to retrieve state data and configuration data, invoke one or more processing components with the retrieved data as parameters after which several queries on the storage components have to be performed to store the result of the processing. If we want all operations on storage components and processing components to be triggered by the workflow engine, the workflow definition will become too complex and too dependent of the storage and processing components. The solution for this is to add an extra component, the interaction manager, that is used between the workflow engine and the storage and processing components. This new component offers an interface to the workflow engine by which the workflow engine can trigger operations with the highest possible granularity, after which the new component performs the required smaller operations on the storage components and processing components.

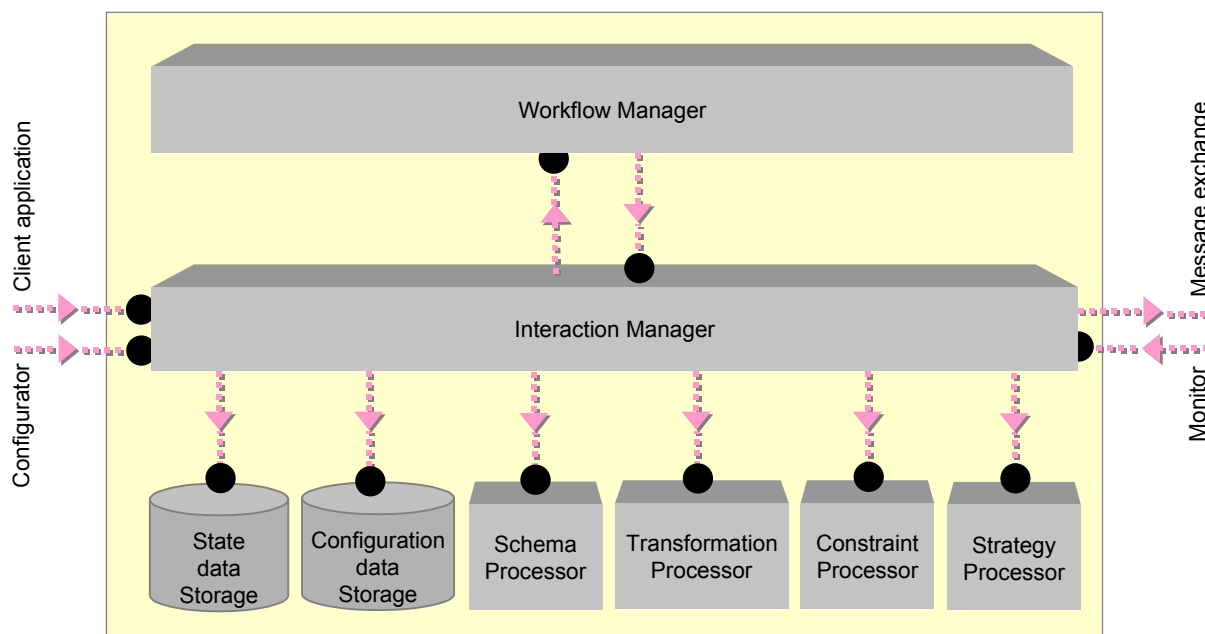


Figure 150 The server component is designed as a workflow application

- **State data Storage**

This component is responsible for maintaining the state data of service contracting processes executed by the Server component. The component exposes a program interface by which the state data can be inspected and modified. It is the responsibility of the component to handle large volumes of data and to maintain the integrity of the state data.

- **Configuration data Storage**

This component is responsible for maintaining the configuration data required for the execution of service contracting processes. The component exposes a program interface by which the configuration data can be inspected and modified.

- **Schema Processor**

This component is responsible for applying a schema to a hierarchic data set. The component exposes an interface by which it takes a hierarchic data set and a schema as input. Output is the set of validation errors that have been detected (can be empty).

- **Transformation Processor**

This component is responsible for the transformation of one hierarchic data set into another. The component exposes an interface by which it takes a hierarchic data set and a transformation function as input. Output is the hierarchic data set, which is the result of the transformation.

- **Constraint Processor**

This component is responsible for applying constraints to a hierarchic data set. The component exposes an interface by which it takes a hierarchic data set and constraints as input. Output is a list of constraints that have not been fulfilled (can be empty).

- **Strategy Processor**

This component is responsible for evaluating inbound messages and for adjusting service data, according to a contracting strategy parameter. The component exposes an interface by which it takes a strategy parameter type with a hierarchic data set as input. Output is either a message value or a modified hierarchic data set.

- **Workflow Manager**

This component is responsible for the control of service contracting processes. The component is configured by an implementation contracting workflow that defines the structure of service contracting processes.

- **Interaction Manager**

This component is the ‘glue’ between all other Server sub-components. All other Server components communicate with the Interaction manager only. Furthermore, all communication of the Server component with external components (client applications, message exchange application, Configurator component, Monitor component) is handled by the Interaction Manager. The Interaction Manager is passive; it acts only when it is triggered by another component.

3.3.2 Structure of interfaces and persistent data

We will first define the interfaces between the components by modelling the components as systems and the interfaces as places between the systems.

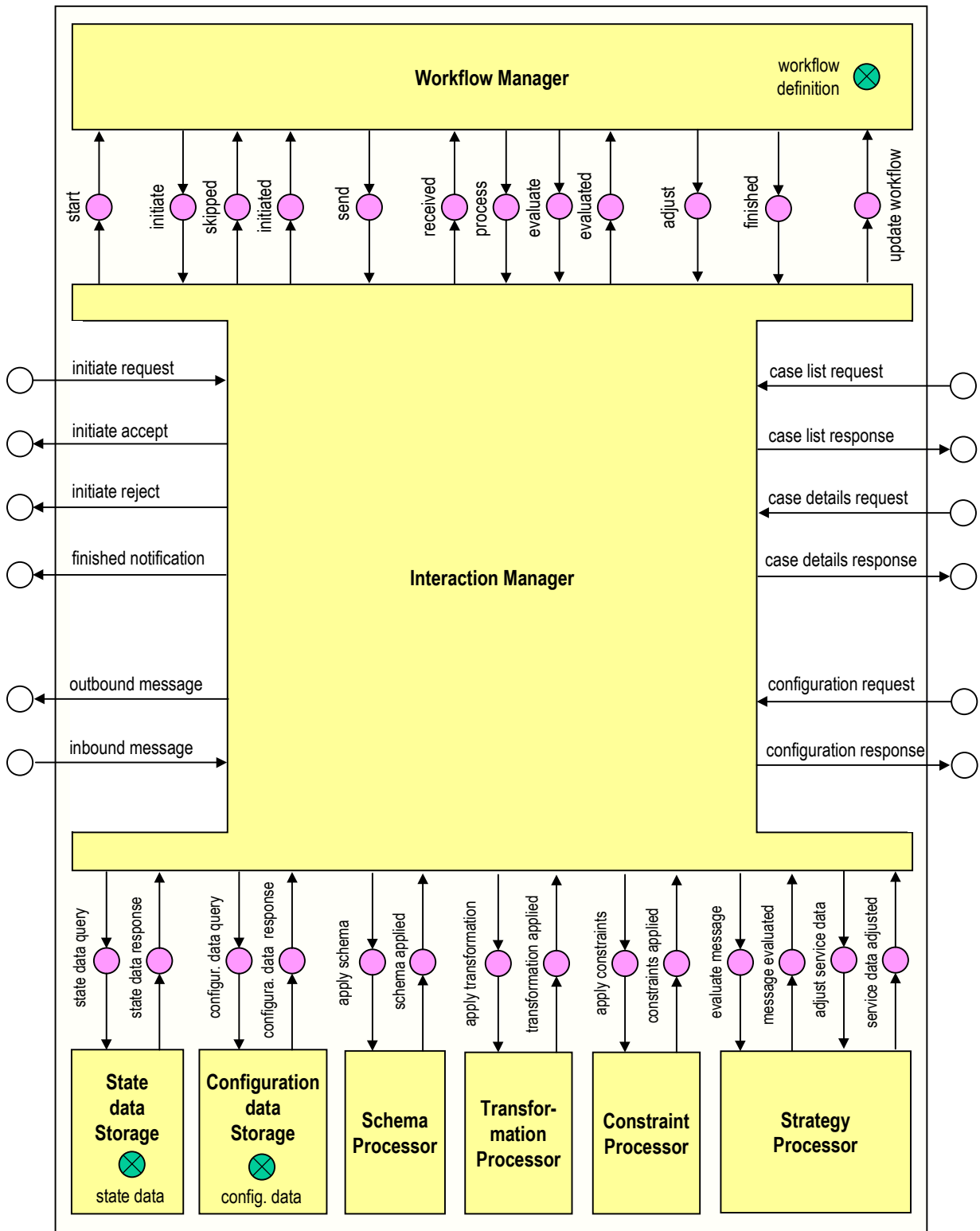


Figure 151 Sub-components in the Contracting Agent Server

Structure of persistent data

The stores '*state data*' and '*configuration data*' in the 'Server' system model persistent data of the Server component. The colour of the tokens in these stores is defined by the object models in Figure 152 and Figure 153 respectively.

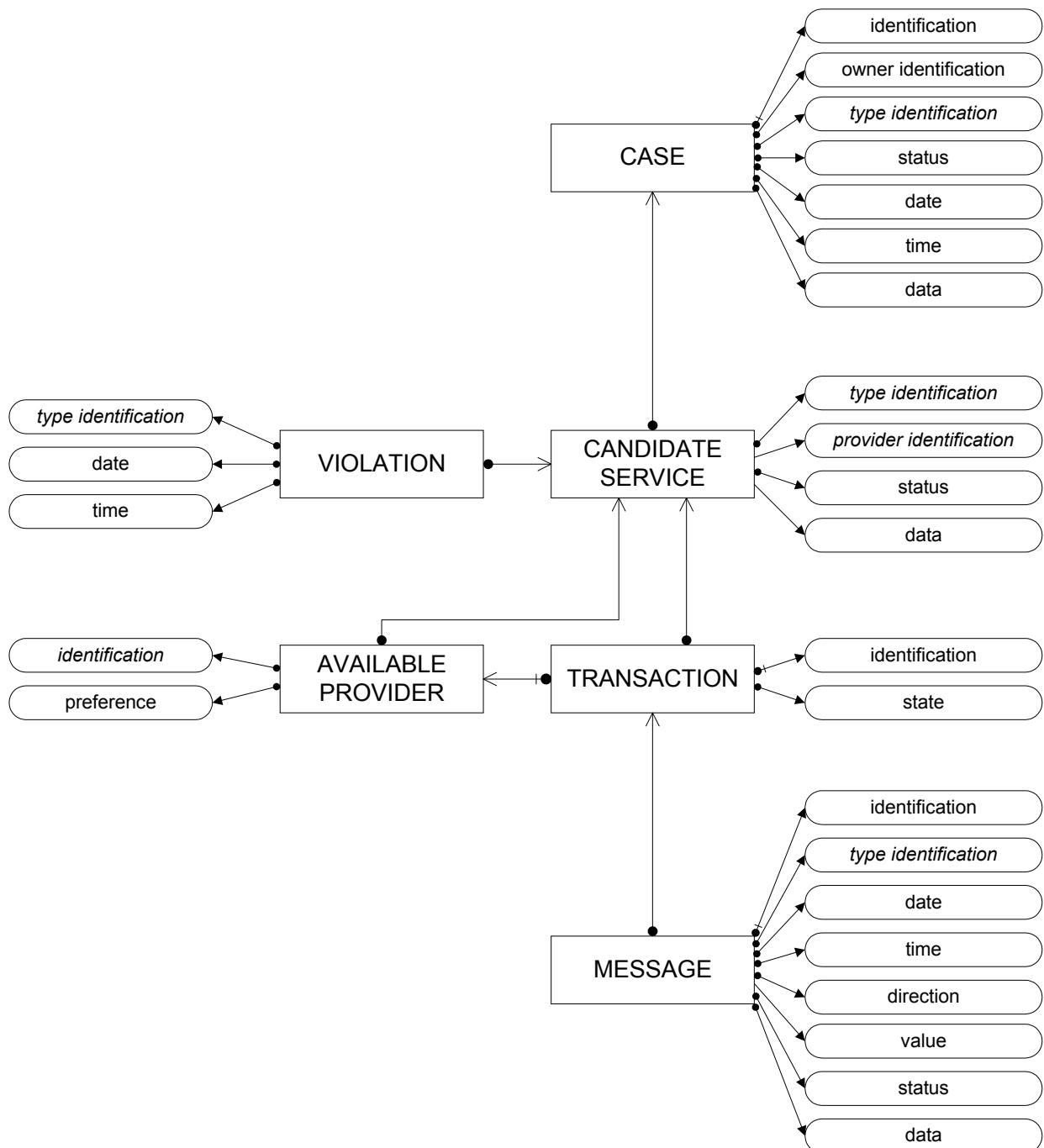


Figure 152 Structure of persistent data store 'state data' of the Server component

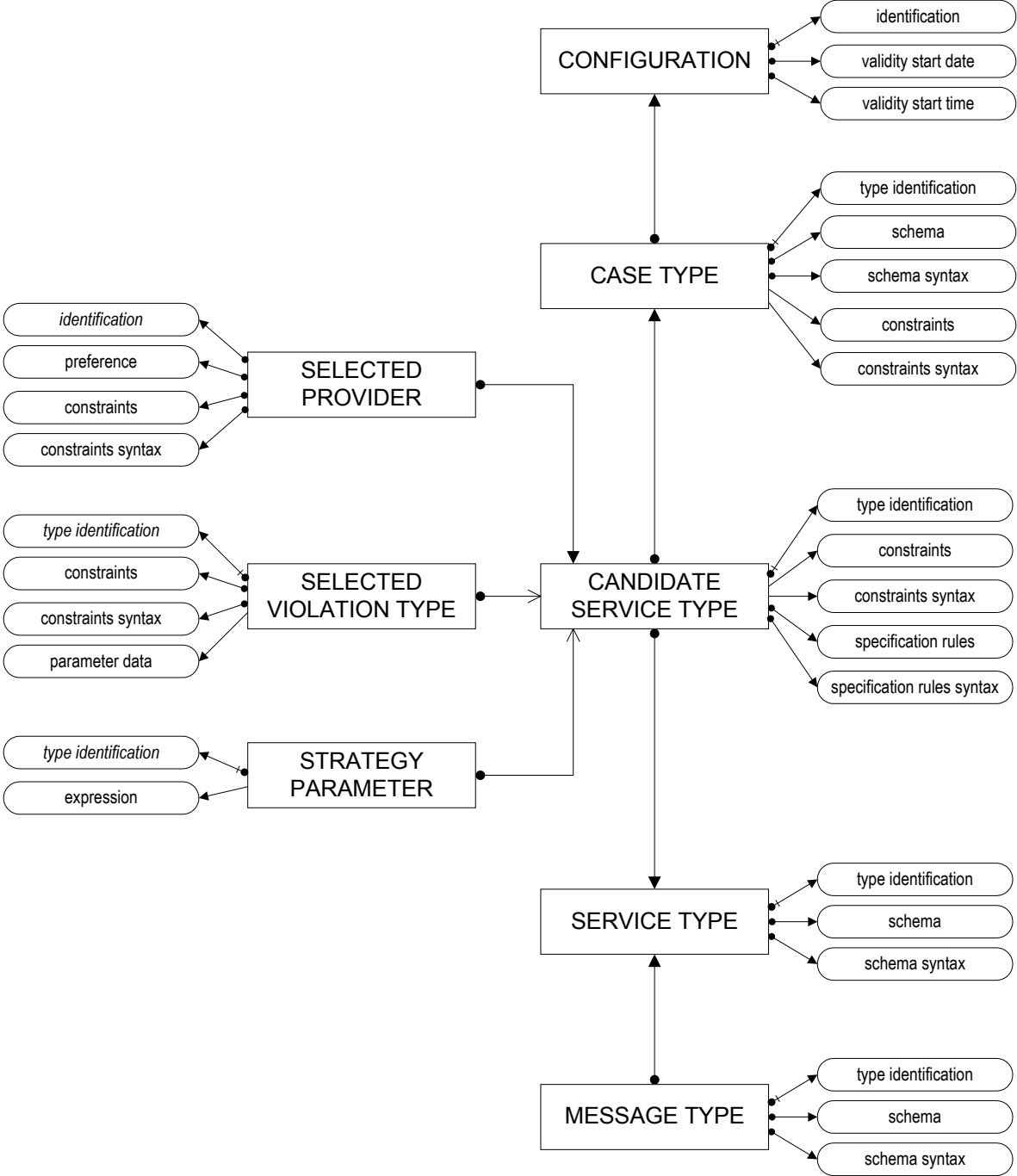
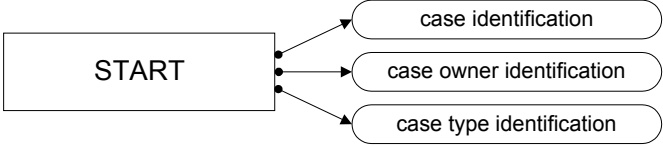
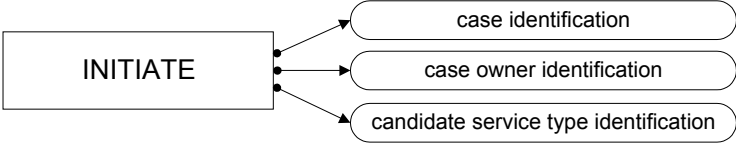


Figure 153 Structure of persistent data in store 'configuration data' of the Server component

Structure of the Workflow Manager interface

The Workflow Manager exposes a *configuration* interface to the Interaction Manager to update its configuration parameters (the contracting workflow). Furthermore, the Workflow Manager exposes an *operational* interface to the Interaction Manager to control the flow of events in a service contracting process. We assume that the Interaction Manager handles requests in FIFO order. Therefore, we do not model a response signal for every request signal. A response signal is only used when the Workflow Manager needs information generated by the Interaction Manager during handling of the request.

	Place 'start'
Role	A token in the 'start' place models a signal from the Interaction Manager to the Workflow Manager to create a new instance of a service contracting process for a specific business case.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 154. The token contains three attributes: the 'case identification' and 'case owner identification' attributes identify the business case uniquely. The 'case type identification' attribute models the case type.</p>  <p style="text-align: center;">Figure 154 Object model for the colour of the 'start' place</p>

	Place 'initiate'
Role	A token in the 'initiate' place models a signal from the Workflow Manager to the Interaction Manager that a service contracting process for a business case is at the point where the contracting of a specific candidate service type must be initiated.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 155. The token contains three attributes: the 'case identification' and 'case owner identification' attributes identify a business case uniquely and the 'candidate service type identification' attribute identifies a candidate service type for the associated case type.</p>  <p style="text-align: center;">Figure 155 Object model for the colour of the 'initiate' place</p>

	<p>Place 'skipped'</p>
<p>Role</p>	<p>A token in the <i>'skipped'</i> place models a signal from the Interaction Manager to the Workflow Manager that a candidate service type must be skipped in the service contracting process for a business case.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 156, which consists of the same attributes as the object model of place <i>'initiate'</i>.</p> <div data-bbox="486 571 1225 712" data-label="Diagram"> <pre> graph LR SKIPPED[SKIPPED] --> CI(case identification) SKIPPED --> COI(case owner identification) SKIPPED --> CSTI(candidate service type identification) </pre> </div> <p>Figure 156 Object model for the colour of the 'skipped' place</p>

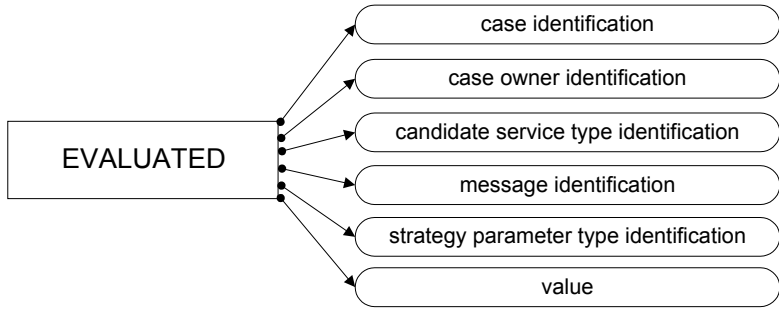
	<p>Place 'initiated'</p>
<p>Role</p>	<p>A token in the <i>'initiated'</i> place models a signal from the Interaction Manager to the Workflow Manager that a new candidate service is created in the state data, of which the service data is specified and the available providers are determined.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 157. The token contains four attributes. The attributes <i>'case identification'</i>, <i>'case owner identification'</i> and <i>'candidate service type identification'</i> are copied from the <i>'INITIATE'</i> entity. The <i>'number of available providers'</i> attribute models the number of <i>'AVAILABLE PROVIDER'</i> entities that has been created for the candidate service in the state data. Because each <i>'AVAILABLE PROVIDER'</i> entity has a unique sequence number that indicates the preference, it is sufficient to pass the number of providers. The contracting workflow references a provider by its sequence number, which can be translated to the identity via the <i>'AVAILABLE PROVIDER'</i> entity in the state data.</p> <div data-bbox="486 1467 1225 1662" data-label="Diagram"> <pre> graph LR INITIATED[INITIATED] --> CI(case identification) INITIATED --> COI(case owner identification) INITIATED --> CSTI(candidate service type identification) INITIATED --> NAPI(number of available providers) </pre> </div> <p>Figure 157 Object model for the colour of the 'initiated' place</p>

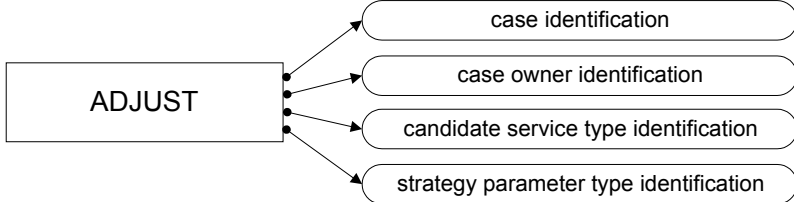
	<p>Place 'send'</p>
<p>Role</p>	<p>A token in the 'send' place models a signal from the Workflow Manager to the Interaction Manager to create a new outbound message of a specific type in a specific transaction. If the message is the first message in the transaction, the transaction must be created in the state data too.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 158. The 'case identification' and 'case owner identification' attributes identify a business case uniquely. These attributes, together with the 'candidate service type identification' attribute and the 'provider identification' attribute identify the business transaction in which the outbound message must be created uniquely. Finally, the 'message type identification' attribute identifies the message type to be created in the business transaction.</p> <div data-bbox="486 817 1225 1075" data-label="Diagram"> </div> <p>Figure 158 Object model for the colour of the 'send' place</p>

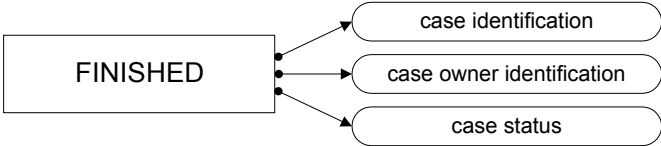
	<p>Place 'received'</p>
<p>Role</p>	<p>A token in the 'received' place models a signal from the Interaction Manager to the Workflow Manager that an inbound message has been received in a business transaction.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 159. The structure of the token is identical to the structure of the tokens in place 'send', except for an extra attribute 'message identification'.</p> <div data-bbox="486 1568 1225 1877" data-label="Diagram"> </div> <p>Figure 159 Object model for the colour of the 'received' place</p>

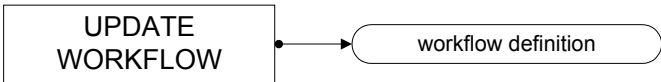
Role	<p>Place 'process'</p> <p>A token in the 'process' place models a signal from the Workflow Manager to the Interaction Manager that the message data of an inbound message must be used to update the service data of the candidate service it belongs to after which the service data must be checked for violations.</p>
	Type

Role	<p>Place 'evaluate'</p> <p>A token in the 'evaluate' place models a signal from the Workflow Manager to the Interaction Manager to evaluate an expression on the message data of an inbound message.</p>
	Type

	<p>Place 'evaluated'</p>
<p>Role</p>	<p>A token in the '<i>evaluated</i>' place models the response of the Interaction Manager to the Workflow Manager on a request to evaluate an expression on the message data of an inbound message.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 162. The first five attributes are copied from the 'EVALUATE' entity in the request. The attribute '<i>value</i>' models the value of the expression that has been applied to the message data.</p>  <p style="text-align: center;">Figure 162 Object model for the colour of the 'evaluated' place</p>

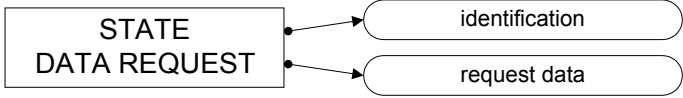
	<p>Place 'adjust'</p>
<p>Role</p>	<p>A token in the '<i>adjust</i>' place models a signal from the Workflow Manager to the Interaction Manager that the service data of a candidate service must be updated according to the rules defined in a strategy parameter.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 163. The '<i>case identification</i>' and '<i>case owner identification</i>' attributes identify a business case uniquely. The '<i>candidate service type identification</i>' attribute identifies the candidate service of which the service data must be updated. The '<i>strategy parameter type identification</i>' attribute defines the strategy parameter that contains the transformation rules that have to be applied to the current service data.</p>  <p style="text-align: center;">Figure 163 Object model for the colour of the 'adjust' place</p>

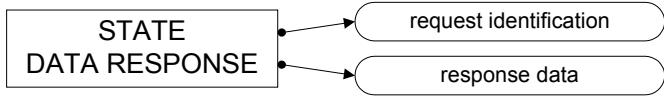
Role	Place 'finished'
	A token in the ' <i>finished</i> ' place models a signal from the Workflow Manager to the Interaction Manager that a service contracting process for a specific business case has ended.
Type	The colour of a token in this place is a complex of which the object model is shown in Figure 164. The ' <i>case identification</i> ' and ' <i>case owner identification</i> ' attributes identify a business case uniquely. The ' <i>case status</i> ' attribute defines the end-state of the service contracting process.
	
<p>Figure 164 Object model for the colour of the 'finished' place</p>	

Role	Place 'update workflow'
	A token in the ' <i>update workflow</i> ' place models a signal from the Interaction Manager to the Workflow Manager to update its workflow definition with a new workflow definition.
Type	The colour of a token in this place is a complex of which the object model is shown in Figure 165. The token contains only one attribute: ' <i>workflow definition</i> ' that contains the definition of the contracting workflow in the syntax used by the Workflow Manager.
	
<p>Figure 165 Object model for the colour of the 'update workflow' place</p>	

Structure of the State Data Storage interface

The State Data Storage exposes an operational interface to the Interaction Manager by which it can retrieve state data and modify state data.

	Place 'state data request'
Role	A token in the ' <i>state data request</i> ' place models a signal from the Interaction Manager to the State Data Storage to retrieve or update a specific part of the state data.
Type	The colour of a token in this place is a complex of which the object model is shown in Figure 166. The ' <i>identification</i> ' attribute is a unique identifier for the request. The ' <i>request data</i> ' attribute models the request itself in the syntax used by the State Data Storage component.
	
	Figure 166 Object model for the colour of the 'state data request' place

	Place 'state data response'
Role	A token in the ' <i>state data response</i> ' place models the response from the State Data Storage to the Interaction Manager on a state data request.
Type	The colour of a token in this place is a complex of which the object model is shown in Figure 167. The ' <i>request identification</i> ' attribute is used to match the response with the request. The ' <i>response data</i> ' attribute models the response to the request in the syntax used by the State Data Storage component.
	
	Figure 167 Object model for the colour of the 'state data response' place

Structure of the Configuration Data Storage interface

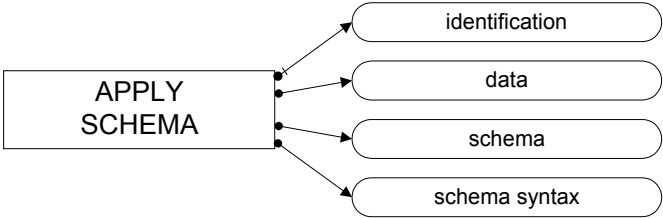
The Configuration Data Storage component exposes an operational interface to the Interaction Manager by which it can retrieve configuration data.

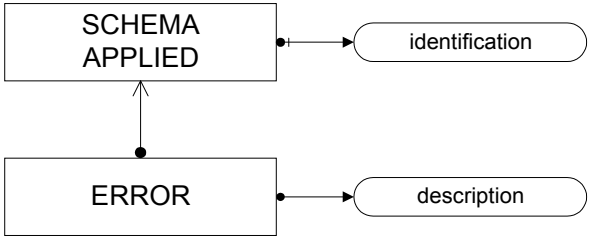
Place 'config data request'	
Role	A token in the ' <i>config data request</i> ' place models a signal from the Interaction Manager to the Configuration Data Storage to retrieve a specific part of the configuration data.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 168. The '<i>identification</i>' attribute is a unique identifier for the request. The '<i>request data</i>' attribute models the request itself in the syntax used by the Configuration Data Storage component.</p> <div style="text-align: center;"> </div> <p style="text-align: center;"><i>Figure 168</i> Object model for the colour of the 'config data request' place</p>

Place 'config data response'	
Role	A token in the ' <i>config data response</i> ' place models the response from the Configuration Data Storage to the Interaction Manager on a configuration data request.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 169. The '<i>request identification</i>' attribute is used to match the response with the request. The '<i>response data</i>' attribute models the response of the request in the syntax used by the Configuration Data Storage component.</p> <div style="text-align: center;"> </div> <p style="text-align: center;"><i>Figure 169</i> Object model for the colour of the 'config data response' place</p>

Structure of the Schema Processor interface

The Schema Processor exposes an operational interface to the Interaction Manager by which his services can be invoked.

Place 'apply schema'	
Role	A token in the 'apply schema' place models a signal from the Interaction Manager to the Schema Processor to validate a hierarchic data set subject to a schema.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 170. The token contains four attributes: the 'identification' attribute that contains a unique identification of the request, the 'data' attribute that models the hierarchic data to be validated, the 'schema' attribute that models the schema to validate with and the 'schema syntax' that identifies the syntax used to express the schema.</p>  <p style="text-align: center;">Figure 170 Object model for the colour of the 'apply schema' place</p>

Place 'schema applied'	
Role	A token in the 'schema applied' place models the response from the Schema Processor to the Interaction Manager on a request to apply a schema.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 171. The 'identification' attribute of the 'SCHEMA APPLIED' entity is used to match the response with the request. If errors have been found during the validation, the 'SCHEMA APPLIED' entity has one or more 'ERROR' entities of which the 'description' attribute contains the description of the error.</p>  <p style="text-align: center;">Figure 171 Object model for the colour of the 'schema applied' place</p>

Structure of the Constraint Processor interface

The Constraint Processor exposes an operational interface to the Interaction Manager by which his services can be invoked.

Place 'apply constraints'	
Role	A token in the 'apply constraints' place models a signal from the Interaction Manager to the Constraint Processor to apply a set of constraints to a hierarchic data set in order to check if the data fulfils the constraints.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 172. The token contains four attributes: the 'identification' attribute that contains a unique identification of the request, the 'data' attribute that models the hierarchic data to be checked, the 'constraints' attribute that models the constraints to check with and the 'constraint syntax' that models the identification of the syntax used to express the constraints.</p> <div style="text-align: center;"> <pre> graph LR A[APPLY CONSTRAINTS] --> B(identification) A --> C(data) A --> D(constraints) A --> E(constraint syntax) </pre> </div> <p style="text-align: center;">Figure 172 Object model for the colour of the 'apply constraints' place</p>

Place 'constraints applied'	
Role	A token in the 'constraints applied' place models the response from the Constraint Processor to the Interaction Manager on a apply constraints request.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 173. The 'identification' attribute of the 'CONSTRAINTS APPLIED' entity is used to match the response with the request. If errors have been found during the validation, the 'CONSTRAINTS APPLIED' entity has one or more 'ERROR' entities of which the 'description' attribute contains the description of the error.</p> <div style="text-align: center;"> <pre> graph TD E[ERROR] --> D(description) E --> CA[CONSTRAINTS APPLIED] CA --> I(identification) </pre> </div> <p style="text-align: center;">Figure 173 Object model for the colour of the 'constraints applied' place</p>

Structure of the Transformation Processor interface

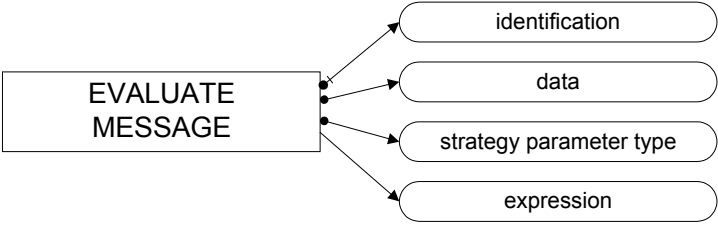
The Transformation Processor exposes an operational interface to the Interaction Manager by which its services can be invoked.

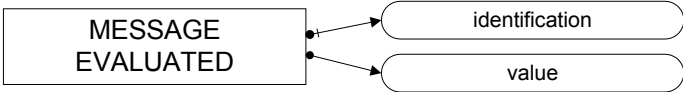
Place 'apply transformation'	
Role	A token in the ' <i>apply transformation</i> ' place models a signal from the Interaction Manager to the Transformation Processor to transform a hierarchic data set to another hierarchic data set with a different structure.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 174. The token contains four attributes: the '<i>identification</i>' attribute that contains a unique identification of the request, the '<i>data</i>' attribute that models the hierarchic data to be transformed, the '<i>transformation function</i>' attribute that models the transformation to be performed and the '<i>transformation syntax</i>' attribute that identifies the syntax in which the transformation function is expressed.</p> <div style="text-align: center;"> </div> <p style="text-align: center;"><i>Figure 174</i> Object model for the colour of the 'apply transformation' place</p>

Place 'transformation applied'	
Role	A token in the ' <i>transformation applied</i> ' place models the response from the Transformation Processor to the Interaction Manager on an apply transformation request.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 175. The '<i>identification</i>' attribute is used to match the transformation response with the transformation request. The '<i>data</i>' attribute models the hierarchic data set that was the result of the transformation operation.</p> <div style="text-align: center;"> </div> <p style="text-align: center;"><i>Figure 175</i> Object model for the colour of the 'transformation applied' place</p>

Structure of the Strategy Processor interface

The Strategy Processor exposes an operational interface to the Interaction Manager by which its services can be invoked.

Place 'evaluate message'	
Role	A token in the 'evaluate message' place models a signal from the Interaction Manager to the Strategy Processor to evaluate a hierarchic data set with message data according to a specific contracting strategy parameter.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 176. The token contains four attributes: the 'identification' attribute that contains a unique identification of the request, the 'data' attribute that models the message data to be evaluated, the 'strategy parameter type' attribute that identifies the evaluation type to be performed and the 'expression' attribute that contains the parameters to be used.</p>  <p style="text-align: center;">Figure 176 Object model for the colour of the 'evaluate message' place</p>

Place 'message evaluated'	
Role	A token in the 'message evaluated' place models the response from the Strategy Processor to the Interaction Manager on an evaluate message request.
Type	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 177. The 'identification' attribute is used to match the message evaluated response with the evaluate message request. The 'value' attribute models the message value that was the result of the evaluation operation.</p>  <p style="text-align: center;">Figure 177 Object model for the colour of the 'message evaluated' place</p>

	<p>Place ‘adjust service data’</p>
<p>Role</p>	<p>A token in the ‘<i>adjust service data</i>’ place models a signal from the Interaction Manager to the Strategy Processor to adjust the service data of a candidate service according to a contracting strategy parameter.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 178. The token contains four attributes: the ‘<i>identification</i>’ attribute that contains a unique identification of the request, the ‘<i>data</i>’ attribute that models the service data to be adjusted, the ‘<i>strategy parameter type</i>’ attribute that models the type of adjustment to be performed and the ‘<i>expression</i>’ attribute that contains any parameter value required for the adjustment.</p> <div data-bbox="507 685 1198 898" data-label="Diagram"> <pre> graph LR A[ADJUST SERVICE DATA] --> B(identification) A --> C(data) A --> D(strategy parameter type) A --> E(expression) </pre> </div> <p style="text-align: center;"><i>Figure 178</i> Object model for the colour of the ‘adjust service data’ place</p>

	<p>Place ‘service data adjusted’</p>
<p>Role</p>	<p>A token in the ‘<i>service data adjusted</i>’ place models the response from the Strategy Processor to the Interaction Manager on an request to adjust service data.</p>
<p>Type</p>	<p>The colour of a token in this place is a complex of which the object model is shown in Figure 179. The ‘<i>identification</i>’ attribute is used to match the service data adjusted response with the adjust service data request. The ‘<i>data</i>’ attribute models the adjusted service data.</p> <div data-bbox="507 1429 1198 1525" data-label="Diagram"> <pre> graph LR A[SERVICE DATA ADJUSTED] --> B(identification) A --> C(data) </pre> </div> <p style="text-align: center;"><i>Figure 179</i> Object model for the colour of the ‘service data adjusted’ place</p>

3.3.3 Behaviour on the component interfaces

The previous section defined the *structure* of the interfaces. We will now define the *behaviour* of the components on their interfaces and the collaboration between the components via their interfaces.

Storage components

The behaviour of the State Data Storage component and the Configuration Data Storage component is defined in Figure 180. Each component behaves according to a request/response protocol.

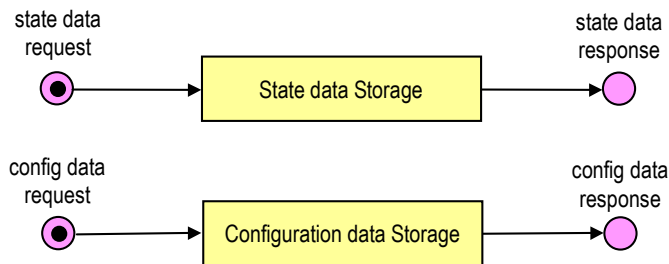


Figure 180 Behaviour of the Storage components

Processing components

The behaviour of the Schema Processor, the Constraints Processor, the Transformation Processor and the Strategy Processor is defined in Figure 181. Each component behaves according to a request/response protocol.

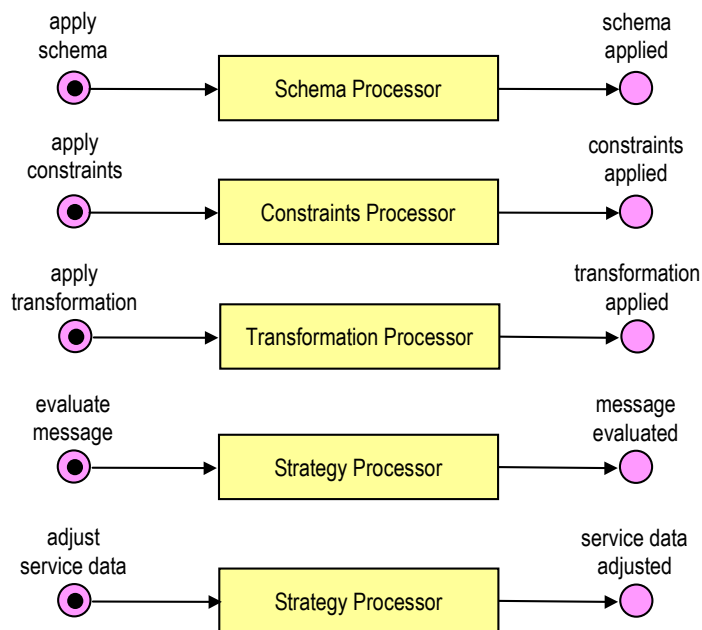


Figure 181 Behaviour of the processing components

Workflow Manager and Interaction Manager

We will now define the behaviour of the Workflow Manager and the Interaction Manager. As we have described in Section 3.3.1, the Workflow Manager triggers the Interaction Manager to perform one standard operation on the state data (see Section 2.6.1), or a group of functionally related standard operations on the state data. As a response, the Interaction Manager invokes

the storage components to retrieve relevant state data and configuration data, whereafter it invokes the required processing components. Next, the storage components are invoked again to store the result of the operation. The behaviour of the Interaction Manager towards the State Data Storage, Configuration Data Storage, Schema Processor, Transformation Processor, Constraints Processor and Strategy Processor is defined by the definition of the standard operations in Section 2.6.1. We will now define the behaviour of the Interaction Manager by defining the relation between its interface towards the Workflow Manager and the standard operations it executes.

• **Places ‘start’ and ‘finished’**

The model in Figure 182 shows that after the Interaction Manager consumes a token from place ‘*initiate request*’ it executes the standard operation ‘*validate case*’. If the validation is successful, the case is created in the state data and tokens are produced in places ‘*initiate accept*’ and ‘*start*’. If the validation is not successful, the case is not accepted and a token is produced in place ‘*initiate reject*’. When the Workflow Manager produces a token in place ‘*finished*’, the client application is notified by producing a token in place ‘*finished notification*’.

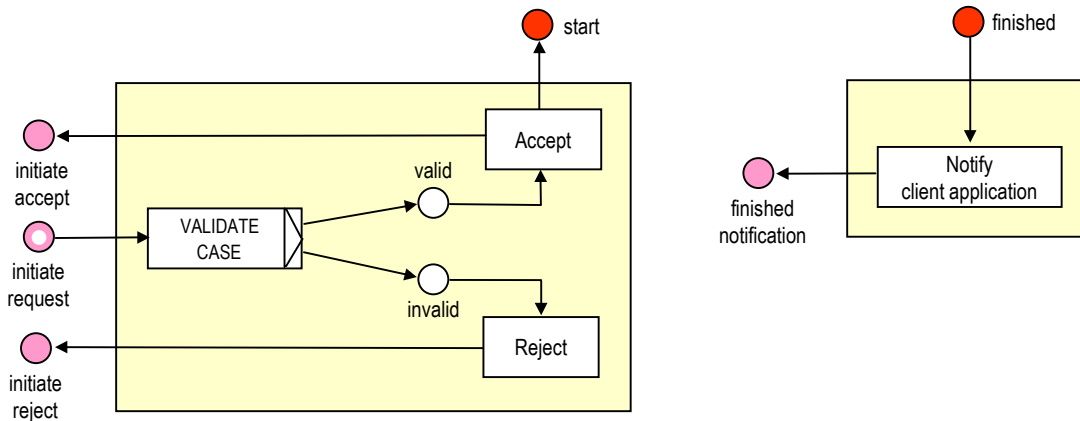


Figure 182 Behaviour of the Workflow Manager and the Interaction Manager (1)

• **Places ‘initiate’, ‘initiated’, ‘skipped’**

The Workflow Manager produces a token in place ‘*initiate*’ when the contracting process for a specific candidate service type must start. The model in Figure 183 shows that after a token is consumed from place ‘*initiate*’ the Interaction Manager first executes the standard operation ‘*determine candidate service status*’. If the candidate service type is not required, a token is produced in place ‘*skipped*’. Otherwise, the Interaction Manager executes standard operations ‘*specify candidate service data*’ and ‘*determine available providers*’. Here after, the Interaction Manager produces a token in place ‘*initiated*’.

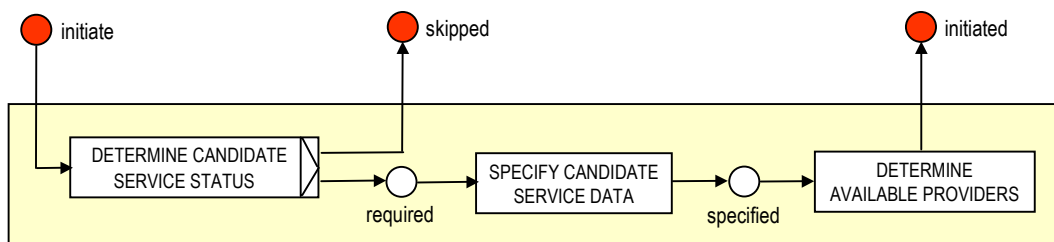


Figure 183 Behaviour of the Workflow Manager and the Interaction Manager (2)

- **Place ‘send’**

The model in Figure 184 shows that after the Workflow Manager produces a token in place ‘send’, the Interaction Manager consumes the token and queries the state data for the existence of the transaction. If the transaction does not exist, the standard operations ‘create transaction’ and ‘create outbound message’ are executed. Otherwise, only standard operation ‘create outbound message’ is executed.

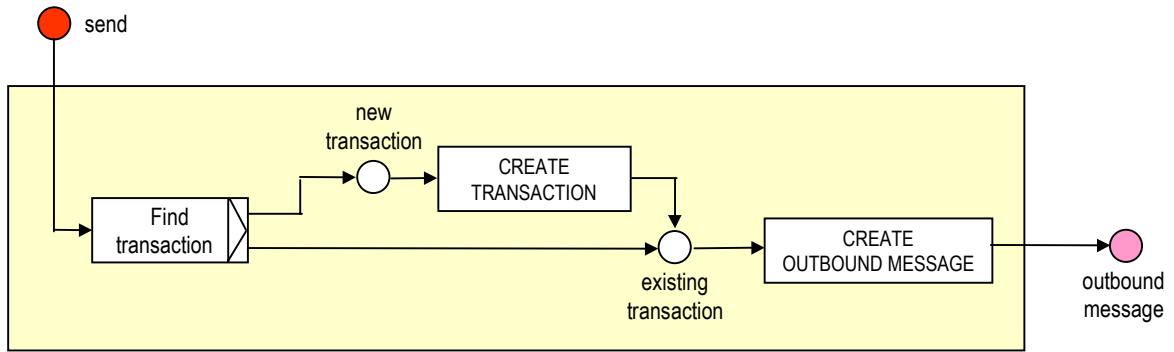


Figure 184 Behaviour of the Workflow Manager and the Interaction Manager (3)

- **Place ‘received’**

The model in Figure 185 shows that after the Interaction Manager consumes a token from place ‘inbound message’, the standard operation ‘store inbound message’ is executed, after which a token is produced in place ‘received’.

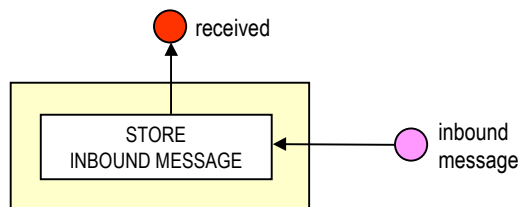


Figure 185 Behaviour of the Workflow Manager and the Interaction Manager (4)

- **Place ‘evaluate’, ‘evaluated’**

The model in Figure 186 shows that after the Workflow Manager produces a token in place ‘evaluate’ the Interaction Manager executes standard operation ‘determine message value’, after which a token with the result is produced in place ‘evaluated’.

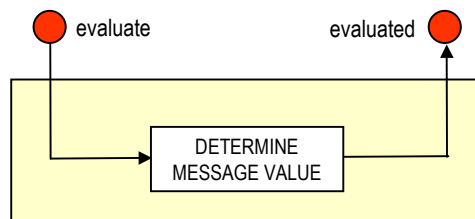


Figure 186 Behaviour of the Workflow Manager and the Interaction Manager (5)

- **Place ‘process’, ‘adjust’**

The model in Figure 187 shows that after the Workflow Manager produces a token in place ‘process’ or place ‘adjust’, the Interaction Manager executes standard operations ‘process inbound message’ / ‘check for violations’ or ‘adjust candidate service data’ respectively.

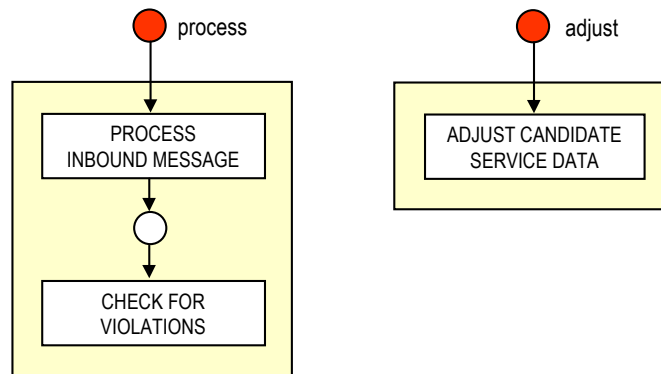


Figure 187 Behaviour of the Workflow Manager and the Interaction Manager (6)

Finally, we define the behaviour of the Interaction Manager on the configuration program interface and the monitoring program interface. We will define the behaviour in parts, each part for one input place or a group of input places.

- **Places belonging to the Configuration program interface**

The configuration program interface is used to update the configuration parameters of the Server component for which a simple ‘request / response’ protocol is used.

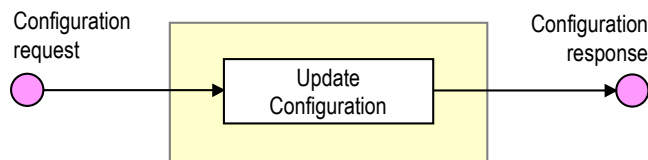


Figure 188 Behaviour of the ‘Interaction Manager’ component (1)

- **Places belonging to the Monitoring program interface.**

The monitoring program interface of the Interaction Manager is used for two functions: to retrieve a list of business cases and to retrieve the state data of one specific business case. For both functions a simple ‘request / response’ protocol is used.

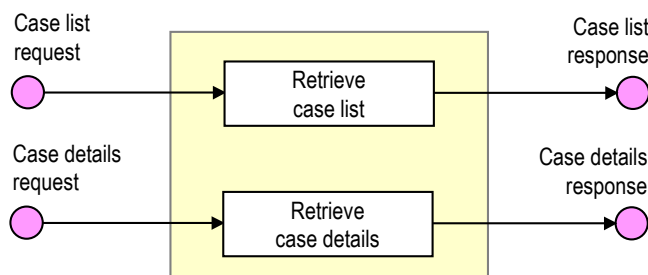


Figure 189 Behaviour of the ‘Interaction Manager’ component (2)

3.4 Architecture of the ‘Configurator’ component

3.4.1 Distribution of functionality over components

The Configurator component consists of the following sub components as is illustrated in Figure 190.

- **Available services**

This data store contains the specification of available service types and service providers, as well as protocol patterns and contracting strategies. The structure of the store is defined in Figure 192.

- **Contracting requirements**

This data store contains the specification of case types and the contracting requirements for cases belonging to these case types in terms of candidate service types, selected providers, selected violations, strategies, etc. The structure of the store is defined in Figure 193.

- **Configuration manager**

The Configuration Manager maintains the persistent data in the two data stores described before. It provides a user interface that supports the user in creating and modifying these configuration parameters. For this purpose, the user can import service types, service providers, protocol patterns and strategy types from a Repository. The creation and modification of attributes for which an extensible number of syntaxes must be supported is not part of the functionality of the Configuration Manager. Instead, if such a parameter must be created or modified, the Configuration Manager invokes another component dedicated for that task. This architectural decision allows us to add support of a new syntax by replacing that sub-component or by adding a new sub-component. Finally, the Configuration Manager allows the user to configure the Server component via the Configuration program interface of the Server component.

- **Schema editor**

The Schema editor provides a syntax-aware user interface that supports the user in creating and modifying the ‘*schema*’ attributes of the ‘CASE TYPE’ and ‘SERVICE TYPE’ entities in the persistent data. It implements one or more schema syntaxes for hierarchic data models.

- **Schema subset editor**

The Schema subset editor provides a syntax-aware user interface that supports the user in creating and modifying the ‘*schema*’ attributes of the ‘MESSAGE TYPE’ entities in the persistent data, based on the ‘*schema*’ attribute of a ‘SERVICE TYPE’ entity in the persistent data. It implements one or more schema syntaxes for hierarchic data models.

- **Constraints editor**

The Constraints editor provides a syntax-aware user interface that supports the user in creating and modifying the ‘*constraints*’ attributes of the ‘AVAILABILITY’, ‘VIOLATION TYPE’, ‘CASE TYPE’ and ‘CANDIDATE SERVICE TYPE’ entities in the persistent data. It implements one or more constraint syntaxes.

- **Transformation editor**

The Transformation editor provides a syntax-aware user interface that supports the user in creating and modifying the ‘*specification rules*’ attribute of the ‘CANDIDATE SERVICE TYPE’ entities in the persistent data. It implements one or more transformation syntaxes.

- **Strategy editor**

The Strategy editor provides a user interface that supports the user in creating and modifying the ‘*expression*’ attribute in the ‘STRATEGY PARAMETER’ entities related to the ‘STRATEGY’ entity defined for a candidate service type. It implements one or more contracting strategies.

- **Workflow analyser**

The Workflow analyser is used to check whether a generated contracting workflow is a sound WF-net.

- **Workflow generator**

The Workflow generator generates the workflow definition enacted in the Workflow Manager of the Server component. Its input is the contracting requirements of an outsourced task and it has knowledge of the negotiation strategies, execution strategies and acceptance strategies referred to in the contracting requirements.

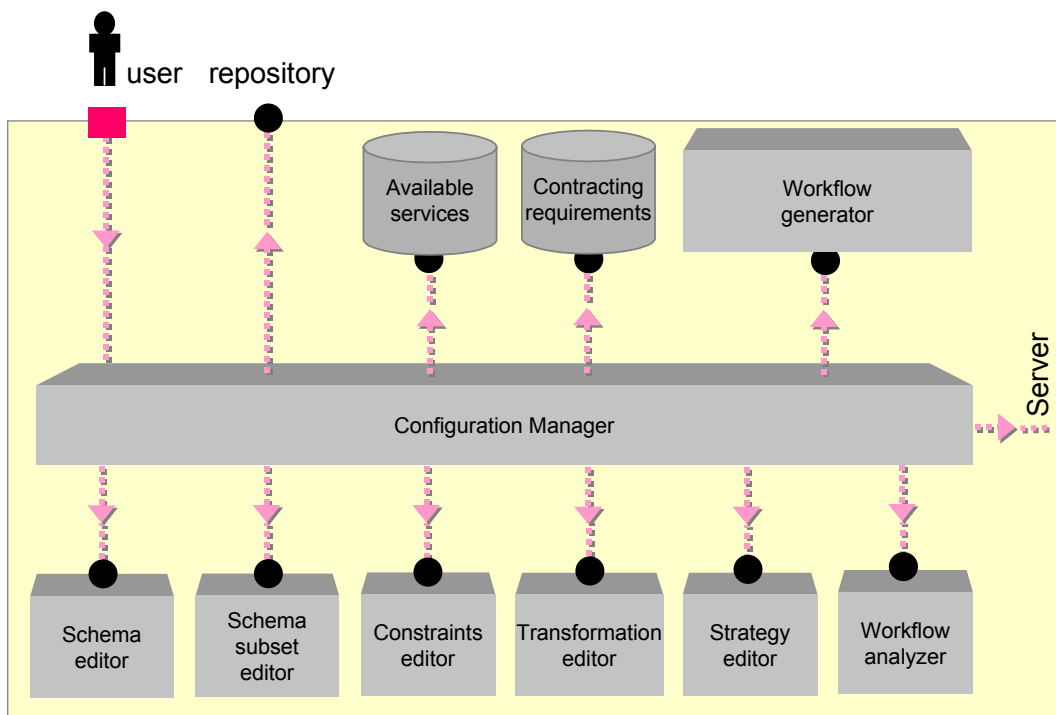


Figure 190 Main components and interfaces of the Configurator component

3.4.2 Structure of interfaces and persistent data

A formal model of the interfaces between the sub components of the Configurator component is given in Figure 191.

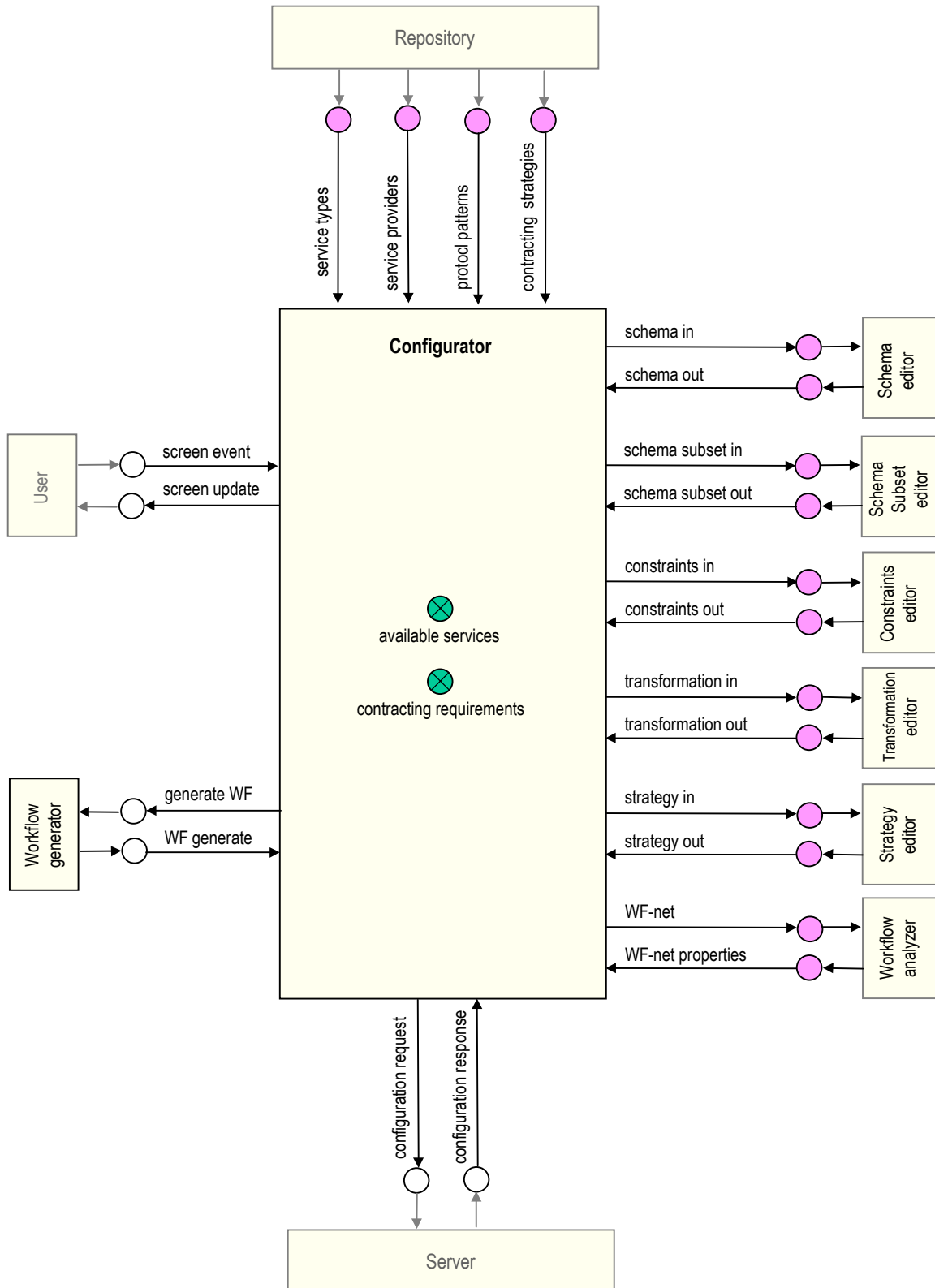


Figure 191 Interfaces between sub components in the Configurator component

Structure of persistent data

The stores '*available services*' and '*contracting requirements*' in the '*Configurator*' system model the persistent data of the Configurator component. The colour of the tokens in these stores is defined by the object models in Figure 192 and Figure 193 respectively.

The object model of the '*available services*' store maintained by the Configurator component is based on the available services complex class in Figure 46 with the following changes. First, in the implementation domain we use the term 'schema' instead of the term 'data model'. Second, since the logical architecture allows different syntaxes for schemas and constraints, we added a '*schema syntax*' attribute to each '*schema*' attribute and a '*constraints syntax*' attribute to each '*constraints*' attribute. Third, since configuration parameters often must be configured from scratch, only those attributes are required which are always filled when an object is instantiated. Finally, we added the entities 'STRATEGY TYPE' and 'PARAMETER TYPE' from Figure 65.

The object model of the '*contracting requirements*' store maintained by the Configurator component is based on the contracting requirements complex class in Figure 66 with the following changes. First, since the logical architecture allows different syntaxes for schemas, constraints and transformations, we added a '*schema syntax*' attribute to each '*schema*' attribute, a '*constraints syntax*' attribute to each '*constraints*' attribute and a '*specification rules syntax*' attribute to the '*specification rules*' attribute. Finally, since configuration parameters often must be configured from scratch, only those attributes are required which are always filled when an object is instantiated.

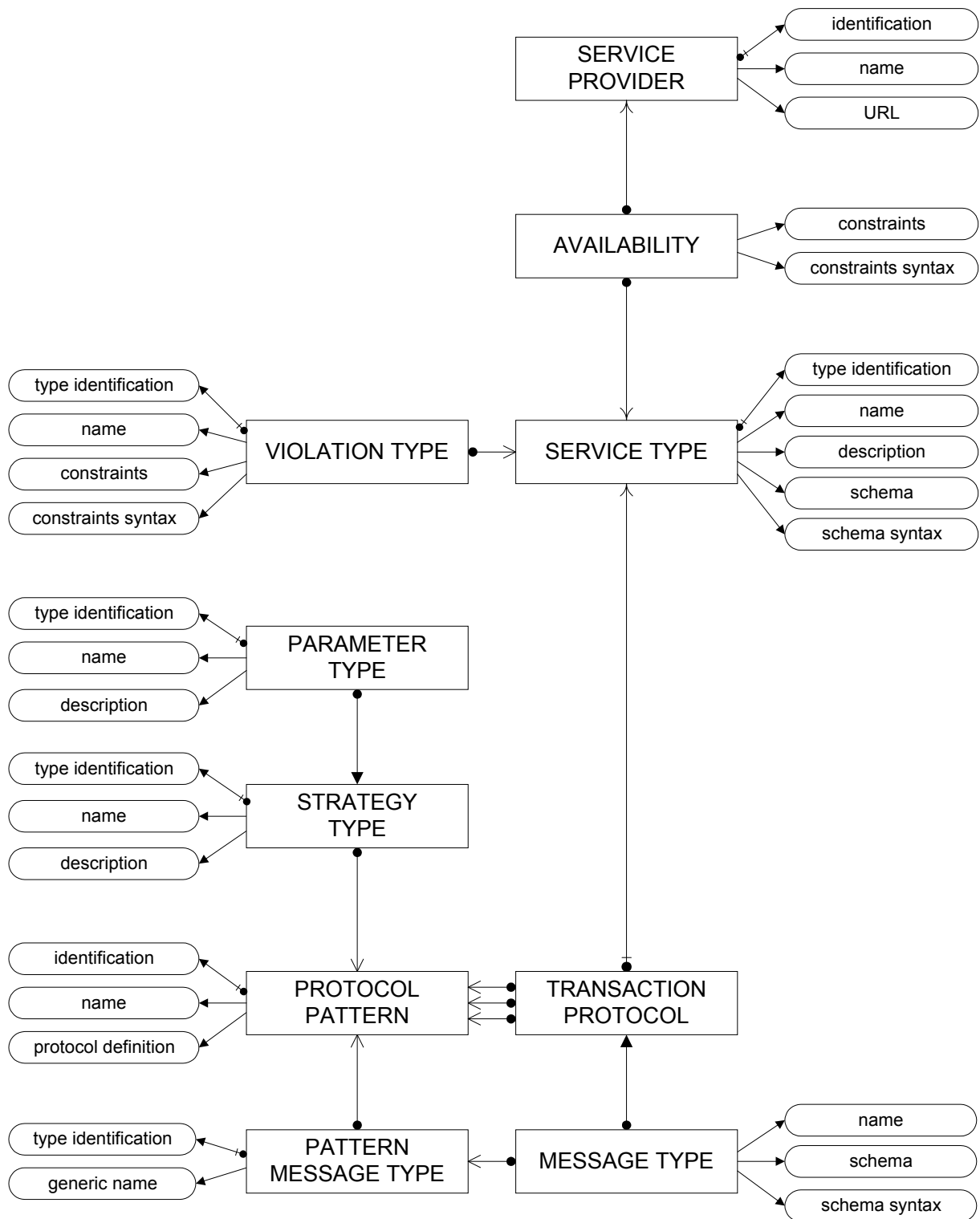


Figure 192 Structure of persistent data store 'available services' of the Configurator component

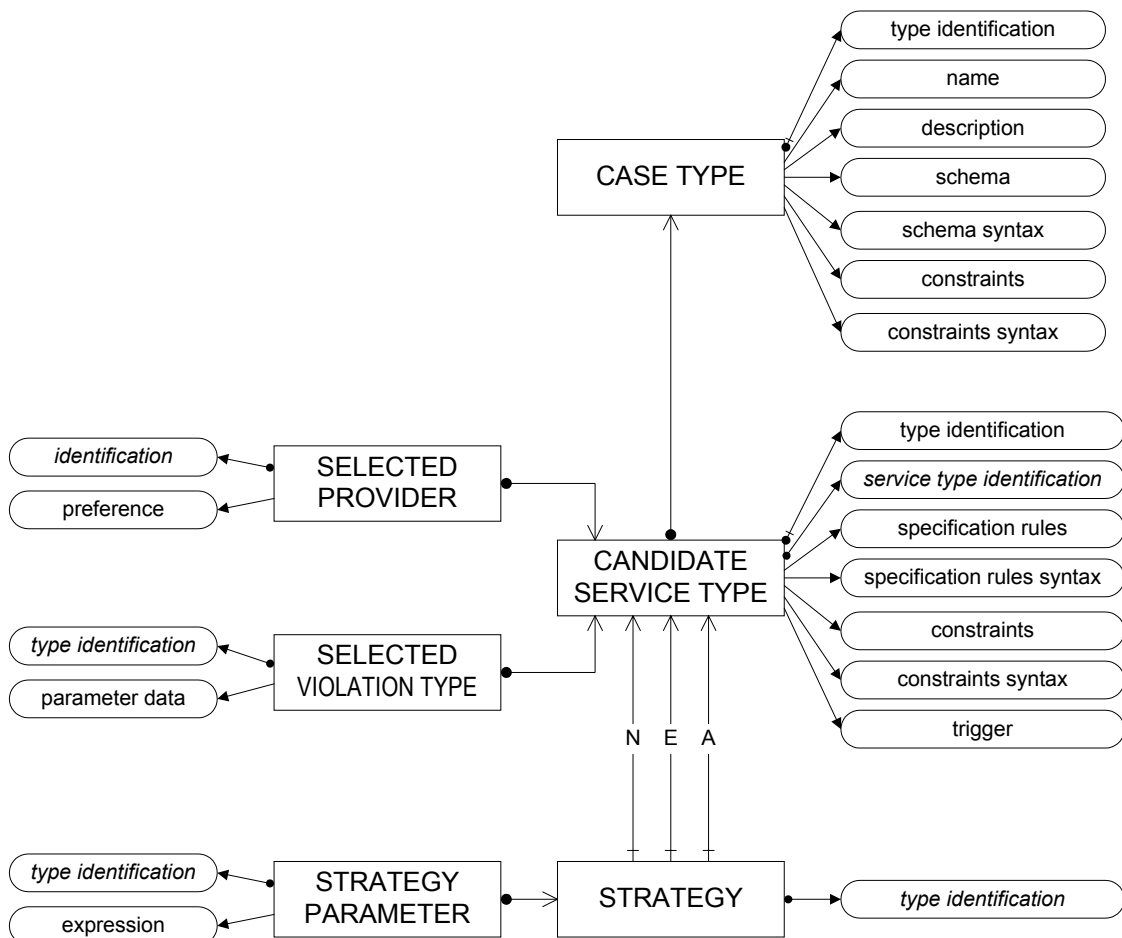


Figure 193 Structure of persistent data store 'contracting requirements' of the Configurator component

Structure of the Schema editor program interface

The Schema editor is a memory-less component that takes a schema as input, modifies it and produces the modified schema as output. The Schema editor supports one or more schema syntaxes.

	Place 'schema in', 'schema out'
Role	These places model the input and output of the Schema editor. The attributes that can be modified by the Schema editor are marked with a grey background.
Type	The colour a token in one of these places is a complex of which the object model is given in Figure 194. The 'schema' attribute models the schema to be maintained, whereas the syntax of this schema is given by the 'schema syntax' attribute.
	Figure 194 Object model for the colour of the 'schema in' and 'schema out' places

Structure of the Schema subset editor program interface

The Schema subset editor is a memory-less component that takes a template schema and one or more subset schemas as input, modifies the subset schemas and produces them as output. A Schema subset editor supports one or more schema syntaxes.

Place 'schema subset in', 'schema subset out'	
Role	These places model the input and output of the Schema subset editor. The attributes that can be modified by the Schema subset editor are marked with a grey background.
Type	<p>The colour of a token in one of these places is a complex of which the object model is given in Figure 195. The schema used as template on which the subset schemas are based is modelled by the 'TEMPLATE SCHEMA' entity. A schema that is a subset on the template schema is modelled by the 'SUBSET SCHEMA' entity. Because multiple subset schema's are allowed, a 'name' attribute is used in each 'SUBSET SCHEMA' entity to distinguish the subset schema's from each other.</p>
<p><i>Figure 195</i> Object model for the colour of the 'schema subset in' and 'schema subset out' places</p>	

Structure of the Constraints editor program interface

The Constraints editor is a memory-less component that takes a schema and a set of constraints as input, modifies the set of constraints and produces the modified set of constraints as output.

Place 'constraints in', 'constraints out'	
Role	These places model the input and output of the Constraints editor. The attributes that can be modified by the Constraints editor are marked with a grey background.
Type	<p>The colour of a token in one of these places is a complex of which the object model is given in Figure 196. The 'constraints' attribute models the set of constraints to be modified (can be empty), whereas the syntax of the constraints is given by the 'constraints syntax' attribute. Since constraints are meant to operate on hierarchic data with a specific structure, the 'schema' attribute models the structure of the hierarchic data on which the constraints operate. The 'schema syntax' attribute models the syntax used for the schema.</p>
<p><i>Figure 196</i> Object model for the colour of the 'constraints in' and 'constraints out' places</p>	

Structure of the Transformation editor program interface

The Transformation editor is a memory-less component that takes two schemas and a transformation as input, modifies the transformation, and produces the modified transformation as output.

	<p>Place ‘transformation in’, ‘transformation out’</p>
Role	<p>These places model the input and output of the Transformation editor. The attributes that can be modified by the Transformation editor are marked with a grey background.</p>
Type	<p>The colour of a token in one of these places is a complex of which the object model is given in Figure 197. The ‘<i>transformation</i>’ attribute models the transformation function to be modified (can be empty), whereas the syntax of the transformation function is given by the ‘<i>transformation syntax</i>’ attribute. Since transformation functions are meant to operate on a source data set with a specific structure and generates a target data set with a specific structure, the ‘<i>source schema</i>’ attribute models the structure of the source data set and the ‘<i>target schema</i>’ attribute models the structure of the target data set. The ‘<i>target schema syntax</i>’ attribute and the ‘<i>target schema syntax</i>’ attribute model the syntax used for the source schema and target schema respectively.</p> <div data-bbox="459 1048 1217 1384" style="text-align: center;"> <pre> graph LR TRANSFORMATION[TRANSFORMATION] --> source_schema[source schema] TRANSFORMATION --> source_schema_syntax[source schema syntax] TRANSFORMATION --> target_schema[target schema] TRANSFORMATION --> target_schema_syntax[target schema syntax] TRANSFORMATION --> transformation[transformation] TRANSFORMATION --> transformation_syntax[transformation syntax] style transformation fill:#ccc </pre> </div> <p><i>Figure 197</i> Object model for the colour of the ‘transformation in’ and ‘transformation out’ places</p>

Structure of the Strategy editor program interface

A strategy defined for a specific candidate service type can require parameter data, e.g. an expression to evaluate offers. The Strategy editor is a memory-less component that takes the identification of a contracting strategy, a service type schema and the parameter data for the selected strategy as input. It lets the user modify the parameter data and produces the modified parameter data as output.

	<p>Place ‘strategy in’, ‘strategy out’</p>
<p>Role</p>	<p>These places model the input and output of the Strategy editor. The attributes that can be modified by the Strategy editor are marked with a grey background.</p>
<p>Type</p>	<p>The colour of a token in one of these places is a complex of which the object model is given in Figure 198. The ‘<i>type identification</i>’ attribute of the ‘STRATEGY’ entity models the unique identification of the strategy type and is a reference to a ‘STRATEGY TYPE’ entity. Since a strategy is meant to operate on service data (and message data) with a specific structure, the ‘<i>service type schema</i>’ attribute models the structure of the service data and the ‘<i>service type schema syntax</i>’ attribute models the syntax used for the service type schema. Each ‘STRATEGY PARAMETER’ entity models a specific parameter used in the contracting strategy. The ‘<i>type identification</i>’ attribute models the parameter type and is a reference to a ‘PARAMETER TYPE’ entity. The ‘<i>data</i>’ attribute of the ‘STRATEGY PARAMETER’ entity models the parameter data to be modified (can be empty).</p> <div data-bbox="475 1153 1197 1478" data-label="Diagram"> <pre> classDiagram class STRATEGY { type identification service type schema service type schema syntax } class STRATEGY_PARAMETER { type identification data } STRATEGY_PARAMETER --> STRATEGY </pre> </div> <p><i>Figure 198</i> Object model for the colour of the ‘strategy in’ and ‘strategy out’ place</p>

Structure of the Workflow generator program interface

The Workflow generator takes the contracting requirements as input and generates the corresponding implementation contracting workflow in the format expected by the Workflow Manager. The Workflow generator has knowledge of contracting strategies and their implementation in high level coloured Petri nets.

Role Type	<p>Place 'generate WF'</p>
	<p>A token in this place models the input to the Workflow generator and contains the part of the contracting requirements relevant to the generation of the contracting workflow.</p> <p>The colour of a token in this place is a complex of which the object model is given in Figure 199. The entities other than entity 'GENERATE WORKFLOW' are taken from the contracting requirements object model in Figure 193. The entity 'GENERATE WORKFLOW' models the request and contains one attribute 'WF syntax' that defines the syntax in which the generated workflow definition must be expressed.</p> <p style="text-align: center;">Figure 199 Object model for the colour of the 'generate WF' place</p>

Role Type	<p>Place 'WF generated'</p>
	<p>A token in this place models the output of the Workflow generator.</p> <p>The colour of a token in this place is a complex of which the object model is given in Figure 200. If the workflow was successfully generated, no 'ERROR' entities are present and the 'workflow definition' attribute contains the entire definition of the generated workflow. Otherwise, the 'workflow definition' attribute is empty and one or more 'ERROR' entities are present.</p> <p style="text-align: center;">Figure 200 Object model for the colour of the 'WF generated' place</p>

3.4.3 Behaviour on the component interfaces

The sub-components invoked by the Configuration manager component have a simple request / response protocol as defined in Figure 201.

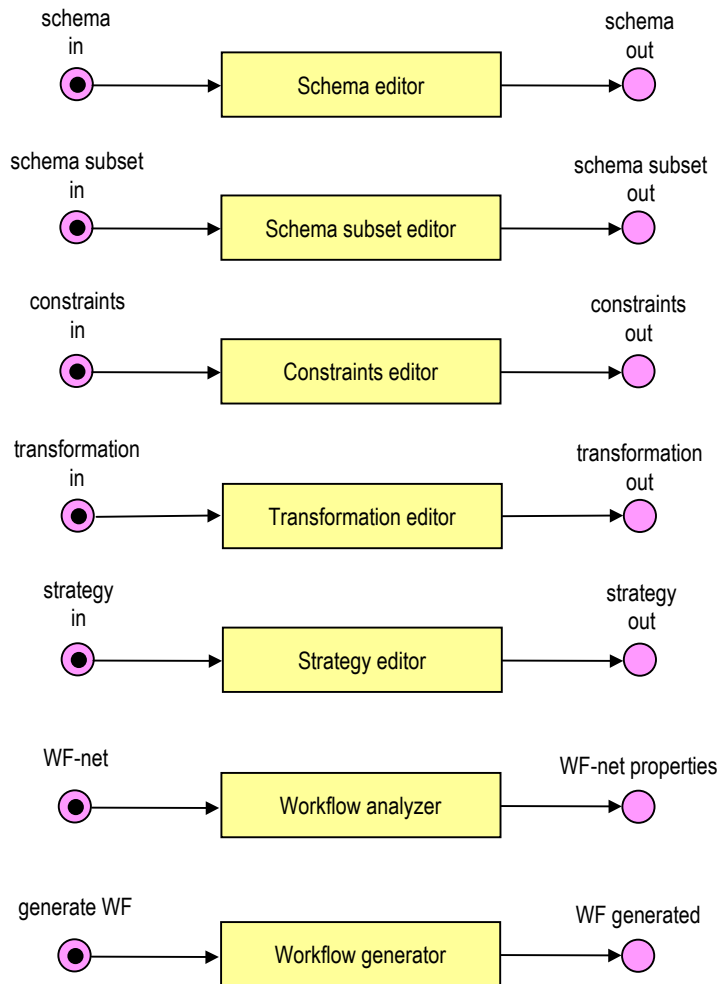


Figure 201 Behaviour of the components invoked by the Configuration manager

4. Technical architecture of the Contracting Agent

This chapter describes the technical architecture of a software component that implements the logical architecture described in the previous chapter. The objectives of this component in the context of the research are given in Section 4.1.1. Basic construction choices like platform, programming system, the use of standards and standard components are discussed in section 4.1.2 – 4.1.7. There after, Section 4.2 gives the technical software architecture of the Contracting Agent by specifying all individual software components and their interfaces. A specification of the custom-made components is given in sections 4.3 and 4.4. An important part of the system is the generated contracting workflow which is enacted by the workflow engine in the Server component. Examples from a generated contracting workflow are given in Section 4.5. Finally, Section 4.6 gives an impression of the use of the configuration and monitoring functions of the software by presenting examples of the user interface.

4.1 Basic construction choices

4.1.1 Objectives

The main objective for constructing a Contracting Agent is to provide a *proof of concept* for the conceptual framework (Chapter 2) and the logical architecture (Chapter 3). The consequence of this objective is an emphasis on *functionality*, rather than on aspects like performance, maintainability, multi-platform, scalability, etc. We will therefore choose technologies that allow us to implement the desired functionality efficiently. The second objective for constructing a Contracting Agent prototype is to show how construction efforts can be minimised by using available standards and components. This objective is derived from a research question formulated in Chapter 1: *‘What is a suitable technical architecture of the service contracting software component that utilises existing standards and existing components maximally in order to minimise the effort to develop the component and to maximise maintainability’*.

4.1.2 Windows and COM

Due to the focus on functionality rather than aspects like performance, multi-platform and scalability we have made the following choices for the platform under which the Contracting Agent will be developed.

- **Windows as operating system**

The Contracting Agent will be developed for the Microsoft Windows platform. Windows is a platform for which a large number of software development tools is available and has a large installed base. Furthermore, the choice for Windows allows us to use standard software components available for that platform, such as MSXML4, ExSpec, WOFLAN and MSOutlook.

- **COM as component framework**

The Contracting Agent will not be implemented as a monolithic piece of software, but will consist of several autonomous software components. In order to be assembled seamlessly with other software components, a software component must be based on a component framework as COM, CORBA or JavaBeans. We have chosen to use the COM framework for the Contracting Agent prototype, because it gives the best fit with the Windows platform, is well supported by popular programming systems like Visual Basic or Visual C++ and does not require any additional components like middleware.

- **Visual Basic as programming system**

The custom-made software components will be developed in Visual Basic. This system is known for its development speed and productivity, especially for graphical user interfaces and data-centric applications on the desktop. The ActiveX technology allows a programmer to compose a Graphic User Interface (GUI) from a wide range of standard ActiveX controls. Another powerful aspect of Visual Basic is its facilities to create a broad variety of COM-based components, e.g. in-process Dynamic Link Libraries (DLL) and out-of-process COM-servers.

The COM framework

The COM standard is an object-based programming model designed for interoperability between binary software components that can be developed in different software languages. COM defines and implements mechanisms that allow applications to connect to each other as *software objects*, an instantiation of a class that conforms to the COM standard. A COM object is accessed only via its *interfaces*, a set of strongly typed semantically related functions (called member functions of that interface). An interface can be defined by the Interface Description Language (IDL) and is implemented by one or more software objects. If a software object implements an interface, it has to implement each of the member functions in that interface. Each interface in each software object has a 128-bit integer Globally Unique Identifier (GUID). A software object can implement one or more interfaces simultaneously. If a vendor creates a new version of a software object, he does not change the existing interfaces, but simply adds new interfaces if necessary. When a client application has access to a software object, it has nothing more than a pointer through which it can access the functions in the interface, called the *interface pointer*. Figure 202 shows the drawing technique for software objects. The interfaces of an object are drawn as a small circle connected to the object via a line. An interface pointer of a client application to an interface is drawn as an arrow.

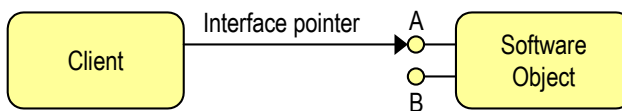


Figure 202 COM objects accessed via interfaces

When a client application initially gains access to an object, it is given one interface pointer. The COM standard defines a mechanism by which a client can navigate through the interfaces of an object. This mechanism relies on the function `QueryInterface` that must be implemented in each interface. The client calls `QueryInterface` with the unique identifier of an interface as parameter. If the object implemented the interface, it returns a pointer to the interface. Otherwise, it returns the null value and the client application will not be able to use the interface.

Components based on COM can be distinguished into two categories with respect to the client that is using the component: *in-process* and *out-of-process*. An in-process component runs in the same process and address space as the client that uses the component. Out-of-process components run in a separate process and address space (or even on a different machine). However, COM is designed to allow clients to transparently communicate with objects, regardless of where those objects are running, be it in the same process, on the same machine or on a remote machine. This functionality is provided by the COM run-time library, which is part of the Windows operating system. If a client calls a function in an interface of an in-process object, the call reaches it directly as illustrated in Figure 203.

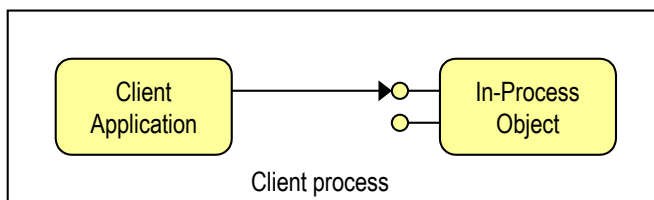


Figure 203 Communication between COM objects in the same process

If the client and server object are not in the same process, the call is intercepted by an in-process 'proxy' object provided by COM. The 'proxy' object generates the appropriate Remote Procedure Call (RPC) to a 'stub' object provided by COM that runs in the same process as the server object. The 'stub' object receives the Remote Procedure Call and makes an interface call to the server object. Communication between objects in different processes is illustrated in Figure 204.

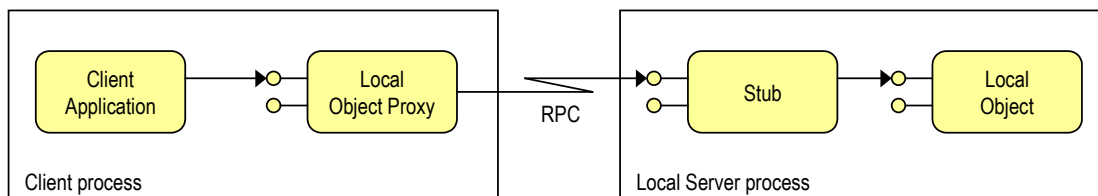


Figure 204 Communication between COM objects in different process

Microsoft extended the COM technology to support distributed computing with the DCOM (Distributed COM) standard [95]. Distributed applications consist of components that reside on different hardware components, connected by a network. The advantage of distributed applications is the possibility to run a component on the platform that is best suited, and the scalabil-

ity. One could start with a single server running all components and deploy additional machines to run one or more components when the load increases.

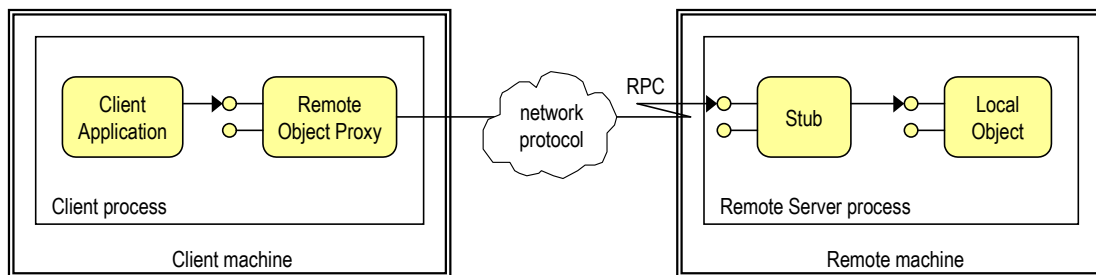


Figure 205 COM components on different machines

There are two types of in-process components: *ActiveX controls* and *Dynamic Link Libraries*. An ActiveX control (with file extension OCX) is a reusable component and is always used in a container, for instance a form in a Visual Basic application or an HTML document. The most well known examples of ActiveX controls are the controls to create user-interfaces from. A number of standard ActiveX controls are distributed with development tools like Visual Basic and Visual C++. In addition to this, a market for ActiveX controls from third-party suppliers emerged. The second type of in-process components are Dynamic Link Libraries (with file extension DLL). A DLL is used to store reusable class libraries, from which client applications can create instances of objects. The object exists in the same address space and process space as the application that created the object. DLL's are ideal for generic functions used by different applications, such as business rules. They are fast to load, and have a high rate of data transfer with the client application because they occupy the same address space.

Out-of-process servers run in a separate process and address space. Because they run in a separate thread of execution, they are ideal for performing background tasks or asynchronous tasks. A client application can invoke a method in an out-of-process server and continue with its own processing. As soon as the out-of-process server completed the task, it raises an event for the client to signal the completion of the task. A disadvantage of out-of-process servers is the slower communication between client and server object because data must be transferred from one address space to another. Out-of-process servers are used when the server must run in a separate thread of execution or when the server can run as a stand-alone application too.

4.1.3 XML, XML-Schema, XSLT, DOM, MSXML4

We have defined case data, service data and message data as hierarchic data structures on which different functions must be performed: validation, transformation, parsing and presentation. Furthermore, case data, service data and message data must also be stored and exchanged. Programming this functionality from scratch (as generic functions) would be a complex and time-consuming effort. XML standards and standard components that support these standards will therefore be used.

- **Storage and exchange: XML documents**

Case data, service data and message data are modelled as XML documents that are part of the persistent data of the Server component. By using the XML standard, we can make use of many additional standards for operations on the XML documents. Furthermore, XML documents can be easily exchanged between software components in the form of a text file

or via a string parameter in a method on a COM component. The ‘*data*’ attributes in the functional data models in Chapter 3 are therefore implemented as XML documents.

- **Parsing: DOM with MSXML4**

When the value of one or more elements or attributes must be retrieved from an XML document, we require a parsing mechanism. The Document Object Model (DOM) is a specification of a standard API for manipulating XML documents. A COM-component that implements the DOM specification is the Microsoft XML parser (MSXML4.DLL). The XML parser can be used to instantiate an object in which the XML document is loaded whereafter it exposes the content of the XML document via a set of methods and properties.

- **Schema validation: XML-Schema with MSXML4**

Validation is checking a hierarchic data set against a schema. The validation function is offered by validating XML parsers such as MSXML4. These parsers take a DTD or a schema of the XML document as input, together with the XML document itself. Although a DTD can be used for validation purposes, its expressive power is not enough to check the structural elements defined by the data model in Figure 47. A standardised schema language with enough expressive power is XML-Schema, which is supported by the MSXML4 validating parser. The ‘*schema*’ attributes in the functional data models in Chapter 3 are therefore implemented as XML-Schema documents.

- **Transformation: XSLT with MSXML4**

Transformation is performed on case data and service data in the specification phase of a service contracting process in order to specify the service data of a candidate service. We will use an XML document to represent the case data and service data which is input for the transformation. Furthermore, we will use the XSLT standard to define the transformation that takes the XML document with case data and service data as input and creates the XML document that contains the service data of the candidate service as output. Therefore, the ‘*specification rules*’ attribute in the ‘*contracting requirements*’ store (see Figure 66) is implemented as an XSLT document. Furthermore, we will use MSXML4 as XSL(T) processor.

- **Constraints checking: XSL/XPath with MSXML4**

The Contracting Agent for which we define the technical architecture implements only simple constraints, for which we will use the XSL standard as representation syntax. Operands in the constraints are represented as an XPath expression. Furthermore, we will use MSXML4 as XSL and XPath processor.

- **Presentation: XSL with Internet Explorer**

Presentation of hierarchic data is required in the Monitor component where case data, service data and message data can be viewed by the user. We will use the XSL standard to define the layout of the case data, service data and message data in HTML structure. We will use Internet Explorer as XSL processor and HTML viewer for presentation of case data, service data and message data.

4.1.4 Relational databases, ADO and Access 2000 Jet engine

The Configurator component and the Server component rely on persistent data heavily. As we have seen before, some attributes of the functional data models that represent persistent data stores are implemented as XML documents. For example, the ‘*data*’ attributes of the ‘*CASE*’, ‘*SERVICE*’ and ‘*MESSAGE*’ entities are implemented as XML documents. The ‘*schema*’ attrib-

utes of the 'CASE TYPE', 'SERVICE TYPE' and 'MESSAGE TYPE' entities are implemented as XML-Schema documents, which are XML documents too. Finally, other attributes like the '*specification rules*' attribute of the 'CANDIDATE SERVICE TYPE' entity, implemented as XSLT document, are also XML documents.

In general, we have two options for storing the persistent data: in a relational database or as an XML document. The main advantages of a relational database are the use of SQL as query language, support of multi-user access and the ability to handle large volumes of data and large numbers of database transactions efficiently. Because an XML document does not provide this functionality, we choose to use a relational database to store persistent data. The database schema of the relational databases is derived from the functional data models by implementing an entity as a table and an attribute of an entity as a column in the table. The choice for a relational database leaves us with the question how to store XML documents, for which two options are available:

- **Data-centric**

If the XML document has a well-defined structure, it is possible to map the structure of the XML document to one or more relational tables. When the XML document is stored, its individual elements and attributes are distributed among columns in the relational tables. Once stored in relational tables, the information can be updated and queried by using SQL. If necessary, the information from the relational tables can be retrieved and formatted into an XML document again.

- **Document-centric**

If the XML document does not have a well-defined structure, or if updates on the XML document are always made by replacing the entire XML document, a document-centric approach can be followed. In this approach, the XML document is stored as a BLOB (binary large object) in a column of a relational table. The advantage of storing intact XML documents in a relational databases is the reliability of a relational database system over file system storage.

We will choose a document-centric approach for storing the XML documents in the persistent data of the Configurator and Server component because they are stored, retrieved and updated as entire XML documents only. Finally, we must choose a database management system (DBMS) and an mechanism to access the relational database from Visual Basic programs.

- **DMBS / access mechanism: ADO / Access 2000**

We will use the Jet 4.0 engine of the Microsoft Access 2000 relational database system. Although the Jet engine does not provide the same level of scalability and robustness as for instance SQL Server offers, its ease-of-use makes it a good choice from the perspective of the objective to provide a proof of concept. The relational database is accessed from the Visual Basic programming environment by ActiveX Data Objects (ADO) which can be used from any COM compliant programming language and which supports a broad range of data sources. Furthermore, since we use ADO to access the database engine, we can easily replace the Jet engine by another database engine.

4.1.5 ExSpect as workflow engine

We will use the ExSpect engine as commercial-of-the-shelf component to implement the functionality of the Workflow Manager. ExSpect is a powerful process-modelling tool, based on the formalism of high-level coloured Petri nets. The components of which the ExSpect tool consists are shown in Figure 206:

- **Design interface**

The Design interface provides a graphical user interface by which a user can define process models in the form of high level coloured Petri nets.

- **Model definition file**

An ExSpect model is stored in a flat file according to the ExSpect notation for high-level coloured Petri nets. The extension ‘.ex’ is used for these files. A model definition file always contains the structure of a Petri net and optionally contains information on the graphical representation in the Designer user interface (size, position, colour).

- **Engine**

The Engine is the part of ExSpect that executes high level coloured Petri nets. It reads a model definition from a ‘.ex’ model definition file and computes the firing sequence.

- **Dashboard**

The Dashboard is the part of the ExSpect simulation environment in which the user can install one or more graphical objects to represent the state of the simulation process. For this purpose, graphical objects in the dashboard can be connected to places in the Petri net executed by the Engine.

- **Animator**

The Animator is the part of the ExSpect simulation environment that displays the structure of the Petri net under execution. The user can see the distribution of tokens over the places. When a transition fires, the user can see tokens being consumed and produced.

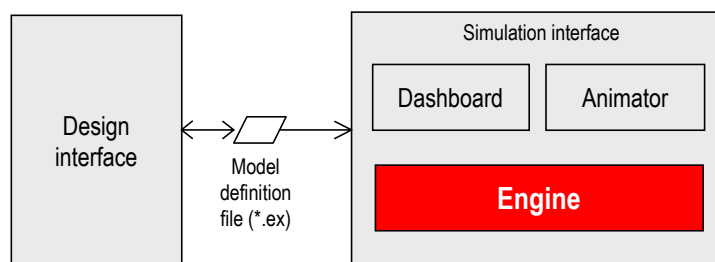


Figure 206 Components of ExSpect and position of the Engine component

The ExSpect engine has been isolated from the other parts of ExSpect and is housed in a separate COM-component. This allows the ExSpect engine to be embedded in other software products as an out-of-process COM-server. The ExSpect engine then acts as a workflow engine, handling the control flow of the application. We will now describe in plain English the use of the API as used in the Contracting Agent.

- **Start the execution**

To start the execution of a Petri net, the client has to call the 'init' method. Parameters of this method are the path and file name of the model definition file and the name of the system at the top of the system hierarchy. When the model is initialised, a 'tracePlace' method is applied for all places that model signals from Workflow Manager to Interaction Manager.

```
Set oExServer = CreateObject("ExServObj.ExServObj.1")
Set oEvents = oExServer

oExServer.Init("c:\ContractingAgent", "contracting", "main")

varRet = oExServer.TracePlace("IM_initiate")
varRet = oExServer.TracePlace("IM_send")
varRet = oExServer.TracePlace("IM_process")
varRet = oExServer.TracePlace("IM_evaluate")
varRet = oExServer.TracePlace("IM_adjust")
varRet = oExServer.TracePlace("IM_finished")
```

- **Produce tokens**

At the start of a service contracting process, a token must be produced in the 'start' place. During a service contracting process, tokens are produced in the places 'initiated', 'skipped', 'received' and 'evaluated'. To produce a token in the Petri net executed by the ExSpect engine, the 'produceToken' method is applied with three parameters: the name of the place, the value of the token and the delay, for example:

```
strValue = "[caseid:'" & pstrCaseId & "',caseowner:'" & . . . .
varRet = oExServer.ProduceToken("IM_start", strValue, 0)
varRet = oExServer.Continue
```

- **Consume tokens**

When a token is produced in a place for which a 'tracePlace' method was applied, the ExSpect engine raises a 'produce' event for the client. The event has three parameters: the name of the place, a unique identification of the token and the time stamp of the token. The client then uses the 'getTokenValue' method with this identification as parameter to retrieve the token value. Finally, the client applies the 'consumeToken' method with the identification as parameter to delete the token from the place.

```
Private Sub oEvents_Produce( ByVal TokenID As Long,
                             ByVal Place As String,
                             ByVal Time As Double)
. . . . .
strValue = oExServer.GetTokenValue(TokenID)
varRet = oExServer.ConsumeToken(TokenID)
```

4.1.6 WOFLAN 2.0 as Workflow net analyser

The Woflan tool is used to verify soundness of the contracting workflow, derived from the triggers for candidate service types defined by the user. If the contracting workflow is not sound, the user has to modify his triggering mechanisms in order to correct the error. Woflan reads a file with a classic Petri net definition as input. Thereafter, it applies standard analysis

techniques to it to determine whether the Petri net is a sound workflow net. If errors are found, Woflan guides the user towards finding the causes of these errors. The Woflan tool contains a number of modules: a GUI module, an analysis module for loading, verifying and diagnosing process definitions, and three conversion modules for process definitions from commercial products. An example of a small Woflan input file is given in Figure 207.

```
place "start";
place "started";
place "end";

trans "A"
  in "start"
  out "started";

trans "B1"
  in "started"
  out "end";

trans "B2"
  in "started"
  out "end";
```

Figure 207 Example of a Woflan input file

4.1.7 MS Outlook as message exchange component

The Contracting Agent uses one or more message exchange components for inter-organisational message exchange. Clearly, a commercial component for service contracting must have adapters for the major standard message exchange components available on the market. Because the emphasis of our prototype is on the functionality of the service contracting process instead of the functionality of the message exchange, we have chosen for Microsoft Outlook as lightweight and low-cost message exchange component. Microsoft Outlook is a desktop information system providing email capabilities and personal management functions to organise contacts, appointments and tasks. Besides a graphical user interface, Outlook also uses COM to expose an object model by which Outlook functionality can be integrated into other applications. We will use this feature to add message exchange functionality to the Contracting Agent.

4.2 Application architecture

The application architecture in Figure 209 defines all software components of which the Contracting Agent consists, together with all software components that interface with the Contracting Agent. The components that are commercial off-the-shelf (COTS) products are marked by a dark grey background. The other components are custom made. A connector between two software components represents an interaction between these components. The following three connector types are used to represent types of interaction.

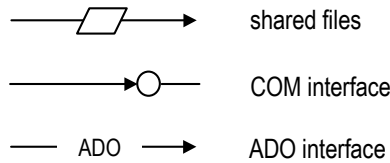


Figure 208 Legend for types of interfaces between components

Two components A and B have a *file sharing* interface when component A creates a file and component B reads the file. In this model, there is no direct communication between the software components A and B. All communication is done via one or more shared files. Two components A and B have a *COM interface* when both components use the COM framework to communicate. In the drawing technique, component A has the role of COM client, whereas component B has the role of COM server.

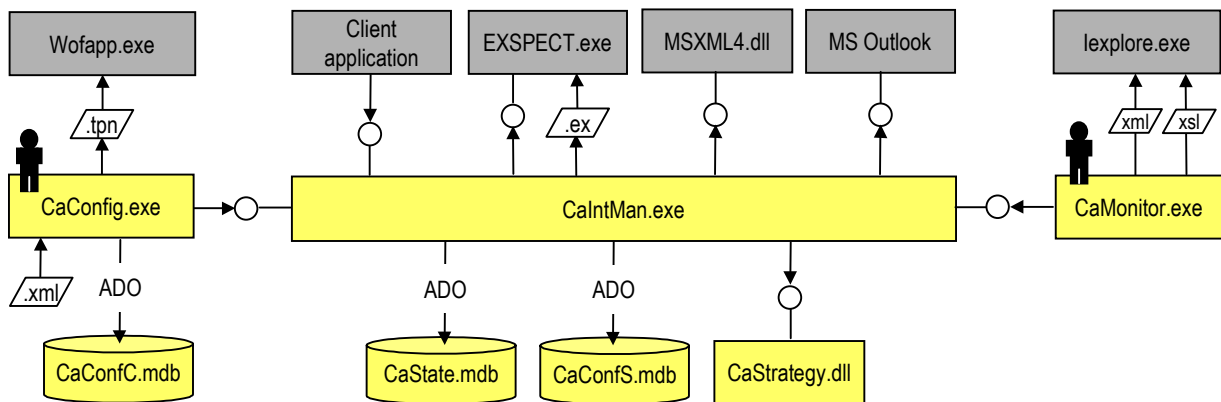


Figure 209 Application architecture of the Contracting Agent

Component name	Function
CaConfig.exe	Stand alone executable program which provides a graphical user interface to maintain configuration parameters used by the Server component
CaIntMan.exe	Out-of-process COM server which is the glue between all subcomponents of the Server component.
CaMonitor.exe	Stand alone executable program that provides a graphical user interface by which the user can inspect the state data of the Server component.
CaStrategy.dll	In-process COM server that executes the strategy depending tasks during a service contracting process (evaluation and adjustment).

Component name	Function
CaConfC.mdb	Relational database to store the persistent data of the Configurator component.
CaState.mdb	Relational database to store the persistent state data of the Server component.
CaConfS.mdb	Relational database to store the persistent configuration data of the Server component.
Component name	Function
ExSpect.exe	Out-of-process COM server that provides workflow engine functionality.
Wofapp.exe	Stand alone executable program that reads a flat file with the definition of a workflow net and analyses the properties of that workflow net (e.g. soundness).
Iexplore.exe	Stand alone executable program that displays the content of an XML document formatted according to an XSL document.
MSXML4.dll	In-process COM server that combines the functionality of a validating XML parser, implementation of the DOM and XSL(T) processor.
Outlook.exe	Out-of-process COM server that provides message exchange functionality to the Server component.

Finally, we give the mapping of components in the logical architecture presented in Chapter 3 to the components in the technical architecture presented in Figure 209. As can be seen, seven logical sub-components of the Configuration component are implemented in one technical component. This choice has been made to implement the configuration functionality efficiently. However, a commercial software product would require at least the logical Workflow Generator component to be housed in a separate technical component, in order to make it possible to support new contracting strategies by replacing that single sub-component only.

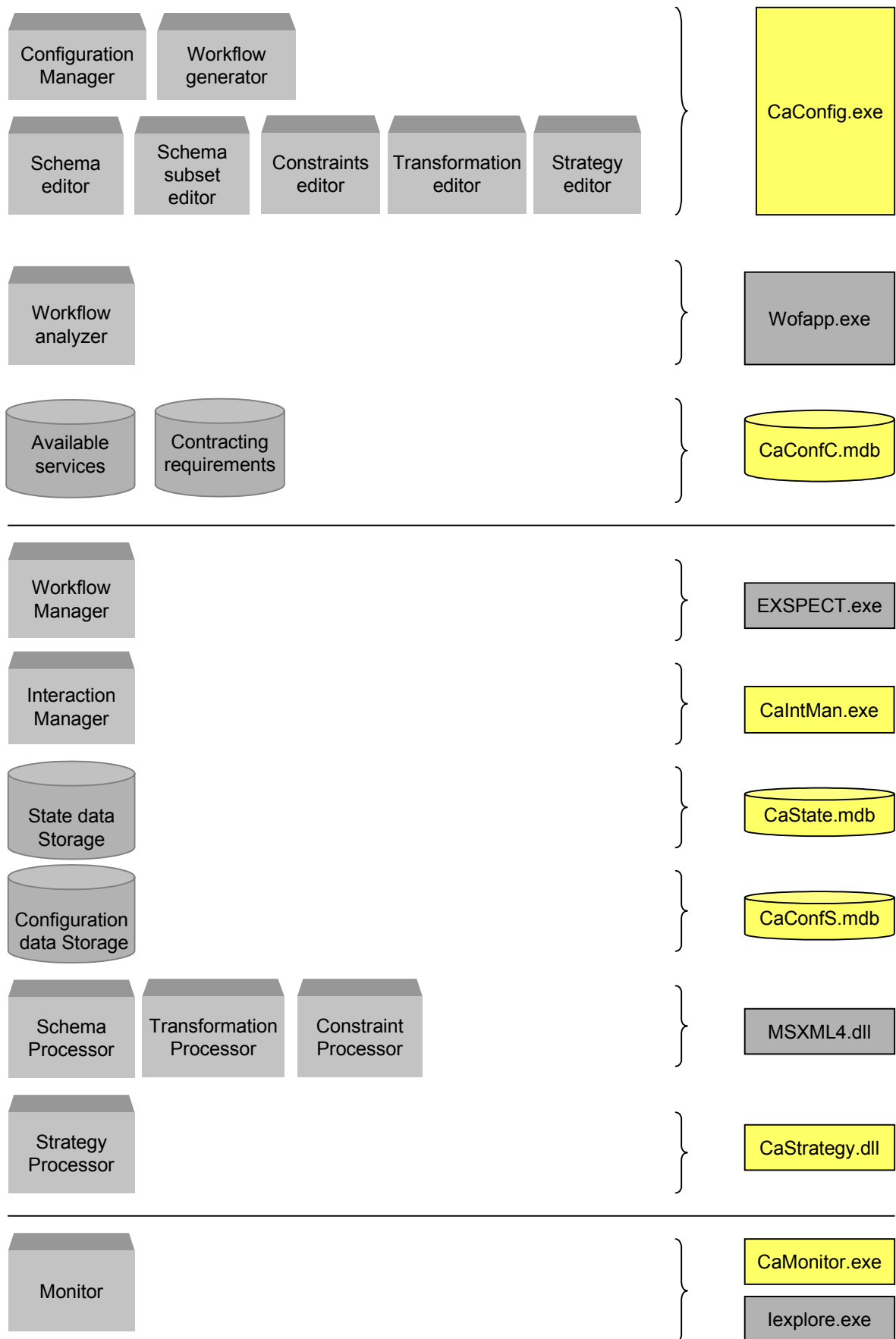


Figure 210 Mapping of components in the logical architecture on components in the technical architecture

4.3 Relational databases

4.3.1 Relational database ‘CaConfC.mdb’

This relational database is used to store persistent data of the Configuration component. The schema of the relational database is given in Figure 211 and Figure 212 and is derived from the functional datamodel in Figure 192. Auxiliary tables and columns, used to store temporary data or derived data are not shown. We documented the schema in two parts for reasons of clarity. An entity is implemented as a table and an attribute is implemented as a column in a table. XML documents are stored in data type ‘Memo’ which allows up to 64 kb of character data. Other data is stored in data type ‘String’. Finally, we do not use the ‘*identification*’ and ‘*type identification*’ attributes as keys, but add an additional primary key ‘*identification*’ to each table that is not an association table. Therefore, the ‘*identification*’ and ‘*type identification*’ attributes in the functional data models of Chapter 3 are represented by a column ‘code’ in the relational database.

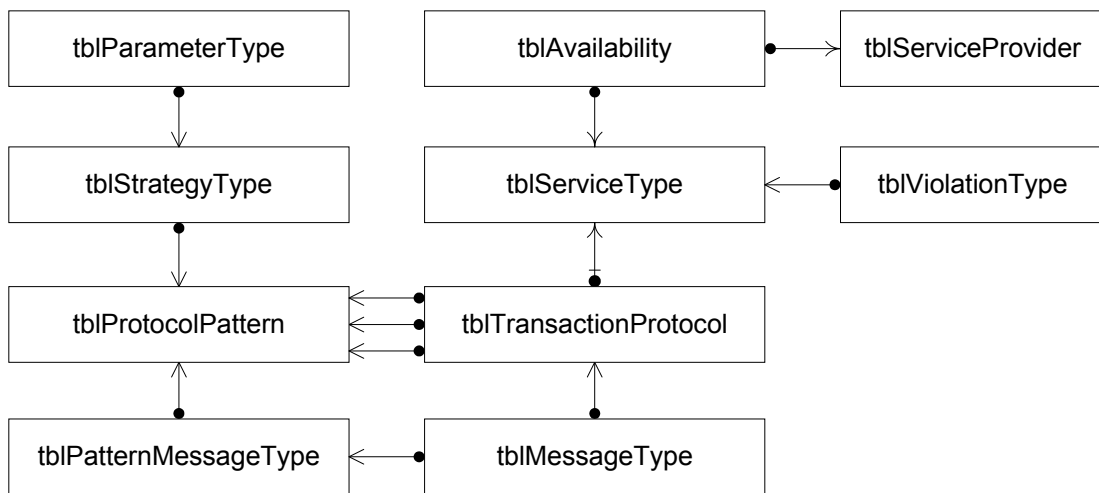


Figure 211 Part I of the relational database ‘CaConfC.mdb’ schema

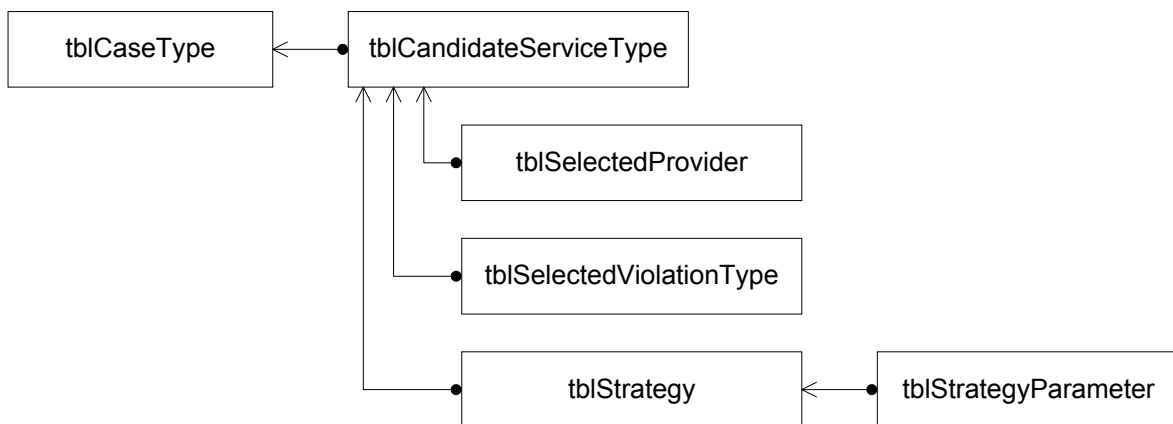


Figure 212 Part II of the relational database ‘CaConfC.mdb’ schema

The following table defines the table structures. The column names printed in bold are primary keys, and the column names printed in italics are foreign keys.

<i>Table</i>	<i>Column</i>	<i>Data type</i>
TblServiceProvider	Identification	Long
	Code	String (50)
	Name	String (100)
	URL	String (250)
TblAvailability	ServiceProvider_Id	Long
	ServiceType_Id	Long
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
TblServiceType	Identification	Long
	Code	String (50)
	Name	String (100)
	Description	Memo
	Schema	Memo (xml)
	SchemaSyntax	String (10)
TblViolationType	Identification	Long
	<i>ServiceType_Id</i>	Long
	Code	String (50)
	Name	String (100)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
TblTransactionProtocol	Identification	Long
	<i>ServiceType_Id</i>	Long
	<i>ProtocolPattern_N_id</i>	Long
	<i>ProtocolPattern_E_id</i>	Long
	<i>ProtocolPattern_A_id</i>	Long
TblMessageType	Identification	Long
	<i>TransactionProtocol_Id</i>	Long
	<i>PatternMessageType_Id</i>	Long
	Name	String (100)
	Schema	Memo (xml)
	Schema syntax	String (10)
TblProtocolPattern	Identification	Long
	Code	String (50)
	Name	String (100)
	Description	Memo
	ProtocolDefinition	Memo (xml)

TblPatternMessageType	Identification	Long
	<i>ProtocolPattern_Id</i>	Long
	Code	String (50)
	Name	String (100)
TblStrategyType	Identification	Long
	<i>ProtocolPattern_Id</i>	Long
	Code	String (50)
	Name	String (50)
	Description	Memo
TblParameterType	Identification	Long
	<i>StrategyType_id</i>	Long
	Code	String (50)
	Name	String (100)
	Description	Memo

<i>Table</i>	<i>Column</i>	<i>Data type</i>
TblCaseType	Identification	Long
	Code	String (50)
	Name	String (100)
	Description	Memo
	Schema	Memo (xml)
	SchemaSyntax	String (10)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
TblCandidateServiceType	Identification	Long
	<i>CaseType_Id</i>	Long
	<i>ServiceType_Id</i>	Long
	Code	String (50)
	SpecificationRules	Memo (xml)
	SpecificationRulesSyntax	String (10)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
	Trigger	Memo
TblSelectedProvider	CandidateServiceType_Id	Long
	ServiceProvider_Id	Long
	Preference	Integer
TblStrategy	CandidateServiceType_Id	Long
	StrategyType_Id	Long
	Phase	String(20)

TblStrategyParameter	Strategy_id	Long
	ParameterType_id	Long
	Code	String(50)
	Expression	Memo
TblSelectedViolationType	CandidateServiceType_id	Long
	ViolationType_id	Long
	ParameterData	Memo

Figure 213 Data types of columns in relational database 'CaConfC.mdb'

4.3.2 Relational database 'CaState.mdb'

The schema of the relational database is given in Figure 214. The schema is derived from the functional datamodel in Figure 152. Auxiliary tables and columns, used to store temporary data or derived data are not shown. An entity is implemented as a table and an attribute is implemented as a column in a table. XML documents are stored in data type 'Memo' which allows up to 64 kb of character data. Other data is stored in data type 'String'.

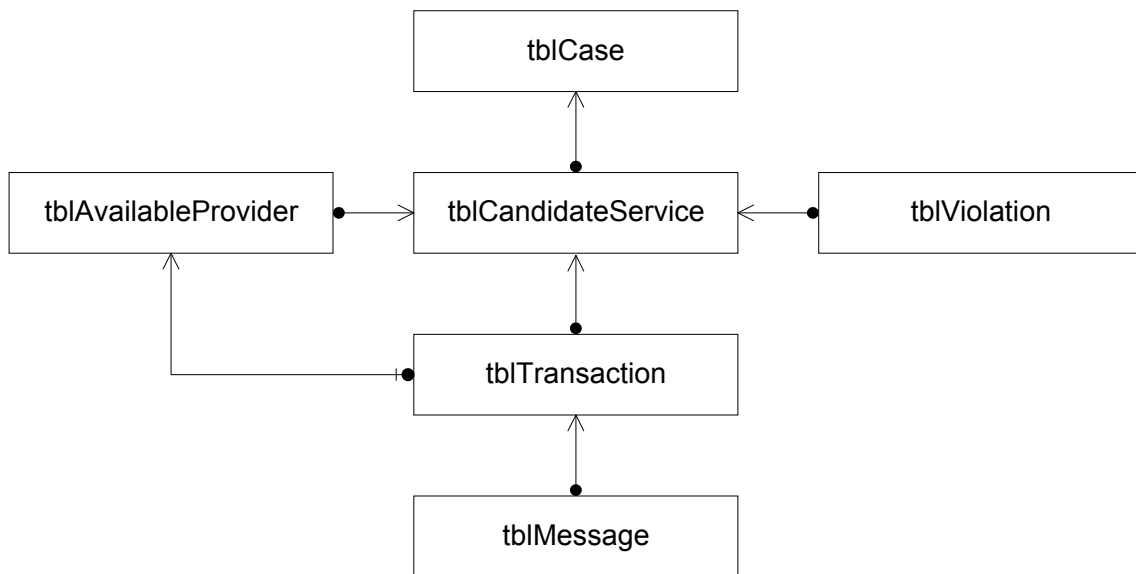


Figure 214 Schema of relational database 'CaState.mdb'

Table	Column	Data type
TblCase	Identification	Long
	<i>CaseType_Id</i>	Long
	Code	String (50)
	OwnerCode	String (50)
	Status	String (50)
	Date	Date
	Time	Time
	Data	Memo (xml)

TblCandidateService	Identification	Long
	<i>Case_Id</i>	Long
	<i>CandidateServiceType_Id</i>	Long
	<i>Provider_Id</i>	Long
	Data	Memo (xml)
TblTransaction	Identification	Long
	<i>CandidateService_Id</i>	Long
	<i>AvailableProvider_Id</i>	Long
	Code	String (50)
	State	String (100)
TblMessage	Identification	Long
	<i>Transaction_Id</i>	Long
	<i>MessageType_id</i>	Long
	Code	String (50)
	Date	Date
	Time	Time
	Direction	String(1)
	Value	String(250)
	Status	String(50)
	Data	Memo (xml)
TblViolation	Identification	Long
	<i>CandidateService_id</i>	Long
	<i>ViolationType_id</i>	Long
	Date	Date
	Time	Time
TblAvailableProvider	Identification	Long
	<i>CandidateService_id</i>	Long
	Code	String (50)
	Preference	Integer

Figure 215 Data types of columns in relational database 'CaState.mdb'

4.3.3 Relational database ‘CaConfS.mdb’

The schema of the relational database is given in Figure 216. The schema is derived from the functional datamodel in Figure 153. Auxiliary tables and columns, used to store temporary data or derived data are not shown. An entity is implemented as a table and an attribute is implemented as a column in a table. XML documents are stored in data type ‘Memo’ which allows up to 64 kb of character data. Other data is stored in data type ‘String’.

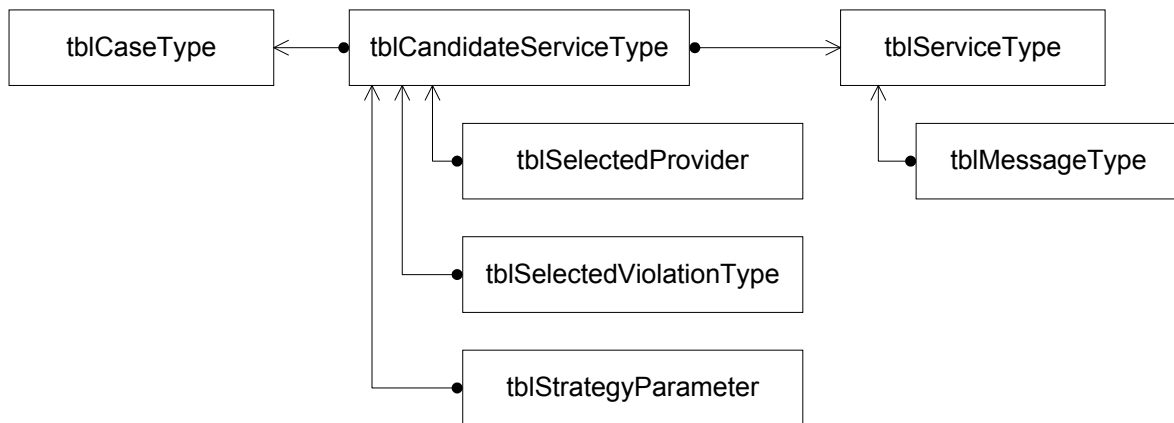


Figure 216 Schema of relational database ‘CaConfigS.mdb’

Table	Column	Data type
TblCaseType	Identification	Long
	Code	String (50)
	Schema	Memo (xml)
	SchemaSyntax	String (10)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
TblCandidateServiceType	Identification	Long
	<i>CaseType_Id</i>	Long
	Code	String (50)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
	SpecificationRules	Memo (xml)
	SpecificationRulesSyntax	String (10)
TblServiceType	Identification	Long
	Code	String (50)
	Schema	Memo (xml)
	SchemaSyntax	String (10)
TblMessageType	Identification	Long
	<i>ServiceType_Id</i>	Long
	Code	String (50)
	Schema	Memo (xml)
	SchemaSyntax	String (10)

TblSelectedProvider	Identification	Long
	<i>CandidateServiceType_Id</i>	Long
	Code	String (50)
	Preference	String (50)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
TblSelectedViolationType	Identification	Long
	<i>CandidateServiceType_Id</i>	Long
	Code	String (50)
	Constraints	Memo (xml)
	ConstraintsSyntax	String (10)
	ParameterData	Memo
TblStrategyParameter	Identification	Long
	<i>CandidateServiceType_Id</i>	Long
	Code	String (50)
	Expression	Memo

Figure 217 Data types of columns in relational database 'CaConfigS.mdb'

4.4 COM components

4.4.1 Out-of-process COM server 'CaIntMan.exe'

The purpose of this component is to act as the 'glue' connecting the other components. As can be seen in Figure 209, the component is used both as COM-client and as COM-server. Therefore, the application is developed as 'ActiveX EXE' in the Visual Basic system. We will now define the structure of the COM-interface exposed by the component. The interface consists of a class 'clsContractingAgent' with the following methods.

<i>Place (in Figure 133)</i>	<i>Implemented as</i>
Initiate request (in)	Method 'Initiate' of COM server 'CaIntMan'
Initiate accept (out)	Return value of method 'Initiate'
Initiate reject (out)	Return value of method 'Initiate'
Finished notification (out)	Event 'Finished' raised by COM server 'CaIntMan'
Case list request (in)	Method 'GetCaseList' of COM server 'CaIntMan'
Case list response (out)	Return value of method 'GetCaseList'
Case details request (in)	Method 'GetCaseDetails' of COM server 'CaIntMan'
Case details response (out)	Return value of method 'GetCaseDetails'
Configuration request (in)	Method 'SetConfiguration' of COM server 'CaIntMan'
Configuration response (out)	Return value of method 'SetConfiguration'

Figure 218 Mapping of the places in the logical architecture to the technical architecture

- **Method 'Initiate'**

```
Public Function Initiate( strCaseId      As String,
                        strCaseOwner   As String,
                        strCaseType    As String,
                        strCaseData    As String) As String
```

The method returns an empty string if the service contracting process for the new business case was started successfully. Otherwise, the method returns a string that contains a description of the error. Possible error messages are:

- 'case identification missing'
- 'case identification is not unique'
- 'case type missing'
- 'case type invalid or unknown'
- 'case data missing'
- 'case data invalid: ' <error description>

- **Event 'Finished'**

```
Public Event Finished ( strCaseId      As String,
                       strCaseOwner   As String,
                       strEndState    As String ) As String
```

This event is raised by the Contracting Agent when a service contracting process ends, either normally or with exceptions.

- **Method 'GetCaseList'**

```
Public Function GetCaseList( strCaseOwner As String,
                            strCaseType  As String,
                            strCaseStartDate As String,
                            strCaseEndDate As String) As String
```

This method is part of the monitor interface and is used to retrieve a list of all business cases for which a service contracting process is being executed or has been executed. The method returns a string containing an XML document based on the DTD in Figure 219.

```
<!ELEMENT CaseList      ( Case+ )>
<!ELEMENT Case          ( #PCDATA )>

<!ATTLIST CaseList
  owner      CDATA #REQUIRED
  type       CDATA #REQUIRED
  startdate  CDATA #REQUIRED
  enddate    CDATA #REQUIRED >

<!ATTLIST Case
  identification CDATA #REQUIRED
  owner          CDATA #REQUIRED
  type           CDATA #REQUIRED
  status         CDATA #REQUIRED
  date           CDATA #REQUIRED
  time          CDATA #REQUIRED >
```

Figure 219 DTD of a 'CaseList' XML document type

- **Method ‘GetCaseDetails’**

```
Public Function GetCaseDetails( strCaseId      As String,
                              strCaseOwner  As String) As String
```

This method is part of the monitoring interface and is used to retrieve an overview of all candidate services, transactions, messages and violations that has been created in the service contracting process of one specific business case. The method returns a string that contains an XML document defined by the DTD in Figure 220. The ‘data’ elements do not contain the actual data, but contain file names that refer to files in the file system.

```
<!ELEMENT Case ( CandidateService* )>
<!ELEMENT CandidateService ( Provider*, Transaction*, Violation* )>
<!ELEMENT Transaction ( Message* )>
<!ELEMENT Message ( #PCDATA )>
<!ELEMENT Violation ( #PCDATA )>
<!ELEMENT Provider ( #PCDATA )>

<!ATTLIST Case
  identification CDATA #REQUIRED
  owner CDATA #REQUIRED
  type CDATA #REQUIRED
  status CDATA #REQUIRED
  date CDATA #REQUIRED
  time CDATA #REQUIRED
  data CDATA #REQUIRED >

<!ATTLIST CandidateService
  type CDATA #REQUIRED
  providerid CDATA #REQUIRED
  status CDATA #REQUIRED
  data CDATA #REQUIRED >

<!ATTLIST Provider
  identification CDATA #REQUIRED
  name CDATA #REQUIRED
  preference CDATA #REQUIRED >

<!ATTLIST Transaction
  identification CDATA #REQUIRED
  providerid CDATA #REQUIRED
  state CDATA #REQUIRED >

<!ATTLIST Message
  identification CDATA #REQUIRED
  type CDATA #REQUIRED
  direction CDATA #REQUIRED
  value CDATA #REQUIRED
  status CDATA #REQUIRED
  date CDATA #REQUIRED
  time CDATA #REQUIRED
  data CDATA #REQUIRED >
```

```

<!ATTLIST Violation
    type          CDATA #REQUIRED
    date          CDATA #REQUIRED
    time         CDATA #REQUIRED >

```

Figure 220 DTD of a 'Case' XML document type

4.4.2 Stand alone executable program 'CaConfig.exe'

This program implements the functionality of the entire Configurator component. It offers a user interface by which the relational database 'CaConfC.mdb' can be maintained (for examples see Section 4.6). Besides manual data entry, the configuration database can also be filled by importing XML documents with service types, service providers, protocol patterns and contracting strategies. The structure of these XML documents is defined by the following DTD's.

```

<!ELEMENT ServiceTypeDef ( ServiceType, ViolationType*, Protocol? )>
<!ELEMENT ServiceType ( TypeId, Name, Description, Schema )>
<!ELEMENT ViolationType ( TypeId, Name, Constraints )>
<!ELEMENT Protocol ( PatternId_N, PatternId_E, PatternId_A,
    MessageType+ )>
<!ELEMENT Schema ( Specification, Syntax )>
<!ELEMENT Constraints ( Specification, Syntax )>
<!ELEMENT MessageType ( TypeId, Name, Schema )>
<!ELEMENT TypeId (#PCDATA)
<!ELEMENT Name (#PCDATA)
<!ELEMENT Description (#PCDATA)
<!ELEMENT PatternId_N (#PCDATA)
<!ELEMENT PatternId_E (#PCDATA)
<!ELEMENT PatternId_A (#PCDATA)
<!ELEMENT Specification (#PCDATA)
<!ELEMENT Syntax (#PCDATA)

```

Figure 221 DTD of a 'ServiceTypeDef' XML document type

```

<!ELEMENT ServiceProviderDef ( ServiceProvider, Availability* )>
<!ELEMENT ServiceProvider ( Identification, Name, Url )>
<!ELEMENT Availability ( ServiceType, Constraints? )>
<!ELEMENT Constraints ( Specification, Syntax )>
<!ELEMENT Identification (#PCDATA)
<!ELEMENT Name (#PCDATA)
<!ELEMENT Url (#PCDATA)
<!ELEMENT ServiceType (#PCDATA)
<!ELEMENT Specification (#PCDATA)
<!ELEMENT Syntax (#PCDATA)

```

Figure 222 DTD of a 'ServiceProviderDef' XML document type

```

<!ELEMENT ProtocolPatternDef ( ProtocolPattern, MessageType+ )>
<!ELEMENT ProtocolPattern ( Identification, Name, ProtocolDefinition)>
<!ELEMENT MessageType ( Type, Name )>
<!ELEMENT Identification (#PCDATA)
<!ELEMENT ProtocolDefinition (#PCDATA)
<!ELEMENT Type (#PCDATA)
<!ELEMENT Name (#PCDATA)

```

Figure 223 DTD of a 'ProtocolPatternDef' XML document type

```

<!ELEMENT StrategyTypeDef ( Identification, ProtocolPatternId, Name,
Description, ParameterType* )>
<!ELEMENT ParameterType ( Identification, Name, Description )>
<!ELEMENT Identification (#PCDATA)
<!ELEMENT ProtocolPatternId (#PCDATA)
<!ELEMENT Name (#PCDATA)
<!ELEMENT Description (#PCDATA)

```

Figure 224 DTD of a 'StrategyTypeDef' XML document type

4.4.3 Stand alone executable program 'CaMonitor.exe'

This program implements the functionality of the Monitor component. The component has no persistent data, but uses the monitoring program interface (methods 'GetCaseList' and 'GetCaseDetails') of the Server component to query its state data and present it to the user. Examples of this user interface are given in Section 4.6. The Monitor component invokes the Internet Explorer when case data, service data or message data must be presented to the user. Internet Explorer 5 supports XML and XSL, which makes it a suitable component for displaying the contents of XML documents in a user-friendly way. The Monitor component creates an output file with the XML document that must be displayed, preceded by a reference to the appropriate stylesheet, for example:

```

<?xml-stylesheet type="text/xsl" href="c:\temp\case_businessstrip.xsl"?>
<Case>
  <Type>Business Trip</Type>
  <Identification>12345</Identification>

```

Figure 225 Including a reference to an XSL stylesheet in an XML document

When the output file containing the XML document and the reference to the XSL style sheet is created, Internet Explorer is started with the file name of the output file as parameter. Internet explorer reads the input file, signals the reference to the XSL style sheet and loads the style sheet too. There after, the information in the XML document is transformed into HTML according to the transformation rules in the style sheet. The HTML is then displayed in the browser user interface.

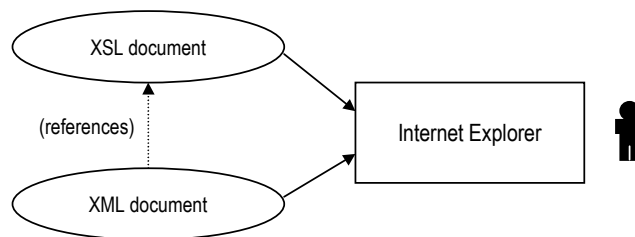


Figure 226 Displaying XML documents formatted by XSL stylesheets in IE

4.5 Structure of the generated ExSpect model

4.5.1 Hierarchic level 1: the main system

We will now give some examples of the structure of the generated ExSpect model, which acts as workflow definition in the Server component. The generated ExSpect model contains one top level system 'main' that consists of a channel for each place in the interface between the Workflow Manager and the Interaction Manager as shown in Figure 151. The channel names are prefixed with 'IM_' for clarification of the generated workflow and to indicate their role as interface with the Interaction Manager. Furthermore, the system contains a subsystem for each case type. An example of the main system in a generated ExSpect model is shown in Figure 227.

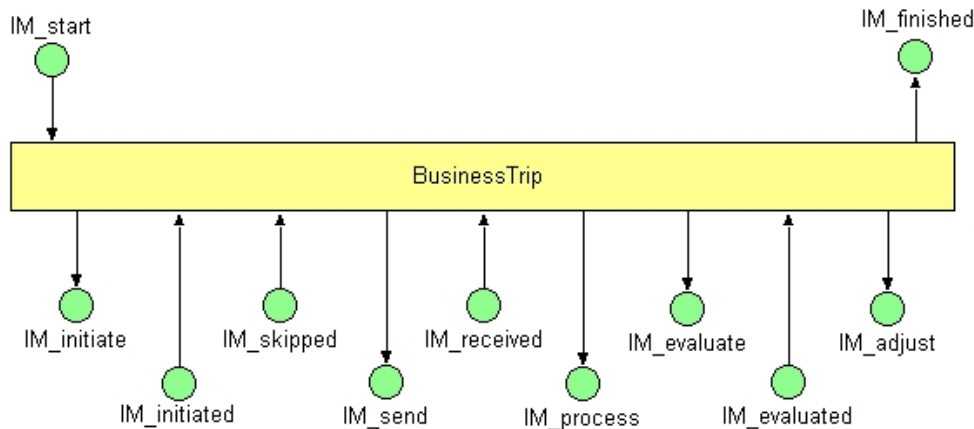


Figure 227 Example of a generated ExSpect model (top level)

The following channel types are used in this ExSpect model.

```

type Tcase      := [caseid:          str,
                    caseowner:       str,
                    casetype:        str,
                    ctrl_i:          num,
                    ctrl_n:          num,
                    ctrl_v:          str,
                    ctrl_vmax:       str,
                    ctrl_pid:        num,
                    ctrl_lastmid:    str,
                    ctrl_optmid:     str];

```

```
type Tstart      := [caseid:      str,
                    caseowner:   str,
                    casetype:    str];

type Tinitiate := [caseid:      str,
                    caseowner:   str,
                    candidateid: str];

type Tskipped  := [caseid:      str,
                    caseowner:   str,
                    candidateid: str];

type Tinitiated := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    providers:   num];

type Tsend     := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    providerid:  num,
                    messagetype: str];

type Treceived := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    providerid:  num,
                    messageid:   str,
                    messagetype: str];

type Tprocess  := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    messageid:   str];

type Tevaluate := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    messageid:   str];

type Tevaluated := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    messageid:   str,
                    messagevalue: str];

type Tadjust   := [caseid:      str,
                    caseowner:   str,
                    candidateid: str,
                    strategyparid: str];

type Tfinished := [caseid:      str,
                    caseowner:   str,
                    casestatus:  str];
```


4.5.2 Hierarchic level 2: case type

The second hierarchic level of the generated ExSpect model represents the contracting workflow for a single case type. The structure of this system is derived from the contracting requirements and consists of systems representing a candidate service type (see 4.5.3) and systems representing OR-joins, AND-splits and AND-joins. An OR-join is modelled as N separate elementary transitions. An example of a system for a case type that involves just two candidate service types is given in Figure 228.

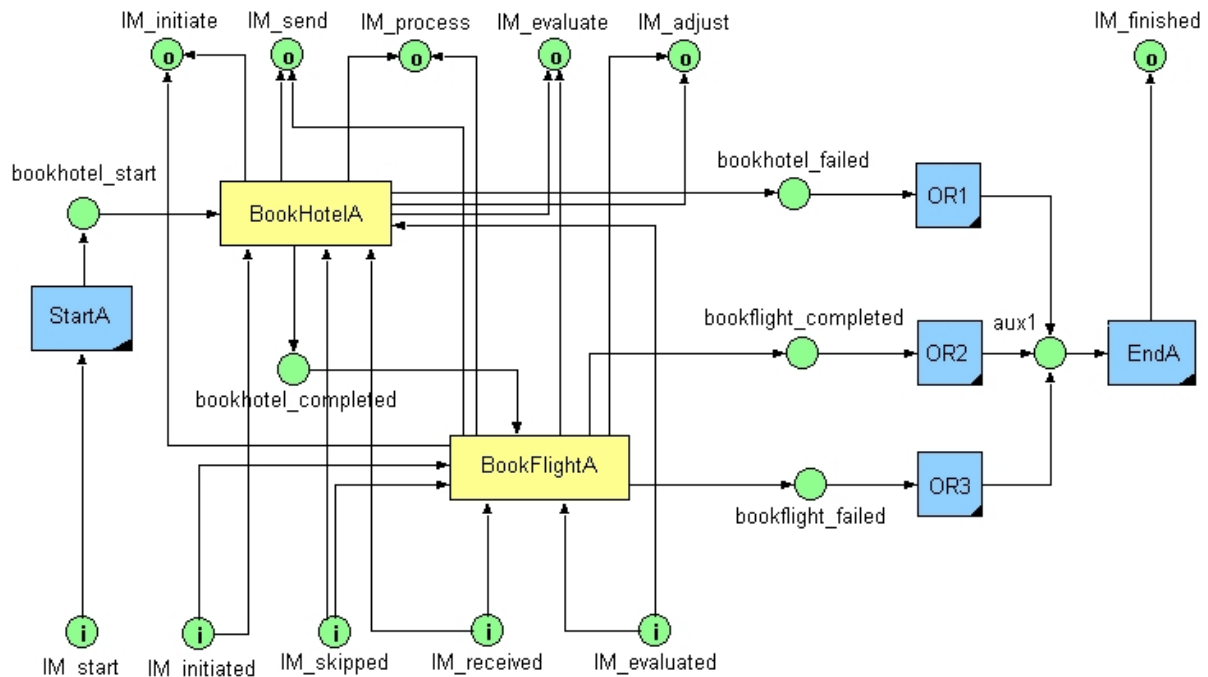


Figure 228 Example of a generated ExSpect model (hierarchic level 2)

An example of an elementary transition used at this level in the hierarchy is the transition that consumes tokens from the 'IM_start' place of which the case type is equal to the case type corresponding to the system. The transition produces a token in the start place for all systems representing candidate service types without a trigger.

```

proc StartA[in IM_start: Tstart,
            out bookhotel_start: Tcase
            | pre IM_start@casetype = 'My case type']
:=
  bookhotel_start <- IM_start
;

```

Figure 229 Example of a processor for initiating a candidate service

4.5.3 Hierarchic level 3: candidate service type

The third hierarchic level of the generated ExSpect model represents the contracting process for a single candidate service. The structure of this system, shown in Figure 230, implements the relevant part of the model in Figure 77.

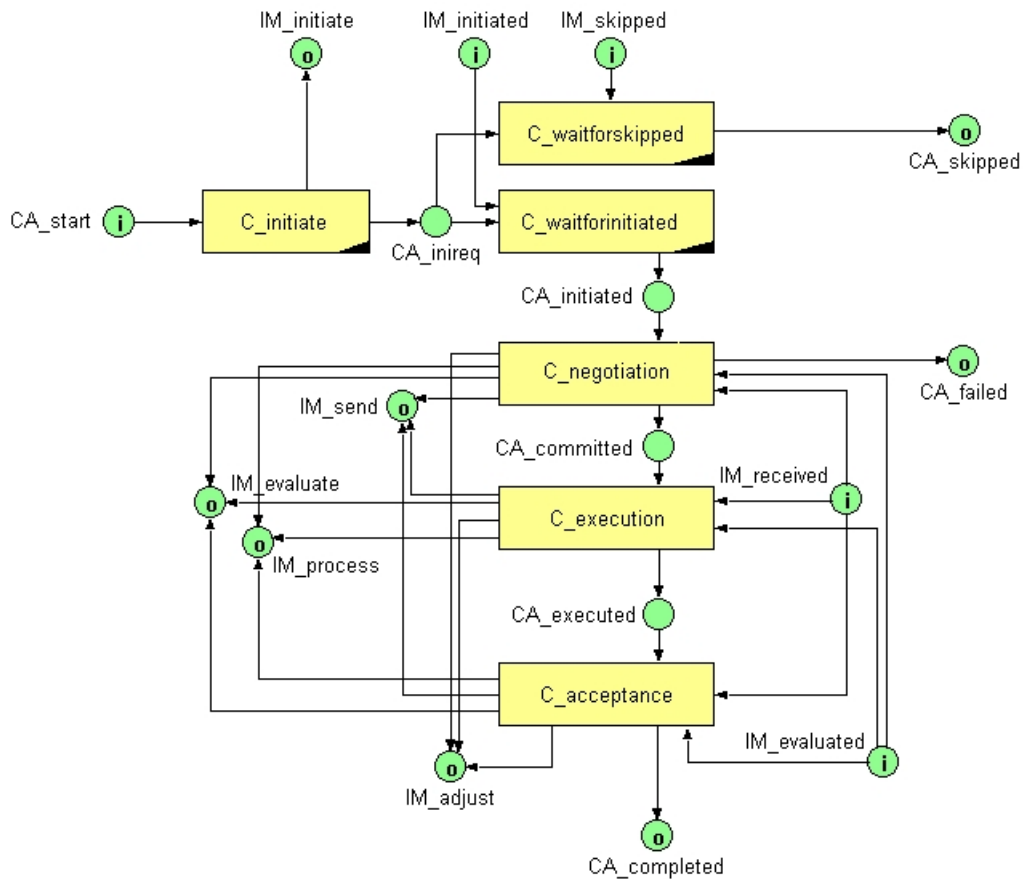


Figure 230 Example of a generated ExSpect model (hierarchic level 3)

We will give two examples of processor definitions used in this system.

```

proc C_initiate[in CA_start: Tcase,
                out CA_inireq: Tcase,
                out IM_initiate: Tinitiate]
:=
  CA_inireq <- CA_start,
  IM_initiate <- [caseid: CA_start@caseid,
                 caseowner: CA_start@caseowner,
                 candidateid: 'Book Outbound']
;
    
```

Figure 231 Example of a processor for initiating a candidate service

```

proc C_waitforinitiated[in CA_inireq: Tcase,
                        in IM_initiated: Tinitiated,
                        out CA_initiated: Tcase
                        | pre IM_initiated@caseid = CA_inireq@caseid and
                          IM_initiated@caseowner = CA_inireq@caseowner and
                          IM_initiated@candidateid = 'Book Outbound']
:=
    
```

```

CA_initiated <- upd(CA_inireq, [ctrl_n:
                        IM_initiated@providers])
;

```

Figure 232 Example of a processor for responding to an initiated candidate service

```

proc C_waitforskipped[in CA_inireq: Tcase,
                     in IM_skipped: Tskipped,
                     out CA_skipped: Tcase
  | pre IM_skipped@caseid = CA_inireq@caseid and
      IM_skipped@caseowner = CA_inireq@caseowner and
      IM_skipped@candidateid = 'Book Outbound']
:=
CA_skipped <- CA_inireq
;

```

Figure 233 Example of a processor for responding to a candidate service that must be skipped

4.5.4 Hierarchic level 4: contracting phase

The fourth hierarchic level of the generated ExSpect model represents a negotiation phase, an execution phase or an acceptance phase. The structure of these systems depends on the strategy chosen. The example in Figure 234 is based on a ‘binding request’ protocol pattern.

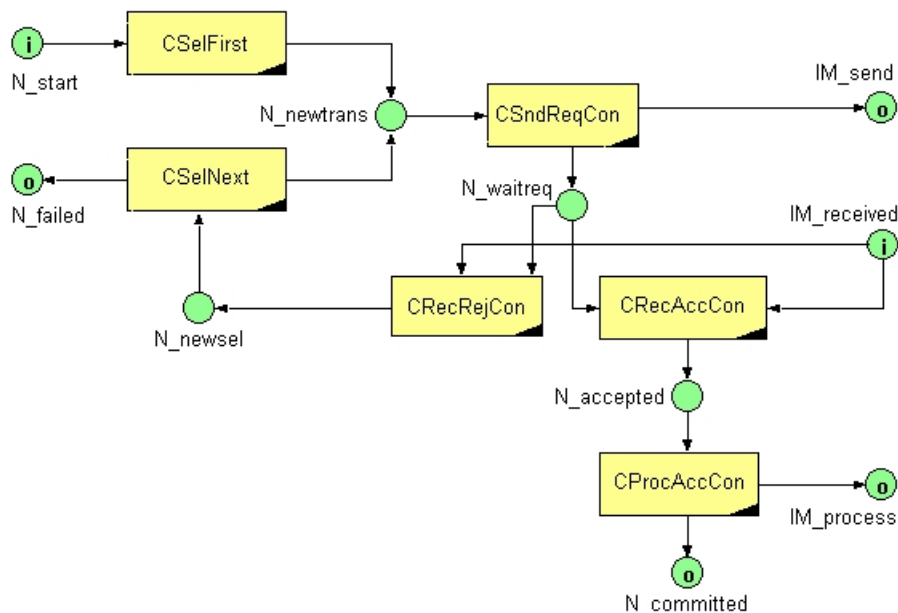


Figure 234 Example of a generated ExSpect model (hierarchic level 4)

Two important types of processors are the equivalents of the standard transitions for sending and receiving messages.

```

proc CSndReqCon[in N_start: Tcase,
                out N_end: Tcase,
                out IM_send: Tsend]
:=
  IM_send <- [caseid: N_start@caseid,
              caseowner: N_start@caseowner,
              candidateid: 'Book Outbound',
              providerid: N_start@ctrl_i,
              messagetype: 'request contract'],
  N_end <- N_start
;

```

Figure 235 Example of a processor for sending a message

```

proc CRecAccCon[in N_start: Tcase,
                in IM_received: Treceived,
                out N_end: Tcase
  | pre IM_received@caseid = N_start@caseid and
      IM_received@caseowner = N_start@caseowner and
      IM_received@candidateid = 'Book Outbound' and
      IM_received@messagetype = 'accept contract']
:=
  N_end <- upd(N_start, [ctrl_lastmid: IM_received@messageid])
;

```

Figure 236 Example of a processor for receiving a message

Another example of a processor definition used in this system is the processor that selects the next available service provider.

```

proc CSelNext[in N_start: Tcase,
               out N_end: Tcase,
               out N_failed: Tcase]
:=
  if N_start@ctrl_i < N_start@ctrl_n then
    N_end <- upd(N_start, [ctrl_i: N_start@ctrl_i + 1])
  else
    N_failed <- N_start
  fi
;

```

Figure 237 Example of a processor for selecting the next provider

4.6 Examples of the user interfaces

4.6.1 Defining available services (repository)

The user interface provided by the Configurator component starts with the screen in Figure 238 from which all available functions can be accessed.



Figure 238 Example of the main screen of the Configurator component

Screen: 'Protocol patterns and strategies'

This screen appears when the user selects the menu option '*Repository | Protocol patterns*' from the menu bar in the main screen. The layout of the screen is shown in Figure 239. It contains the available protocol patterns per phase (negotiation, execution, and acceptance) and the available strategies per protocol pattern. The user can click a protocol pattern or strategy, after which an illustration of the protocol pattern or strategy appears in the right part of the screen.

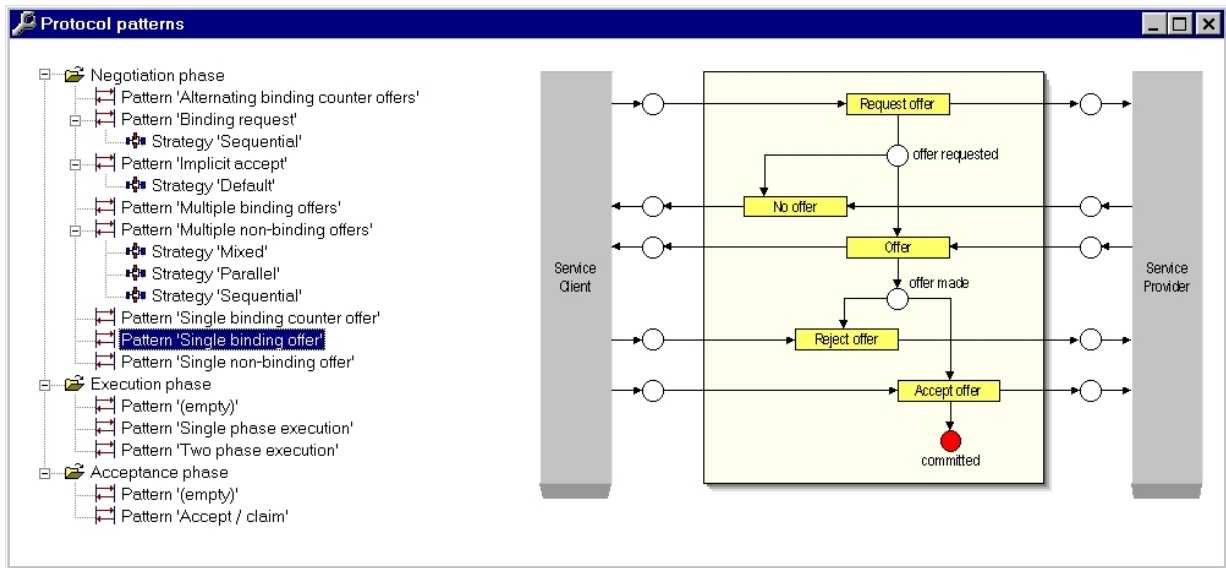


Figure 239 Example of the 'protocol patterns' screen

Screen: 'Service types'

This screen appears when the user selects the menu option 'Repository | Service types' from the menu bar in the main screen. The layout of the screen is shown in Figure 240. It contains a list of service types with for each service type the name and description.



The screenshot shows a window titled 'Service types' with a menu bar containing 'Files' and 'Help'. Below the menu bar is a vertical toolbar with icons for adding, deleting, editing, and navigating. The main area contains a table with two columns: 'Service Type' and 'Description'. The 'Book Flight' row is highlighted in yellow.

Service Type	Description
Arrange road transport	The transportation of general cargo (pallets, boxes, etc.) from a place of loading to a place of discharge.
Book Flight	Book a seat on a scheduled flight.
Book Hotel	Book a room in a hotel for one or more days.
Book Rental car	Book a rental car for one or more days.
Cancel Flight	Cancel an earlier made reservation for a scheduled flight.

Figure 240 Example of the 'Service types' screen

The user can add, modify and delete items in the list. The user can invoke the schema editor to maintain the service type schema by selecting a service type in the list and clicking the button. The user can navigate to the 'transaction protocol' screen by selecting a service type in the list and clicking the button. The user can navigate to the 'violation types' screen by selecting a service type in the list and clicking the button.

Screen: 'Schema editor'

The service type schema is maintained by the schema editor, of which the screen layout is given in Figure 241. The hierarchic data model is represented as a tree in which a  node represents an entity and a  node represents an attribute. The user can select a node by clicking on it, after which the properties of the node can be maintained in the right part of the screen. The *name* property is used to identify the entity or attribute in documentation, and for deriving the XML tag in 'CamelCase' format. The *required* property applies to attribute nodes only and is used to specify whether the attribute must occur in an instance of the data model or not. The *description* property is used to specify a description in free text of the meaning of an entity or attribute. The *minoccurs* and *maxoccurs* properties apply to entity nodes only. They specify the minimum and maximum number of instances of an entity type in a data model. The *type* and *length* properties apply to attribute nodes only. They specify the datatype and maximum length of the value an attribute can take.

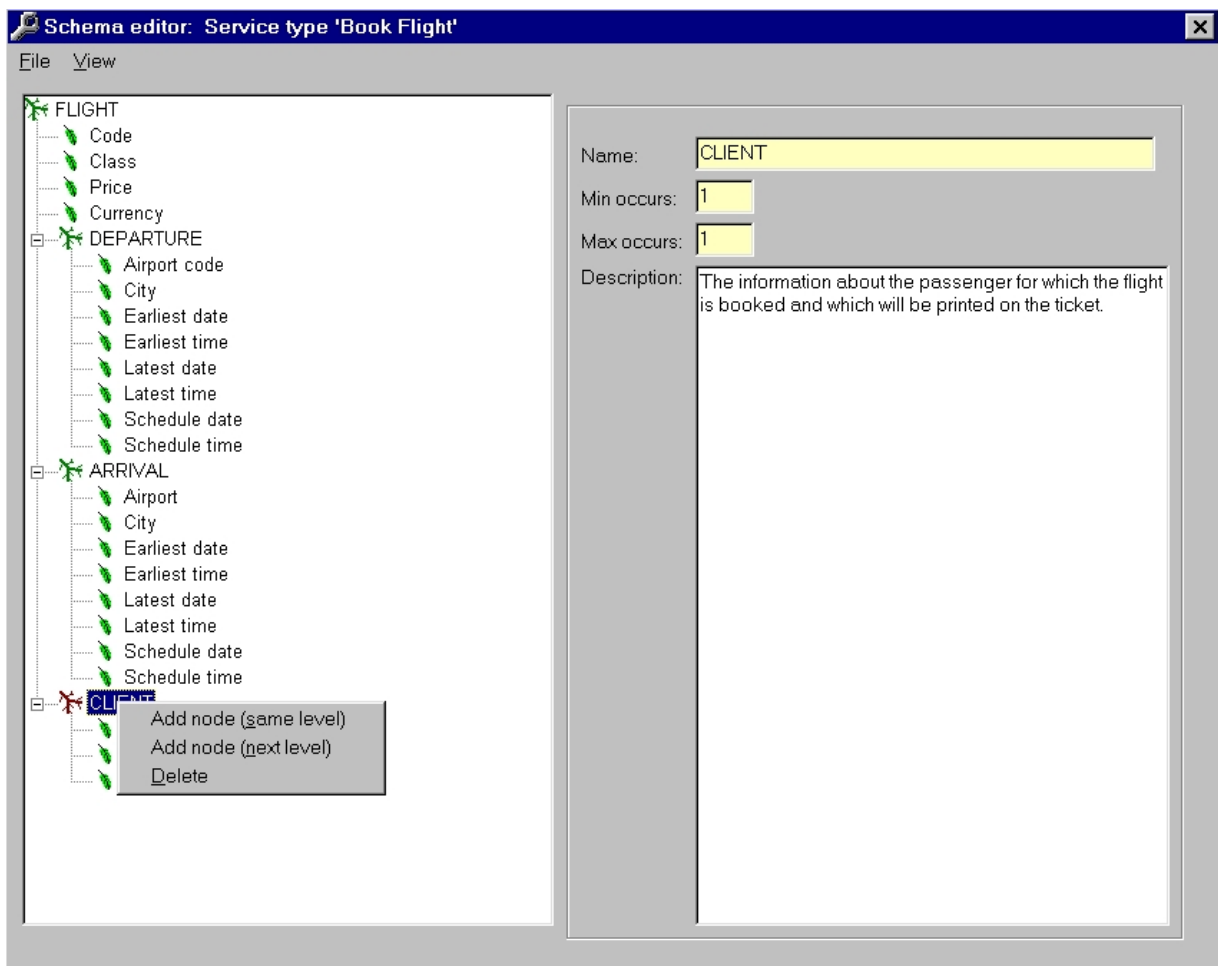


Figure 241 Example of the 'Schema editor' screen

All functions to manipulate the structure of the tree are available via context sensitive menus. These menus appear when the user selects a node and clicks the right mouse button. Each node type has its own menu containing only the functions applicable for that node type. The screen in Figure 241 shows the functions that can be applied to any entity that is not the root-entity.

Finally, the user can view the underlying XML-Schema specification by selecting the menu item 'View | Source'. Here after, a screen like shown in Figure 242 appears.

```

File viewer
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="Flight" type="FlightType"/>
  <xsd:complexType name="FlightType">
    <xsd:sequence>
      <xsd:element name="Code" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Class" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Price" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Currency" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Departure" type="DepartureType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Arrival" type="ArrivalType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="Client" type="ClientType" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="DepartureType">
    <xsd:sequence>
      <xsd:element name="AirportCode" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="City" type="xsd:string" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>

```

Figure 242 Source view in the Schema editor

Screen: ‘Select transaction protocol patterns’

A transaction protocol is based on a negotiation pattern, an execution pattern and an acceptance pattern. The ‘select transaction protocol patterns’ screen allows the user to select these three patterns from three lists of available protocol patterns. The layout of this screen is shown in Figure 243. The three lists contain the available patterns for the three phases. The user can select a pattern in each list and click the ‘Ok’ button to select these patterns for the transaction protocol of the selected service type.

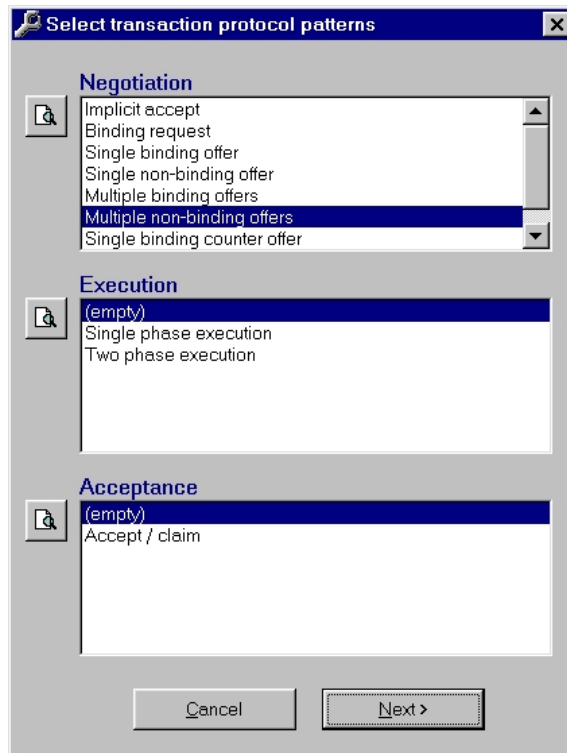


Figure 243 Example of the ‘Select transaction protocol patterns’ screen

Screen: 'Service type transaction protocol'

This screen allows the user to define a transaction protocol by selecting a subset of the message types used in the selected patterns and by assigning specific names to the message types from the selected protocol patterns, e.g. 'transport instruction' instead of 'request contract'. The screen shows a list of message types grouped by the phases negotiation, execution and acceptance. Each arrow represents a message type. The generic name of the message type is printed in the middle of the screen, whereas the specific name can be entered for selected message types at the right part of the screen. If the arrow is displayed dark blue, it is selected in the transaction protocol. If the arrow is displayed grey, it is not used in the transaction protocol. The user can change the status of a message (selected <-> not selected) by clicking on the arrow.

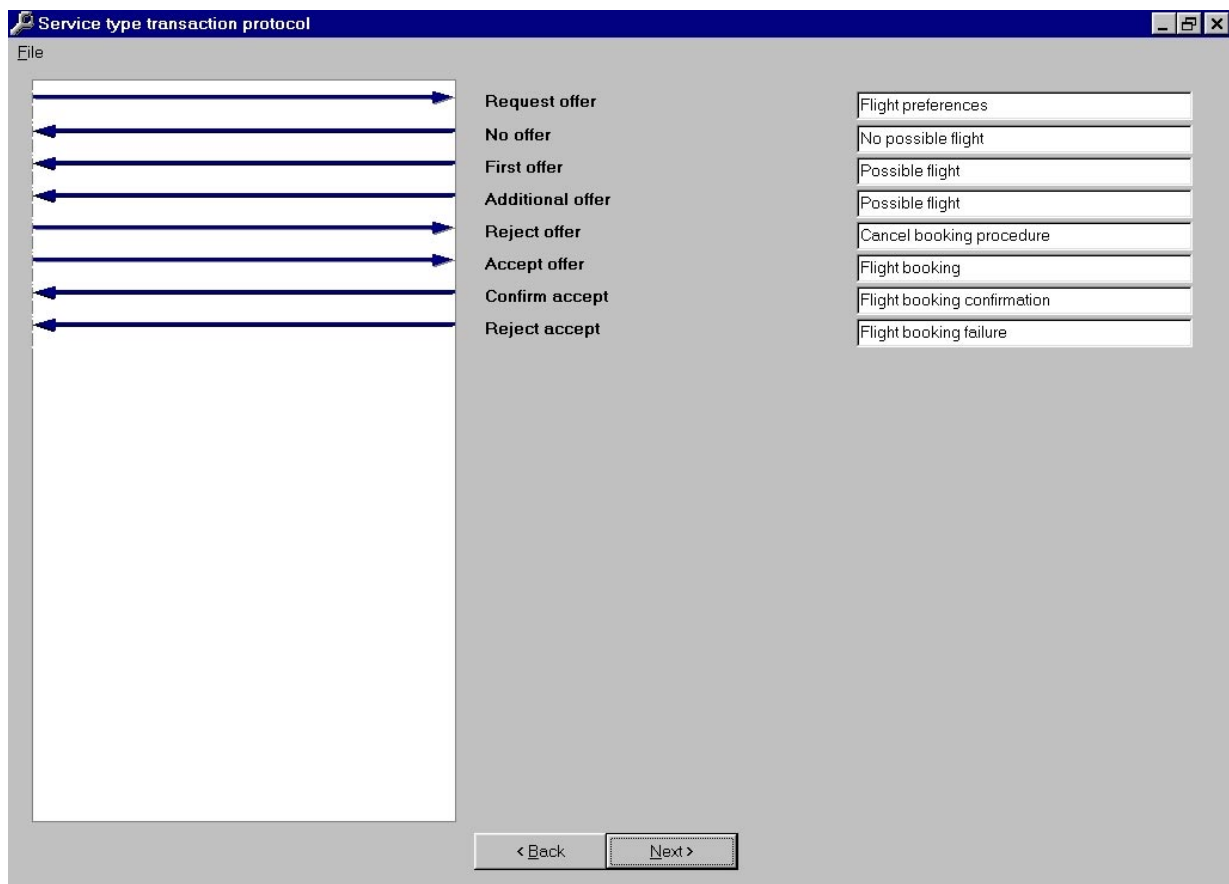


Figure 244 Example of the 'Service type transaction protocol' screen

Screen: 'Schema subset editor'

The function of defining message schemas is available in the screen shown in Figure 245. The screen contains a matrix in which the columns represent the message types that have been selected in the transaction protocol. The rows represent entities and attributes in the service type schema. Nesting of entities is modelled by indentation. The cells represent the status of an entity / attribute in a message type schema, which can be 'R' (= required), 'O' (= optional) and '-' (not used). The user can change the status from 'R' to 'O', from 'O' to '-', and from '-' to 'R' by double clicking the cell.

	Flight preferences	No possible flight	Possible flight	Possible flight	Cancel booking procedure	Flight booking	Flight booking confirmation	Flight booking failure
FLIGHT	R	-	R	R	-	R	R	R
Code	-	-	R	R	-	R	R	R
Class	O	-	R	R	-	R	R	-
Price	-	-	R	R	-	R	R	-
Currency	R	-	R	R	-	R	R	-
DEPARTURE	R	-	R	R	-	R	R	-
Airport code	O	-	R	R	-	R	R	-
City	R	-	R	R	-	-	R	-
Earliest date	O	-	-	-	-	-	-	-
Earliest time	O	-	-	-	-	-	-	-
Latest date	O	-	-	-	-	-	-	-
Latest time	O	-	-	-	-	-	-	-
Schedule date	-	-	R	R	-	-	R	-
Schedule time	-	-	R	R	-	-	R	-
ARRIVAL	R	-	R	R	-	R	R	-
Airport	O	-	R	R	-	R	R	-
City	R	-	R	R	-	-	-	-
Earliest date	-	-	-	-	-	-	-	-
Earliest time	-	-	-	-	-	-	-	-

Figure 245 Example of the 'Schema subset editor' screen

Screen: 'Violation types'

Each service type can have one or more violation types defined, which are used during the execution and acceptance phases to detect violations of the contract. The screen by which the violation types are defined is shown in Figure 246 and contains a grid in which each row represents a violation type. The user can add a new violation type by clicking the button and delete an existing violation type by clicking the button. Finally, the user can open the constraint editor for a violation type by clicking the button.

Description
Planned date/time of delivery exceeds latest date/time of delivery.
Actual date/time of delivery exceeds latest date/time of delivery.
Number of colli reported is lower than number of colli ordered.

Figure 246 Example of the 'Violation types' screen

Screen: 'Constraint editor'

This screen allows the user to create and modify constraints. The layout of the screen is shown in Figure 247. The input field 'constraint syntax' is used to define the constraint expression in the applicable constraint syntax. A constraint can contain parameters, which are identified by using the keywords <P1>, <P2> and <P3> in the constraint syntax. The operands that can be used in the constraint expression are listed in the listbox at the bottom of the screen.

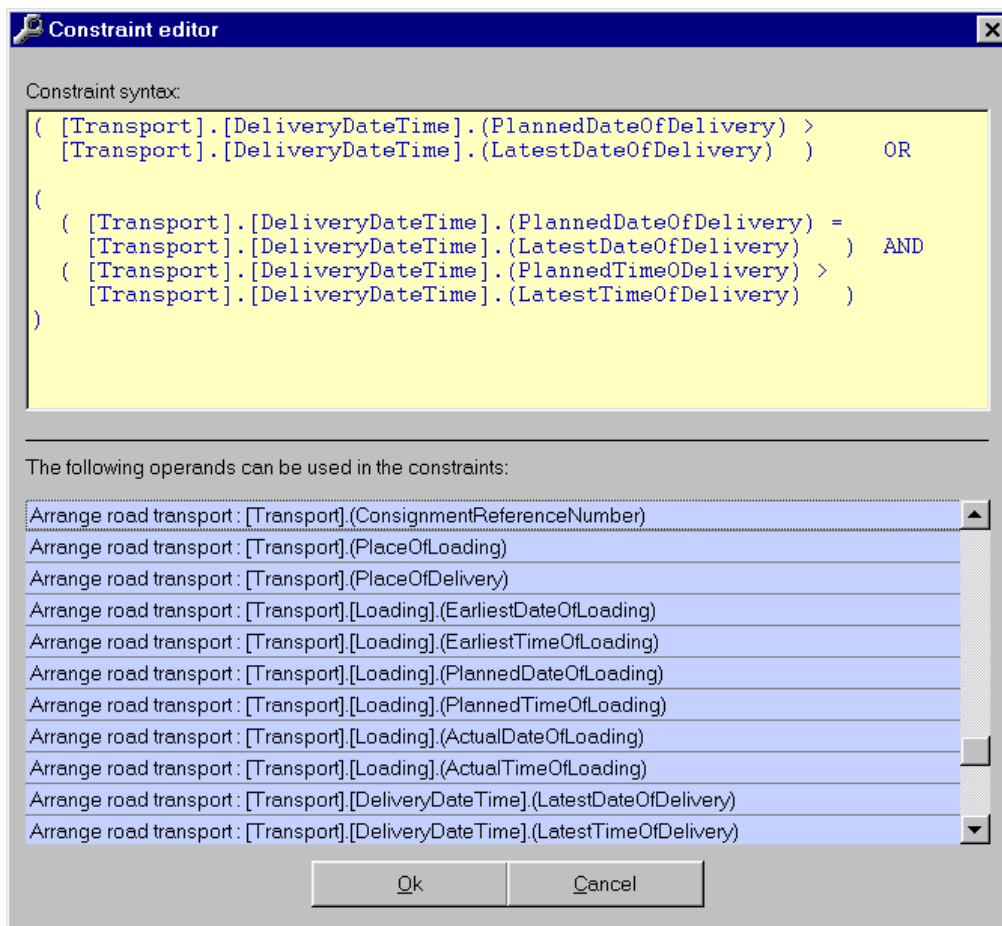


Figure 247 Example of the 'Constraint editor' screen

Screen: 'Service providers'

This screen appears when the user selects the menu option '*Repository | Service providers*' from the menu bar in the main screen. The layout of the screen is shown in Figure 248. It contains a list of service providers, with for each service type the code, the name and the URL.

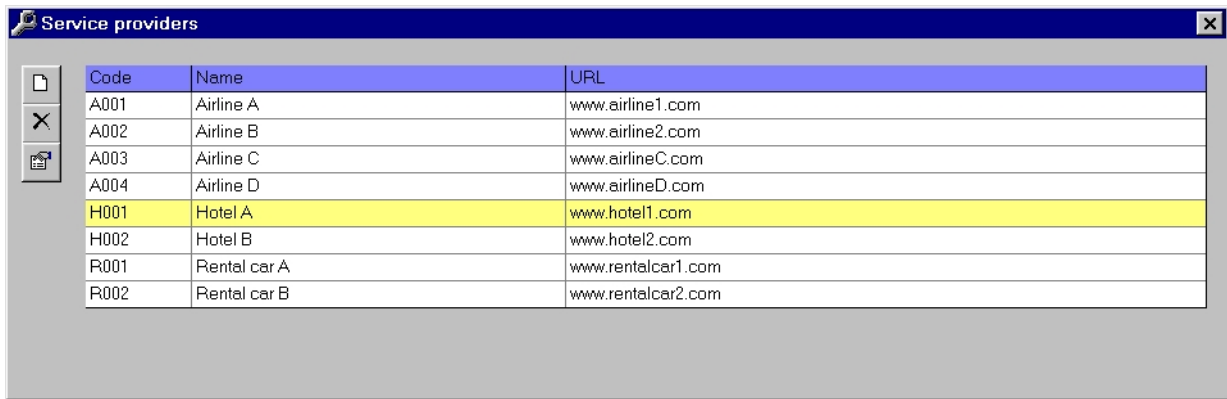


Figure 248 Example of the 'Service providers' screen

The user can add, modify and delete items in these lists and open the details of a provider by clicking the buttons left of the grid.

Screen: 'Service provider details'

This screen lets the user edit the identification, name and URL of a service provider. Furthermore, the user can select the service types provided by the service provider. The layout of the screen is shown in Figure 249. The first column of the grid contains the name of a service type. The second column in the grid contains 'yes' if the service provider offers the service type and 'no' otherwise. The user can change the value from 'yes' to 'no' and from 'no' to 'yes' by double clicking the cell. If a service type is offered subject to constraints on the service data, the availability constraints can be defined by double clicking the third column in the row of that service type.

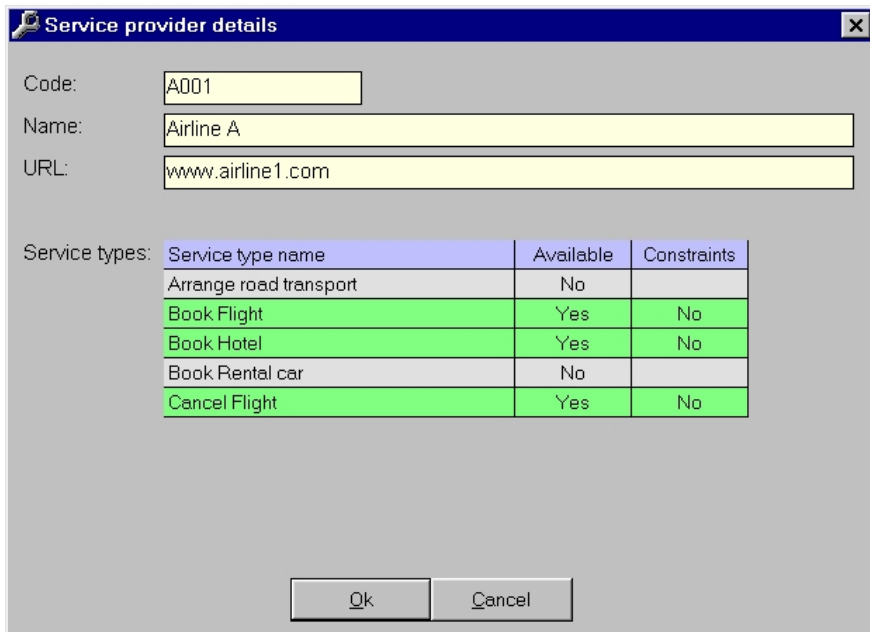







Figure 249 Example of the 'Service provider details' screen

4.6.2 Defining contracting requirements

Screen: 'Case types'

This screen is used to maintain the list of case types that can be processed by the Server component. The layout of the screen is shown in Figure 250. Each row represents a case type with his name and description. The user can add a new case type by clicking the  button and delete an existing case type by clicking the  button. The schema editor can be invoked to maintain the case type schema by clicking the  button. The constraints editor can be invoked to maintain the constraints on the case type data by clicking the  button. Finally, the user can navigate to the 'contracting workflow' screen to define candidate service types and the structure of the contracting workflow (triggers) by clicking the .

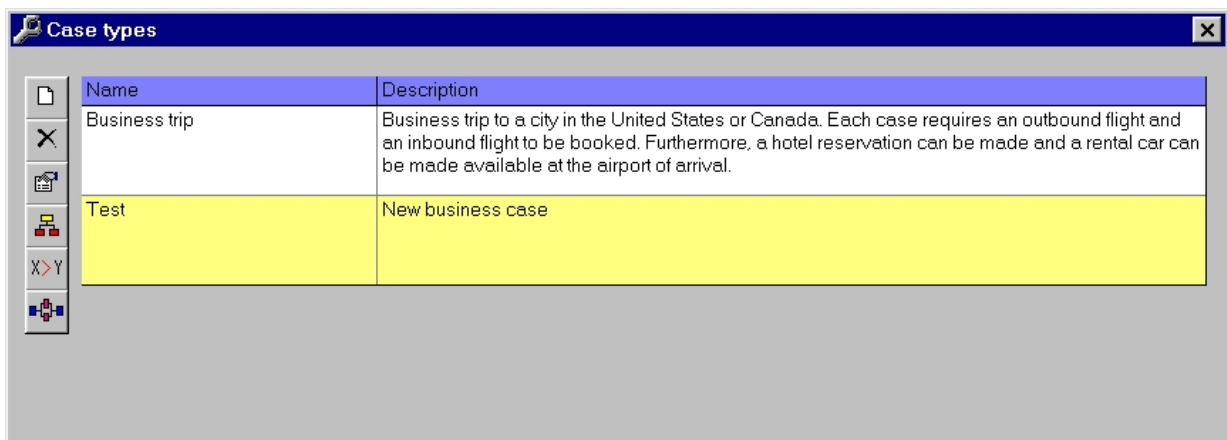


Figure 250 Example of the 'Case types' screen

Screen: 'Contracting workflow'

This screen is used to define the candidate service types for a case type and the structure of the contracting workflow (triggers). An example of the layout of this screen is shown in Figure 251. The contracting workflow is modelled as a graph consisting of *nodes* and *arcs*. There are two types of nodes: a yellow node represents a *candidate service type* and a white node represents a *routing construct*. A candidate service type refers to a specific service type and has a label that is a unique identifier for the candidate service type in the graph. A routing construct can be an OR-join, an AND-split or an AND-join.

An important function of this screen is to complete the partial contracting workflow (consisting of candidate service type transitions and places and transitions that model triggers) into a sound contracting workflow. The user can invoke this function by selecting the '*Tools – Complete contracting workflow*' option in the menu bar. The places, transitions and connectors added by this function can be modified by the user thereafter. Finally, the user can check the soundness of the contracting workflow by selecting the '*Tools – Check soundness*' option from the menu bar.

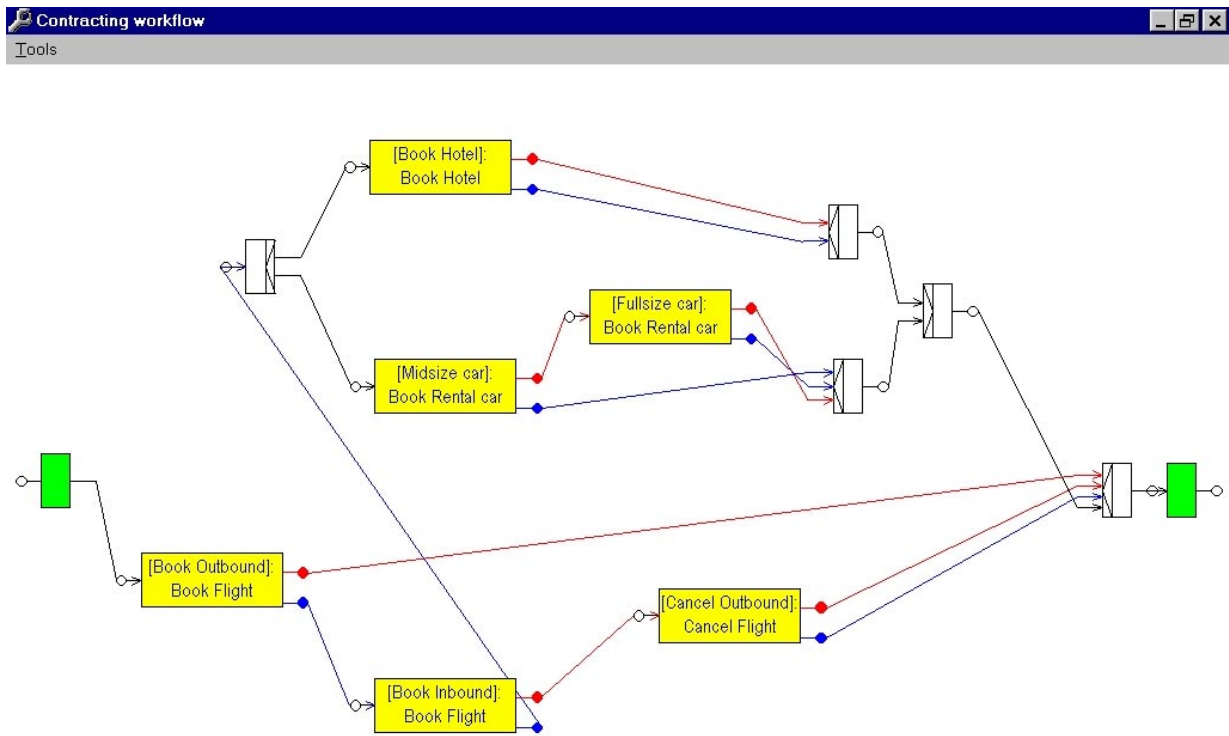


Figure 251 Example of the 'Contracting workflow' screen

Adding a candidate service type or a routing construct is done by clicking the right-mouse button at the desired location on the screen. The popup menu from Figure 252 appears in which the user selects the appropriate option. If he selects the option 'add candidate service', a selection list with all available service types appears from which one service type must be selected.

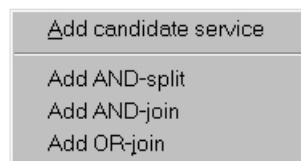


Figure 252 Popup menu to add tasks to the contracting workflow



Figure 253 Result of the 'add candidate service' operation

After a candidate service type is added to the graph, the user is able to perform operations on it by clicking the right mouse button above the candidate service type. The popup menu from Figure 254 appears.

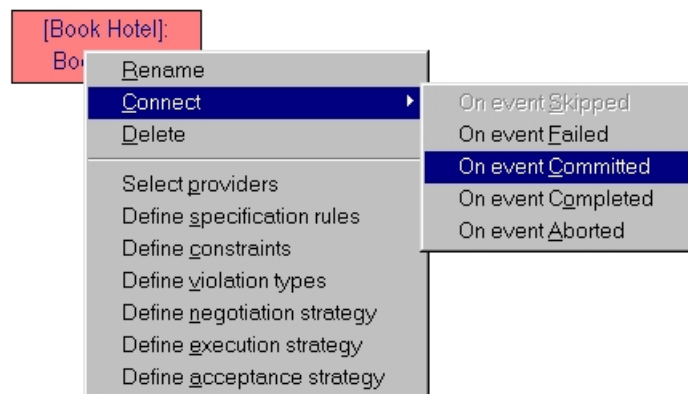


Figure 254 Popup menu with operations on a candidate service type

After a routing construct is added to the contracting workflow, the user is able to perform operations on it by clicking the right mouse button above it. The popup menu from Figure 255 appears.

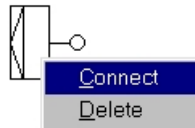


Figure 255 Popup menu with operations on an routing construct

Screen: Workflow analyser (Woflan)

When the user has completed the contracting workflow, he can use Woflan to check the soundness of the contracting workflow. An example of the Woflan user interface with the result of the diagnosis is given in Figure 256.

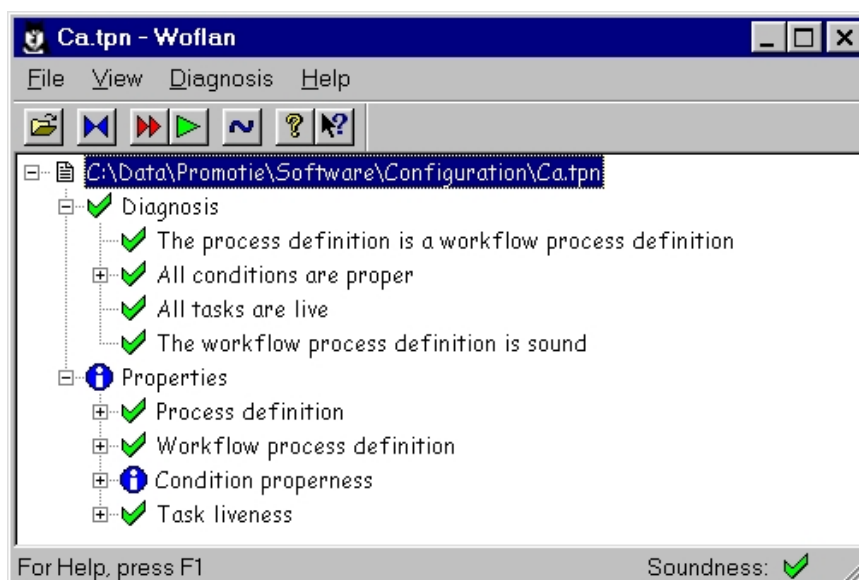


Figure 256 Example of the diagnosis result presented by Woflan

Screen: Selected providers

After defining a new candidate service type, the user can select the service providers that will be involved in the negotiation phase. Since each candidate service type refers to exactly one service type and each service type has one or more available providers defined, the screen in Figure 257 displays all available providers for the service type and lets the user make a selection. Furthermore, the user can make a ranking indicating the preference of a provider.

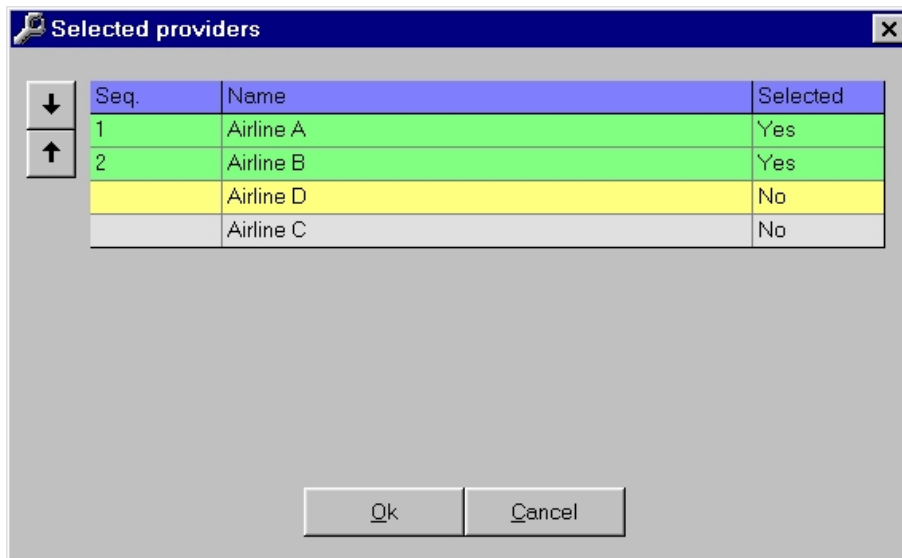


Figure 257 Example of the 'Selected providers' screen

Screen: 'Transformation editor'

When the candidate service types are defined, each candidate service type must be assigned specification rules used to create the initial value of the service data. The screen via which this is done is shown in Figure 258. A transformation is defined from a source hierarchic data structure to a target hierarchic data structure. In this case, the target data structure is the service type data structure, which is displayed in the grid in the upper part of the screen. The left column of the grid contains the entities and attributes in the hierarchic service type data model, whereas the right column contains an expression in which elements from the source data structure appear as operands. The available elements in the source data structure are listed in the listbox at the bottom of the screen. In this case, the source data structure consists of the case data and the service data of all candidate service types contracted before in the contracting workflow.

The user can assign an attribute from the source data structure to an attribute in the target data structure by double clicking the item in the listbox after which the item can be dropped on the target attribute.

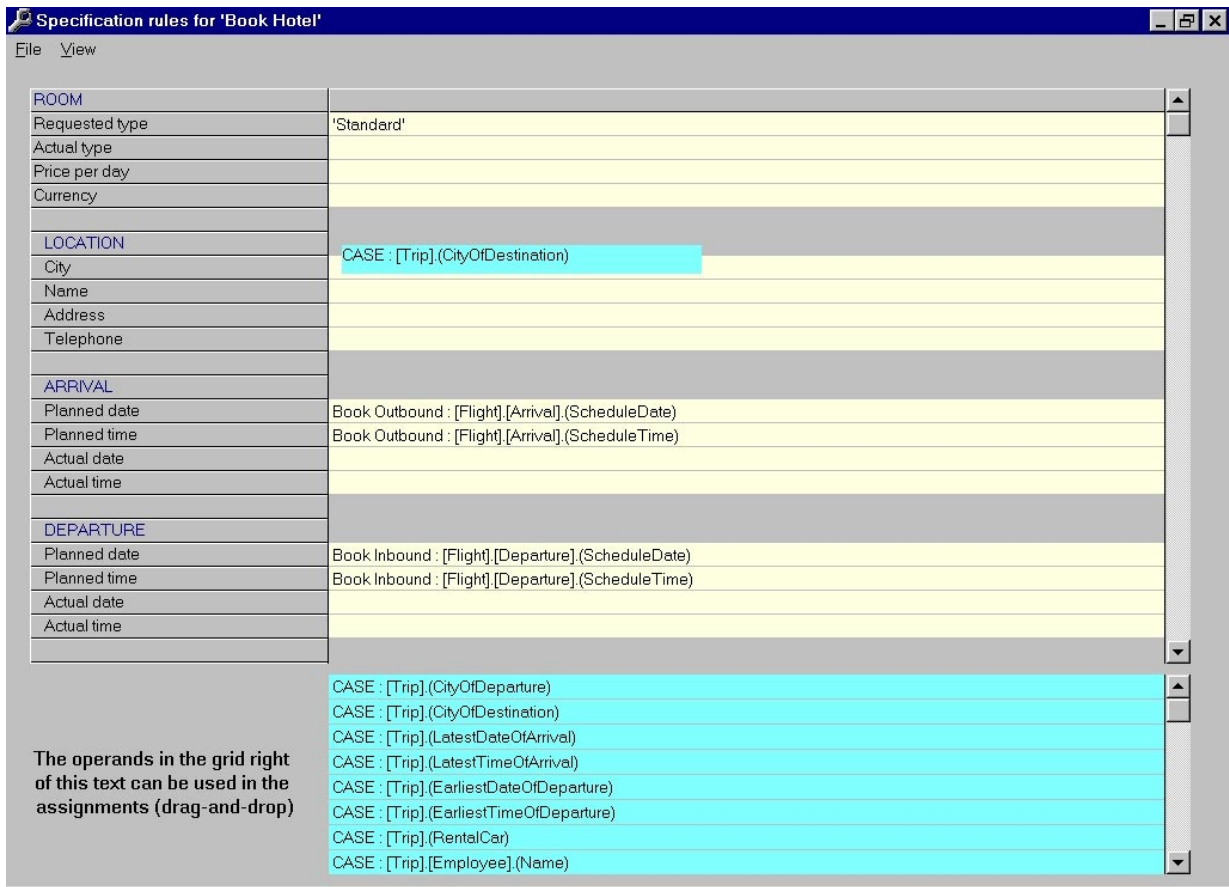


Figure 258 Example of the 'Transformation editor' screen

Screen: 'Constraints'

Each candidate service type can have a constraint that must be fulfilled in order to start the contracting process for the candidate service. Constraints are defined by the constraint editor, of which the layout has already been shown in Figure 247.

Screen: 'Strategy selection'

The negotiation, execution and acceptance phases of the contracting process for a candidate service are based on a selected transaction protocol. The next step is to define the structure of the negotiation, execution and acceptance processes, for which one or more strategies are available. This screen, of which the layout is shown in Figure 259, allows the user to select a strategy for a specific phase. When a selected strategy requires parameters, the 'next' button becomes enabled and invokes the strategy parameter editor.

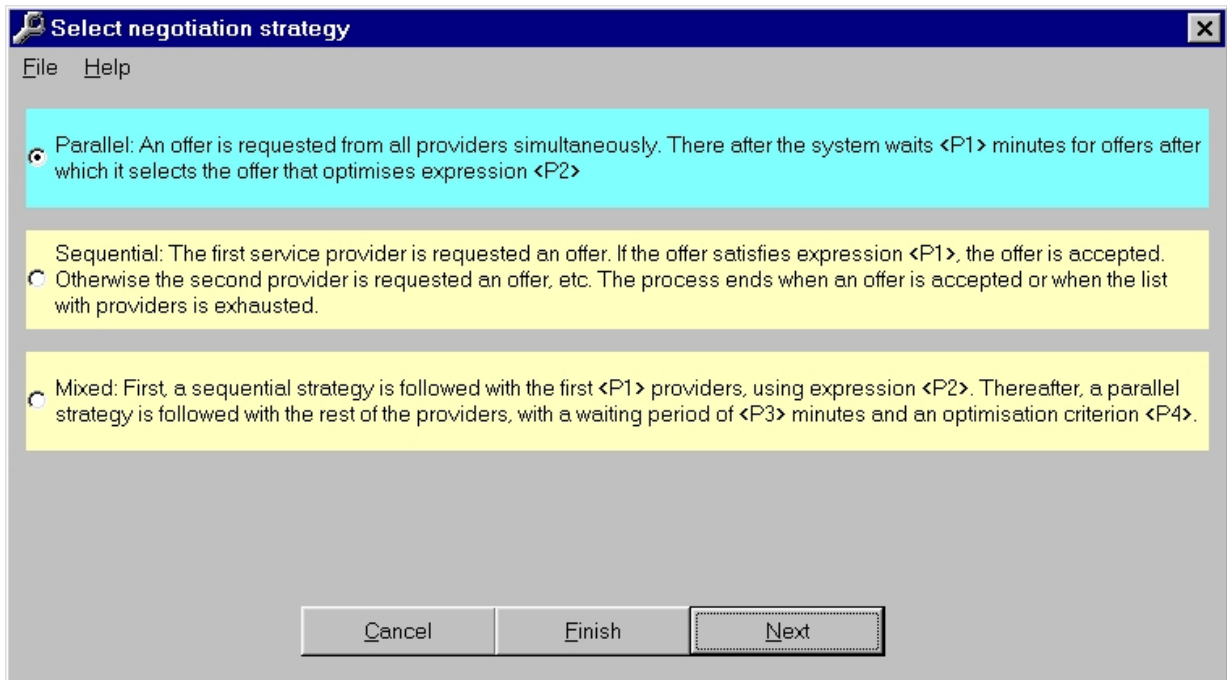


Figure 259 Example of the 'Strategy selection' screen

Screen: 'Strategy parameters editor'

This screen allows the user to enter parameters for a selected strategy. The layout of the screen is shown in Figure 260. Parameters can be expressions in which service attributes are used as operands.

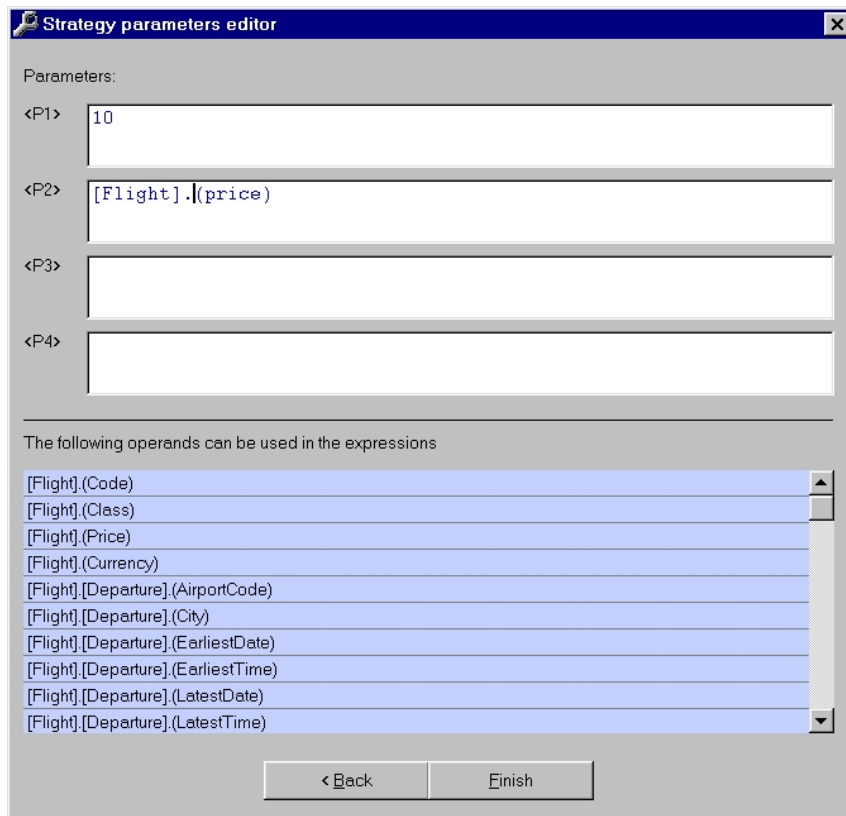


Figure 260 Example of the 'Strategy parameters editor' screen

4.6.3 Configuring the Server component

Screen: 'Server configuration'

The screen by which the user can submit the configuration parameters maintained in the Configurator component to the Server component is shown in Figure 261. Submitting configuration parameters to the Server component starts by checking the configuration data for completeness and correctness. If the check is successful, the user can click the 'send' button to actually submit the configuration parameters to the Server component.

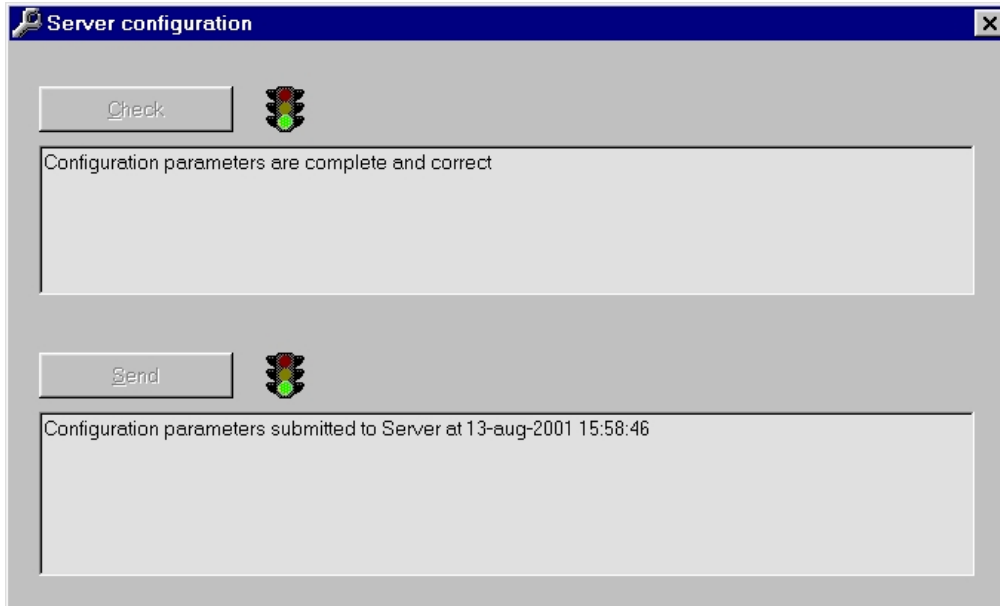


Figure 261 Example of the 'Server configuration' screen

4.6.4 Monitoring the Server component

The 'Monitor' component has no data storage of its own. It queries the Server component and presents the result in the user interface. The user interface is structured as a hierarchy of screens that is illustrated in Figure 262.

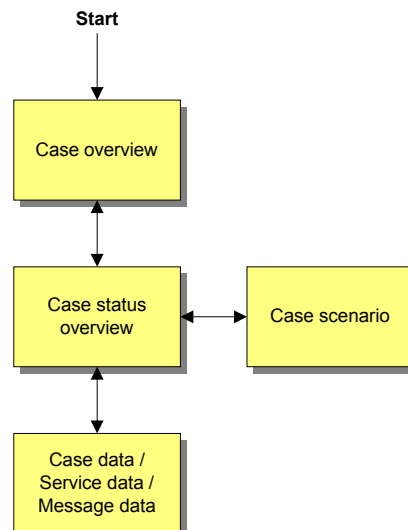
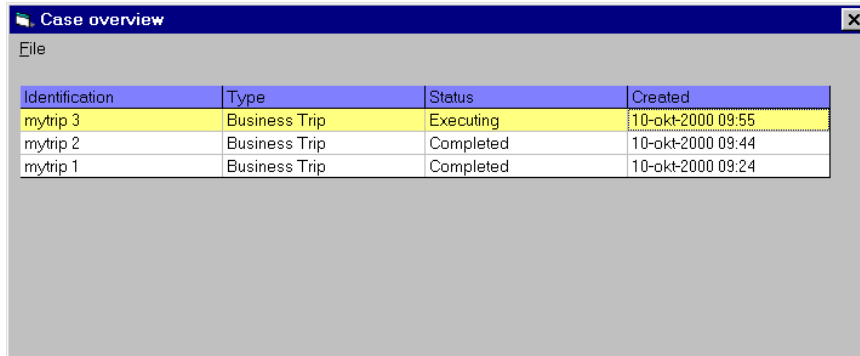


Figure 262 Screen hierarchy for the 'Monitor' component

Screen: 'Case overview'

The first screen gives the user access to an overview of business cases. The structure of the screen is given in Figure 263.







Identification	Type	Status	Created
mytrip 3	Business Trip	Executing	10-okt-2000 09:55
mytrip 2	Business Trip	Completed	10-okt-2000 09:44
mytrip 1	Business Trip	Completed	10-okt-2000 09:24

Figure 263 Layout of the 'Case Overview' monitoring screen

The user can double click a business case in the 'case overview' screen, after which the 'case status' screen appears (see Figure 264).

Screen 'Case status'

The 'case status' screen contains a tree with all data objects relevant to the service contracting process. The root object in the tree is always a *business case*, displayed by an  icon, followed by the case type description. The root object appears when the service contracting process is started. Nested under the business case node are the *candidate service nodes*. A candidate service node appears when the service data of a candidate service is specified during the specification phase of the service contracting process. It is displayed by an  icon, followed by the service type description. Nested under the service nodes are the *transaction nodes*. It is displayed by an  icon, followed by the name of the service provider. Nested under the transaction nodes are the *message nodes*. A message node appears when a message is sent or received in the business transaction that controls the service. It is displayed by an  icon, followed by an arrow to indicate the direction ('→' from client to provider, '←' from provider to client), the message type description and the date and time at which the message was sent / received.

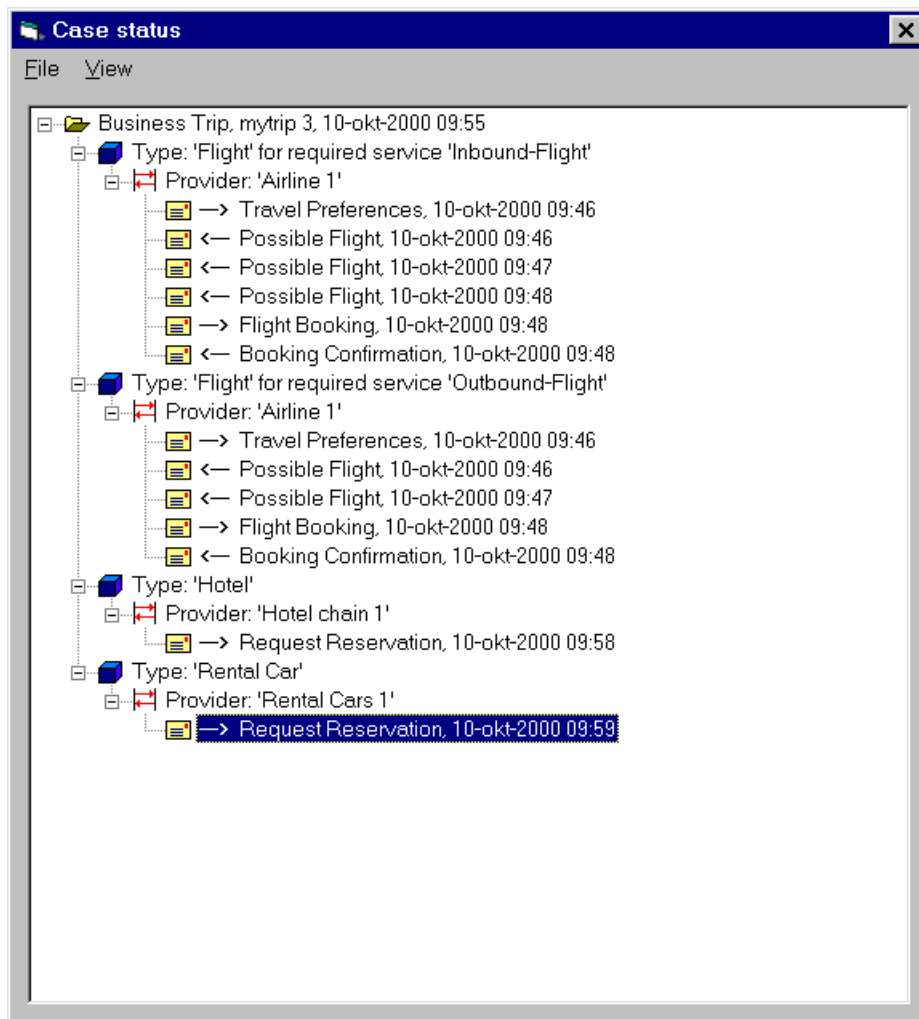


Figure 264 Layout of the 'Case Status' monitoring screen

Finally, the tree can contain *exception* nodes displayed by an **X** icon, followed by the description of the exception and the date/time at which the exception occurred. An exception node appears when an exception (e.g. data error, protocol error and violation) occurs in the service contracting process. The exception node is nested under the data object to which the exception applies. For instance, if a business case is rejected because of a missing attribute, the following situation appears.

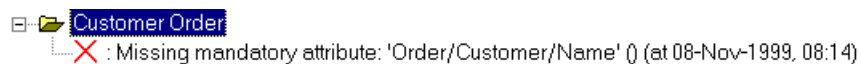


Figure 265 Examples of an exception for a business case

Screen 'Case scenario'

The 'case status' screen displays the exchanged messages nested under the services to which they belong. A different representation of the exchanged messages is a *message sequence chart*. An example of such a diagram is shown in Figure 266. The diagram contains a vertical bar for each candidate service. The service type description and service provider name is printed above each bar.

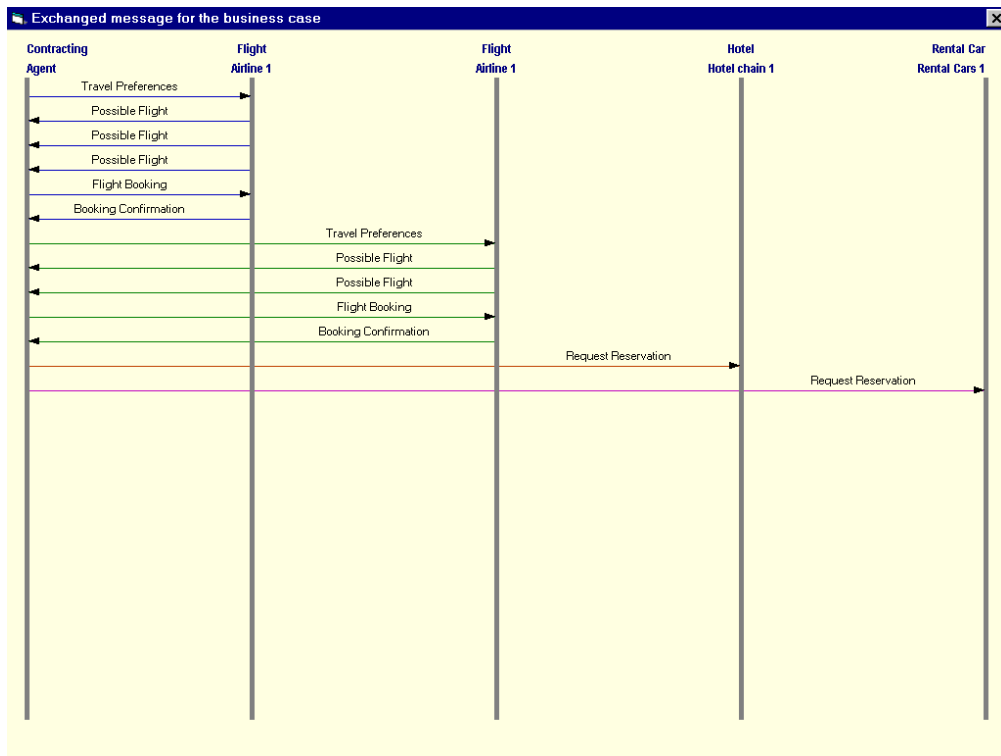


Figure 266 Layout of the 'Case scenario' monitoring screen

Screen 'Case data / Service data / Message data'

Each case, service and message has associated data that can be viewed via the Monitor component. To do this, the user can click the right mouse button on a case, service or message. There after, the case data, service data or message data is displayed via Internet Explorer. An example of the layout of case data is given in Figure 267.

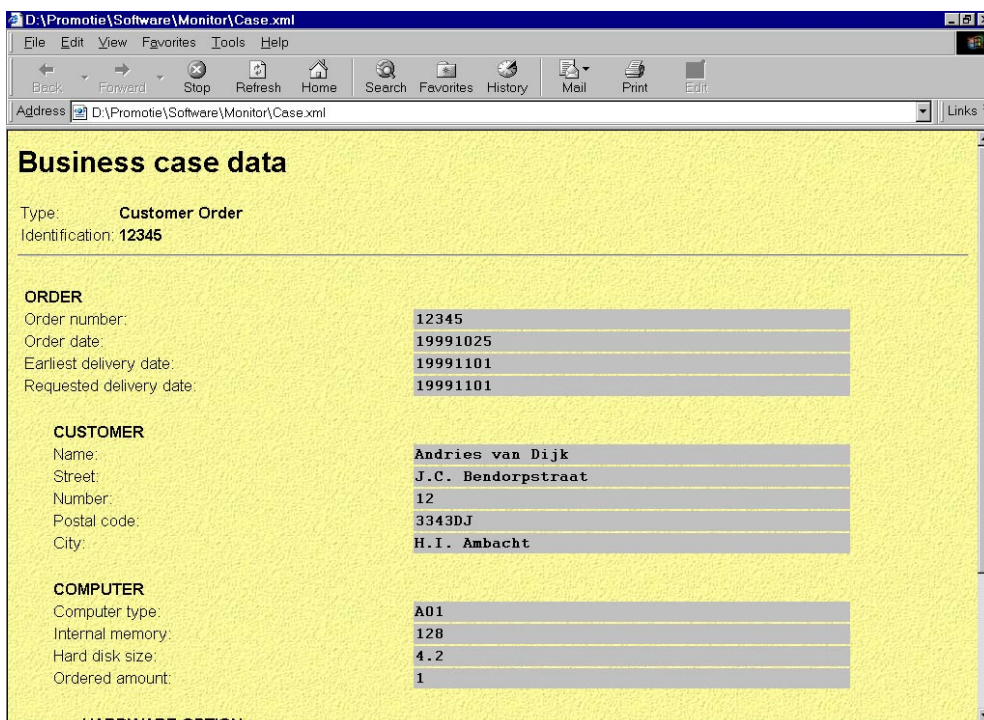


Figure 267 Layout of the 'Case data' monitoring screen

5. Evaluation

5.1 The research problem in retrospective

This research is about a new generic software component for electronic business based on outsourcing of work. During the last years, electronic business has received much attention. Companies have been thinking about the changes that electronic business can bring to their business. Many companies have devoted resources to electronic business development. New standards, like XML, have become available and new software suppliers emerged, bringing new standard software components for electronic business to the market. A result of these developments was an increasing level of maturity, seen from a technological point of view.

In the last decade, we have seen a shift in focus with respect to the use of the Internet. Companies who started to use the Internet used it as a public relations instrument mainly to provide static information via their websites. Later, companies added the element of *interaction* to their web-sites by integrating their web-applications with a database, hereby allowing the user to view more dynamic data. Next, we have seen a shift of focus from interaction to *transaction*. Companies recognised the Internet as a new distribution place and provided their customers with the ability to perform a transaction via the Internet. Today, we see another shift in focus from transaction to *integration*. Companies start to use the Internet to integrate their business processes hereby forming virtual organisations. One aspect of the integration of business processes will certainly be service contracting. The research problem and research objectives are therefore of high relevance for future electronic business developments.

One of the impacts of Internet technology to existing companies is a shift from *batch-oriented* processes to *online* processes. A front office that is available 24 hours a day during 7 days a week raises expectations of fast fulfilment processes at the customer. When a customer uses a website to order a product online, he does not want to wait weeks for a response. The ability to respond immediately to a customer request does not only require a well-organised front office, but a state-of-the-art back-office too. Transforming batch-oriented back-office processes to online processes is a major challenge to an organisation these days. Clearly, an environment where service providers are able to give an online response to requests of service clients is a condition under which a component like the Contracting Agent can prosper.

A second aspect of the research is component-based software development. During the last years, we have seen component-based development become mainstream ICT technology. A number of reasons can be found. First, there is a widespread acceptance of COM and CORBA as standard component frameworks. Second, the support of popular software development environments for these component frameworks brings component development within the reach of the mass of developers. Third, suppliers of large standard software systems like ERP or

CRM systems have used component technology to make their products open and extensible. The availability of middleware products makes it possible to integrate standard components efficiently, while maintaining flexibility. The rapid changing environment of organisations and the relative ease with which large components can be integrated with the rest of an organisations ICT infrastructure has contributed to the increasing tendency to use standard components instead of custom made software.

Concluding, we can say that the relevance of the research has only increased from the moment we started until this moment.

5.2 Achievements

Chapter 1 gave the background of the research and the research objectives. We will now turn in retrospect to these objectives and address the achievements of the research by discussing the research questions phrased in Section 1.6.4.

- **Research question 1: modelling service contracting processes as workflows**

We have given a conceptual framework for service contracting in Chapter 2. First, we defined underpinning concepts like workflow, service and business transaction. There after, we discussed frameworks for contracting processes found in literature. Next, we addressed the specification of *interface agreements* between parties involved in contracting processes, for which we have proposed a number of *transaction protocol patterns*. After having defined the interface agreements between service client and service provider, we focused on the specification of *contracting requirements* for outsourced tasks. Here after, we addressed the *contracting process* by defining the state data structure, a set of standard operations on the state data and the configuration parameters used by these standard operations. Next, we considered the *contracting workflow* as a high level coloured Petri net that defines the control flow of service contracting processes and in which standard operations on the state data are invoked. We proposed a set of *standard transitions* from which contracting workflows can be assembled. We used these standard transitions to create composite transitions that implement an entire negotiation strategy, execution strategy or acceptance strategy. Here after, we defined the rules according to which an entire contracting workflow can be *generated* from the contracting requirements on one hand and the standard transitions on the other hand.

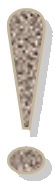


We have succeeded in modelling service contracting processes as workflows using the formalism of high level coloured Petri nets. Furthermore, we have shown how analysis techniques like soundness can be used to define correctness criteria for the contracting workflow and standard components from which the contracting workflow is composed.

- **Research question 2: flexibility**

The logical architecture of the Contracting Agent has been given in Chapter 3. The architecture identified the sub-components of the Contracting Agent, the structure of persistent data maintained by the components and the structure of data exchanged on the interfaces. We also defined the behaviour of each sub-component on its interface and the collaboration of the sub-components via their interfaces. An important characteristic of the logical archi-

ture was a clear separation of execution and control by using a workflow engine to control the behaviour of the Contracting Agent.



We have given a logical architecture in which the behaviour of the component is determined by a workflow engine, which is configured by an explicit model of the service contracting process in the form of a workflow net. This workflow net is a configuration parameter of the Server component. The flexibility of the component is therefore limited by the expressive power of the language in which the workflow net is expressed. However, since one workflow engine can be easily replaced by another, we have obtained the highest possible flexibility.

- **Research question 3: use of domain knowledge**

The conceptual framework showed how domain knowledge in the form of protocol patterns, contracting strategies and standard building blocks from which the contracting workflow can be assembled can successfully be used to define service contracting processes efficiently. Furthermore, new service types, service providers, protocol patterns and contracting strategies can be added to the configuration environment. With all this available, the user can define his service contracting process in minimal time. Finally, the ability of the Workflow Generator component to generate a complete workflow definition for the workflow engine used in the Server component demonstrated the advantages of using domain knowledge.



We have shown how domain knowledge can be used in the configuration user interface to assemble a contracting workflow in minimal time by selecting from an extensible repository with standard building blocks. Furthermore, we have shown how complex configuration parameters for the Server component (e.g. the workflow net for the workflow engine) can automatically be generated from the contracting requirements entered by the user.

- **Research question 4: use of standard software components**

The technical architecture of the Contracting Agent identified commercial-of-the-shelf (COTS) components and custom-made components. Although we did not use many COTS components, we used them for the most complex functions in the Contracting Agent. First, we used MSXML4.dll for all operations on XML documents: *parsing* (with the DOM), *validation* (with XML-Schema), *transformation* (with XSLT) and *presentation* (with XSL). The functionality of the Interaction Manager could therefore be minimal and implemented in very few lines of code. The second COTS component was the ExSpect workflow engine. The ability of ExSpect to execute high level coloured Petri nets allowed us to define the structure of the implementation contracting workflow as a Petri net, as we also did with the conceptual contracting workflow.



We have shown that standard software components can be used for all important functions of the Contracting Agent.

Concluding, we can say in retrospect that we have achieved the research objectives stated in Chapter 1.

5.3 Contribution

This research focuses on a specific class of contracting processes, as demarcated in Section 1.6.2. The business objective of this research is to contribute to the efficiency of organisations by providing an optimal support for the demarcated class of service contracting processes with information and communication technology. The claim of this research is to have provided:

- the *explication* of the demarcated area of electronic contracting by providing a conceptual framework.
- a *specification language* for contracting requirements of outsourced tasks;
- a set of *standard transitions* from which contracting workflows can be composed;
- a mechanism by which sound contracting workflows can be *generated* from contracting requirements;
- an *architecture* (logical and technical) of a software component that supports the demarcated class of service contracting processes;
- a *proof of concept* of the conceptual framework, the specification language and the architecture in the form of a working software component.

We will now compare our work to the work of others, for instance the work that has been discussed in Section 1.4. When we compare our work to the research related to the *Language Action Perspective*, we see a difference in the techniques used to define the interface agreements between service client and service provider, for which we have used Petri nets. Weigand et al [139] argue that deontic logic is more suitable for modelling electronic commerce processes than Petri nets because concepts like ‘obligation’ and ‘permission’ are part of the formalism intrinsically and make it possible to reason about properties of electronic contracting processes. Although we agree with the authors that deontic logic has advantages over Petri nets, we made a deliberate choice to use the formalism of Petri nets though. The major reason for this choice was that *reasoning* about transaction protocols is outside the scope of this research, we just need a formalism to *express* transaction protocols. On the other hand, the formalism of Petri nets has the advantages of a formal basis (e.g. to prove the soundness property) and a graphical representation that is easily understood. Finally, the formalism of Petri nets has a great fit with our approach of using workflow management principles for electronic contracting.

The work on *Documentary Petri Nets* by Lee et al [89, 92] is a research area closely related to our research. An advantage of the DPN formalism is its ability to model the exchange of information, goods and money in trade procedures. A DPN can be seen as a workflow that is distributed among a number of organisations. The InterProcs [90] system shows how an organisation can download a DPN and execute the part corresponding to his own role. When we compare our approach to the DPN approach, there is however a difference. First, we consider message exchange only and ignore the exchange of goods and money. Second, in our approach service clients and service providers agree on a common transaction protocol, where after each party has to define his internal procedures. Because the internal procedures are kept private, an organisation can change his internal procedure easily, as long as he conforms to the transaction protocol. Clearly, a disadvantage of the approach is that each party must define an internal procedure, which has the potential of bringing a lot of work to each party. Therefore, our research focuses on *generating* contracting workflows (i.e. the procedure of a service client).

The work on verification of *inter-organisational workflows* by Van der Aalst [9] shows the importance of applying analysis techniques to avoid anomalies such as livelocks or deadlocks. We have used standard analysis techniques to the contracting workflow to prove that it is sound.

Finally, we ask ourselves the question what is the added value of a dedicated component for service contracting when we see generic workflow enabled (XML) messaging tools emerge in the market. These tools offer a broad range of functions for inter-organisational message exchange, and a workflow engine on top of that. The user can define his own workflow, in which he can trigger outbound messages and respond to inbound messages. Message data is created and processed by applications that are invoked by the workflow engine. Clearly, these platforms are powerful tools for organisations that want to implement inter-organisational processes, and could be used for service contracting processes too. However, although service contracting processes *can* be implemented in generic workflow enabled XML messaging tools, this does not mean that it can be done *efficiently*. Our research shows how domain knowledge of service contracting processes such as protocol patterns and contracting strategies can be used as the vocabulary used to specify the contracting requirements in only a few parameters, from which the entire contracting workflow can be generated. Therefore, the added value of the Contracting Agent is (i) in the Configurator component that makes maximum use of domain knowledge by generating the entire contracting workflow and (ii) in the Server component that contains sub-components for storing the state data of service contracting processes and sub-components for performing the standard operations of which a service contracting process consists.

5.4 Business opportunities

From the start of this dissertation until this point, we have considered the design of the conceptual framework for service contracting and the architecture of a software component for service contracting as an academic exercise only. We will now view electronic service contracting from a business point of view: how can a company make business out of service contracting, and what kind of organisations are likely to enter these markets.

- **Become a software vendor**

The first option to make business out of service contracting is by selling software licenses to service clients. We distinguish two markets for software vendors. The first one is the market for service contracting components with extensive functionality to be integrated with client applications and message exchange components. The second market is the market for add-in components like constraints editors, transformation editors, strategy editors and strategy processors. Future vendors of service contracting components are likely to emerge from two directions. First, current suppliers of generic workflow enabled XML messaging tools can extend their products with functionality to support specific types of processes, of which service contracting could be one. Second, new third party suppliers may emerge who would aim at offering a best-of-breed Configuration component that can be used in combination with existing workflow enabled XML messaging tools easily.

- **Become an intermediary**

When service clients outsource the execution of one or more tasks to external service providers, they can perform the necessary service contracting processes themselves. However, a service client can also outsource the execution of his service contracting process to an in-

termediary. The intermediary may offer the functionality of the Contracting Agent via a web-interface and operate a Contracting Agent himself. An intermediary like this could make a profit from the margin between the price at which he buys services from service providers and the price he charges the service clients for services. This requires the intermediary to establish long term relationships with a number of service providers that grant him a discount in turn for the business he brings them.

- **Offer application consultancy**

Current state-of-the-art software components like ERP or CRM systems are highly configurable. The number of configuration parameters can be up to tens of thousands and requires specialists to tailor a generic component to the needs of a specific business situation. Application consultancy involves making an inventory of business needs, mapping these business needs into parameter settings of the component, entering the required parameter settings in the component and programming additional functions in for instance COM components.

- **Offer web-service engineering consultancy**

In this research, we have focused on the service client and the service contracting process performed by him. However, this requires a service provider that is able to offer his services to the market via a business-to-business electronic commerce interface. A consultancy firm can provide the following services to a service provider:

- define his service types (schema, transaction protocol, and message types) and publish these in one or more repositories;
- install and configure a business-to-business electronic commerce infrastructure by which business transactions with service clients can be performed via a variety of standards;
- design and implement the integration between the business-to-business electronic commerce front-end systems and the back-office systems used for the fulfilment processes (planning applications, etc.).

5.5 Conclusions

- **Conclusion 1: Service contracting as a generic function**

The research was built on the hypothesis that service contracting is a generic function, for which a generic software component can be developed. With the creation of the conceptual framework for service contracting and the logical and technical architecture of the Contracting Agent, we have proven this hypothesis true. We have shown how domain knowledge like transaction protocol patterns and contracting strategies can be used as configuration parameter of the configuration function.

- **Conclusion 2: Contracting Agent as a generic component**

Although service contracting processes can be implemented in a variety of software products (a C-compiler would be sufficient), we have shown the added value of a dedicated component for service contracting, especially in the configuration and monitoring functions. We therefore conclude that there is at least a market for dedicated configuration and monitoring tools that operate in conjunction with a more generic engine, e.g. a workflow enabled XML messaging tool.

- **Conclusion 3: Integration of e-business and workflow management**

The component on which the research focuses can be characterised by both the terms ‘electronic business’ and ‘workflow management’. One could view the Contracting Agent as an electronic business component with embedded workflow technology. It is however also possible to view the Contracting Agent as a workflow application for a specific purpose, i.e. electronic business based on outsourcing of work. Clearly, electronic commerce and workflow management is showing an increasing overlap. This is illustrated by the attention of the Workflow Management Coalition for electronic business [143, 144]. Currently available workflow enabled XML messaging tools show that the integration of e-business and workflow is to stay and set the standard for an entire class of software products.

- **Conclusion 4: Aptitude of standard software components**

From an architectural point of view, we have designed the Contracting Agent as a workflow application, separating execution from control. In the construction of the component, we used the ExSpect COM-server as workflow engine. Looking back on the experiences with the component, we conclude that this has been a good choice seen from the perspective of creating a proof of concept. An appealing feature of the ExSpect COM-server is the use of high-level coloured Petri nets as process modelling technique. These Petri nets have proven to be a powerful technique for modelling service contracting protocols. Another appealing feature of the ExSpect COM-server is the API. A client application can start a process, create tokens in places and retrieve tokens from places. This interface has proven to be extremely flexible, yet still easy to use by a programmer. These features, together with the simplicity by which a COM-component can be assembled with other components, makes it a good choice for a light-weight workflow engine invisibly embedded into another product. The other major COTS component, MSXML4.dll, proved to be a very powerful component, of which the usefulness can hardly be underestimated. The support of industry standards like XML, XML-Schema, XSL and XSLT makes it a component that provides functionality to applications for every standard operation to XML documents: creation, parsing, validation, transformation and presentation. Since these operations are of a complex nature, the added value of this small component is large.

5.6 Directions for future research

- **Research topic: ad-hoc contracting workflows**

In this research, we focused on service contracting processes that were ‘repeating’ in nature (see Section 2.3.1). Therefore, we were able to define the contracting requirements for all business cases belonging to one business case type during configuration time. The contracting workflow itself is static; it is created once and then used for each business case of a specific case type. A different approach would be to define the service contracting workflow on the fly during run time. For this purpose, the Contracting Agent should know the demands of each specific business case and the available service types on the market. The latter can for instance be obtained from an on-line catalogue of service types and service providers on the Internet. If both demand and supply are described in a structured way, *automatic matchmaking* could be performed. The results of the matchmaking process can then be used to construct a contracting workflow for a specific business case dynamically.

- **Research topic: updates**

In this research, we did not allow case data to be *updated* after the service contracting process had been initiated. An extension would therefore be to allow case data to be updated *during* the service contracting process. These changes can have significant consequences for a service contracting process. Service data of candidate services will have to be specified again with the new case data. If negotiations have already started and the changes are relatively small, an update of the service contract can be requested. If the update is not accepted, or if the changes are too large, an established contract may have to be aborted. The ability to allow updates is desirable for two reasons. First, business practice shows that unforeseen circumstances always will occur, inevitably leading to changes in the case data of business cases being processed. Second, a business process can be designed such that the value of case data attributes becomes available at different moments. For example, when a shipper makes a transport booking, he may know the place and time of loading, the place and time of delivery and the container type, but not the container number. In this case, it should be possible to start the service contracting process with case data without a container number. When the container number becomes known, the client application sends the additional information (container number) as an update to the Contracting Agent.

- **Research topic: aborts**

In this research, we did not allow a service contracting process to be *aborted* after the service contracting process had been initiated. An extension would therefore be to allow a client application to abort a running service contracting process. This may require to cancel all negotiation phases that did not end in a service contract yet and to cancel all established service contracts. However, it is possible that a service contract can not be cancelled, either because the transaction protocol does not offer this function, or because the execution of the service can not be stopped any more. Finally, abortion of a running service contracting process may require compensating actions to be taken to undo the effects of services that have already been executed.

- **Research topic: negotiation mechanisms**

This research focused on service contracting in a situation of *partial knowledge*, i.e. the service client does not know in advance if a specified service can be provided by a service provider. Since each service provider has a limited capacity, service clients will sometimes request services that are not feasible for the service provider. In many cases however, the service provider would be able to provide a service that is not equal to the requested service, but very much alike. This may lead to a *negotiation* between service client and service provider. The objective of the negotiation is to find a specification of a service that is feasible for both service client and service provider. In this research, we described simple negotiation mechanisms, which will be too limited for some business cases. Therefore, efforts should be made to develop more powerful negotiation mechanisms, which in turn can be integrated in the contracting workflows as standard building blocks.

- **Research topic: integration of financial settlement**

In Chapter 2, we discussed a pattern for business transactions, consisting of four phases: specification, negotiation, execution and acceptance. At some point during these phases, there can be a financial settlement, which was deliberately excluded from the domain of our research. A minimum requirement for support of financial settlement is an interface by which the client application can retrieve the financial obligations, in order to be settled by one of the business applications. A more complex solution is to include financial settlement in the Contracting Agent itself, using existing industry standards for electronic payment.

5.7 Concluding remark

This research has brought a new generic software component that contributes to the integration of business processes via electronic business. Considering the efforts that still have to be made, both from a business point of view and from a technological point of view, our research was only a brief encounter with the challenging area of service contracting.

A. Modelling techniques

A.1 Functional data modelling

Functional data modelling is a technique to model the structure of objects, described by Van Hee [61]. A functional data model consists of simple objects called *simplexes*, which are mutually related by *relationships*. Simplexes have the following characteristics:

- a simplex is *atomic* (it has no internal structure);
- a simplex is *distinguishable*;
- a simplex belongs to a type, called a *simplex class*;
- a simplex class can be named by a noun.

Examples of simplexes are ‘Andries’, ‘Heleen’, ‘Rachel’ and ‘Loï’s’, all belonging to the simplex class ‘Person’. A simplex is called an *instance* of a simplex class. A relationship is characterised by a pair of simplexes and a name. Relationships also belong to a type: the *relationship class*. A relationship is called an instance of a relationship class. A relationship class is determined by its name and connects a simplex of one simplex class with a simplex of another (or the same) simplex class. Each relationship has also a *direction*, indicated by an arrow. We call the simplex class at the tail of the arrow the *domain* class and the one at the head of the arrow the *range* class. An example of a relationship class is ‘is_father_of’, to which the following two relationships belong: (‘Andries’, ‘Rachel’) and (‘Andries’, ‘Loï’s’). A structured set of simplexes and relationships is called a complex object, or *complex*. The set of all possible complexes with the same structure is called a *complex class*. A complex class is defined by its simplex classes and the relationship classes.

Simplex classes and relationship classes are defined in an object model, which has the structure of a labelled graph. A simplex class is denoted by a node in the form of a rectangle. A relationship class is denoted by labelled and directed lines known as edges. For example:

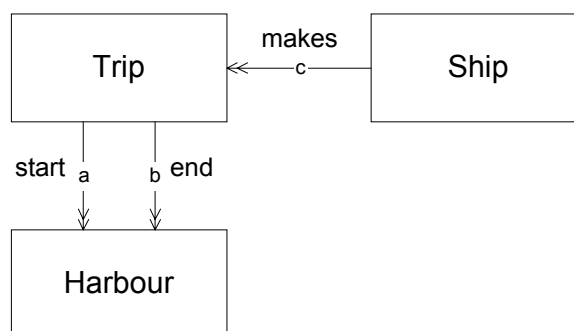
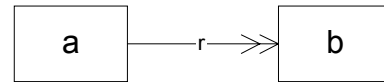


Figure 268 Example of an object model (after [61]).

We often want to impose *constraints* on relationships. A number of frequently used constraints can be represented in the object model. We will use the example of two simplex classes, *a* and *b*, connected by a relationship class *r*.

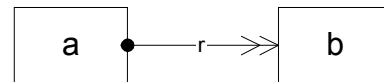
- *unconstrained*

A relationship without constraints is denoted by an arrow with an open double arrowhead.



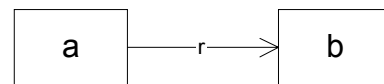
- *total*

A relationship is called *total* if every simplex of class *a* is associated with at least one simplex of class *b*. Totality is denoted by a solid circle at the tail of the arrow.



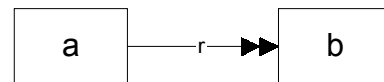
- *functional*

A relationship is called *functional* if there is at most one simplex of class *b* connected to each simplex of class *a*. Functionality is denoted by a single arrowhead.



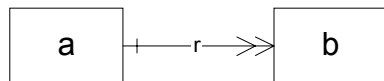
- *surjective*

A relationship is called *surjective* if every simplex of class *b* is connected to at least one simplex of class *a*. Surjectivity is denoted by a solid arrowhead.



- *injective*

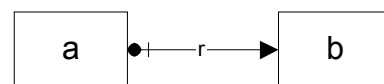
A relationship is called *injective* if every simplex of class *b* is connected to at most one simplex of class *a*. Injectivity is denoted by a vertical bar at the tail of the arrow.



A relationship can have any combination of these four properties. A relationship that has all four properties is called a *bijection* relationship.

- *Bijection*

A relationship is called *bijection* if every simplex of class *a* is connected to exactly one simplex of class *b*, and if every simplex of class *b* is connected to exactly one simplex of class *a*. Bijection is denoted by a solid circle and a vertical bar at the tail of the arrow and a solid, single arrowhead.

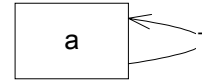


Finally, since hierarchic data structures are used frequently in this dissertation, we introduce another standard type of constraint. The constraint is used to model a hierarchical relationship (or tree structure) between simplexes of one simplex class. A set of simplexes has a tree struc-

ture if one simplex is the root of the tree and all other simplexes are nested under another simplex. In other words, one simplex has no father and all other simplexes have one father.

- *Tree*

A relationship class r from domain class a to range class a models a tree structure if (i) exactly one simplex of class a does not appear as domain in a relationship of class r (ii) all other simplexes of class a appear as domain in exactly one relationship of class r and (iii) the network defined by the relationships does not contain cycles. We will use a letter T in the arrow to indicate a relationship with a tree constraint.



In the drawing technique we introduced before, each simplex class was represented by a rectangle. We will now introduce different representations for different types of simplex classes. First, we will use the term *attribute* for simplex classes that only occur as a range class in relationship classes. Attributes will be represented by a rounded rectangle. All other simplex classes are called *entities*. An entity that only occurs as a domain class in relationship classes of which the range class is also an entity is called an *association*. An association is represented by a diamond.

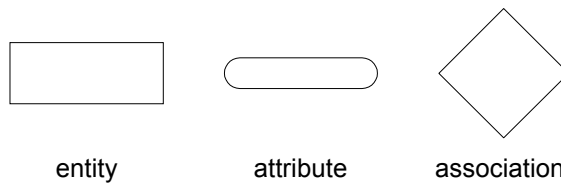


Figure 269 Graphical symbols for simplex classes

An example of the drawing technique is given in Figure 270. There is an entity ‘article’ with three attributes ‘identification’, ‘name’ and ‘price’. The association ‘component’ models a hierarchic relationship between articles: an article can be composed of other articles. Association ‘component’ is the domain class in two relationship classes. The range class of the relationships ‘is_part_of’ and ‘is_composed_of’ model the child article and the parent article respectively.

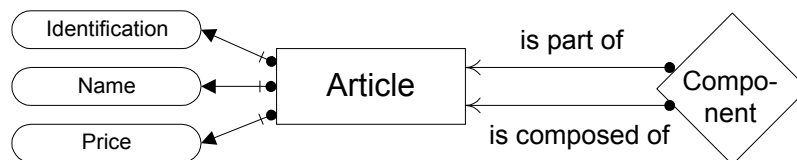


Figure 270 Example of the drawing technique

A.2 High level coloured Petri nets

The formalism of Petri nets was introduced by Carl Adam Petri in 1962 [103]. This classical Petri net is a directed bipartite graph with two kinds of nodes: *places* and *transitions*. A synonym for transition is *processor*. The nodes are connected by directed arcs called *connectors*. A connector can only connect a place to a transition or a transition to a place. Places can contain *tokens* and transitions can consume and produce tokens. A special kind of place called *store* always contains one token. A transition that consumes a token from a store always produces a token to the store again. The way in which a transition consumes and produces tokens is well defined. A transition is called *enabled* when all input places contain at least one token. If a transition is enabled, it *fires* immediately. A transition that fires consumes one token from each of its input places and produces one token in each of its output places.

One of the strong points of the formalism is the graphical representation of which an example is given in . Places are drawn as circles, transitions as rectangles, connectors are drawn as arrows and tokens are drawn as bullets. Figure 271 shows a Petri net with two transitions, four places and six connectors. Place *a* is called an *input place* of transition *r* and place *b* is called an *output place* of transition *r*. A place can be both an input place and an output place, like for instance place *d*. Although Petri nets can be represented graphically, they have a sound formal basis. The formal basis of Petri-Nets makes it possible to *prove* properties of the process that is modelled by a Petri-Net.

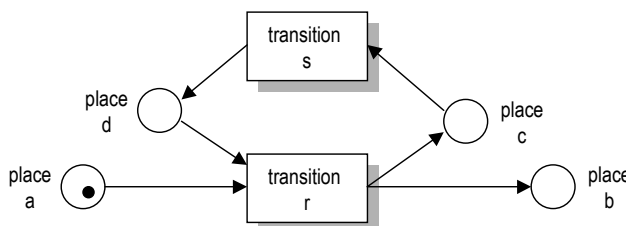


Figure 271 Example of a classical Petri net

Although classical Petri nets proved to be a powerful formalism, its expressive power fell short for some real-life problems. A number of extensions have therefore been introduced and Petri-Nets with these extensions are called *high-level* Petri-Nets. A full definition of high-level Petri-Nets is given by Jensen [77] and Van Hee [61]. We will discuss (i) the extension with colour, (ii) the extension with time and (iii) the extension with hierarchy.

The extension with colour

In a classical Petri-Net, two tokens can not be distinguished from each other. If tokens are used to model objects in the real world however, one often wants to include properties of objects in the model. In a high-level Petri-Net, tokens can have a *value*, also called the *colour* of the token. The colour of a token is modelled as a complex object, or *complex*, according to the formalism of functional data modelling (appendix A.1). We can use the colour of a token in the firing rules of a transition by specifying a *precondition*. A precondition for a transition consists of constraints to the colour of the tokens that are consumed from the input places. If a precondition is specified, the transition will only be enabled if the colour of the tokens in the input places satisfies the constraints in the precondition. If a transaction is enabled, it fires. The colour of the produced tokens can be different from the colour of the input tokens. Unlike the classical Petri-Net, where one token is produced in each output place, high-level Petri-Nets allow the number of tokens produced in each output place to be determined during the firing of

a transition. The *postcondition* of a transition determines (i) the number of tokens to be produced in each output place and (ii) the data transformation rules to create the colour of the produced tokens out of the colour of the consumed tokens.

In the approach of Van Hee [61], the formalisms of Petri nets and functional data modelling are integrated. Figure 272 shows the relationships between concepts of Petri nets and functional data modelling. A Petri-Net consists of *transitions*, *places* and *connectors*. A place can contain one or more *tokens*. Each token has a *colour* expressed as a *complex*. A complex is a network of *simplexes* and *relationships*. The structure of a complex is defined by a *complex class*, which is a network of *simplex classes* and *relationship classes*. Each simplex belongs to one simplex class and each relationship belongs to one relationship class.

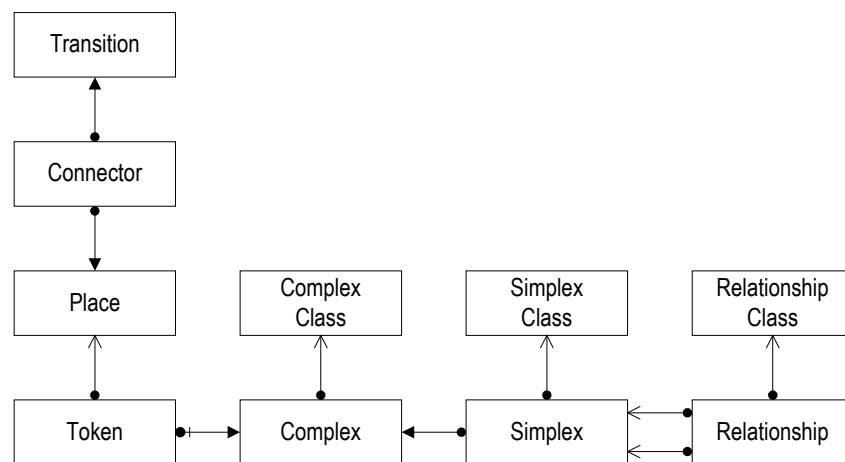


Figure 272 Petri nets and functional data modelling

The extension with time

To be able to model the behaviour of a process in time, high-level Petri-Nets are extended with time. Tokens in a high-level Petri-Net have a *timestamp* that indicates the time at which the token is available for consumption by a transition. A transition is only enabled when the timestamp of all tokens to be consumed is smaller than or equal to the current time. The *enabling time* of a transition is the value of the highest timestamp in the tokens to be consumed. If two or more transitions are enabled, the transition with the lowest enabling time fires first. If more than one token is available in a place, tokens are consumed in the order in which they became available. The token with the lowest timestamp is always consumed first. If a transition fires, the timestamp of the produced tokens will have a value equal to or larger than the time of firing. A token that gets a timestamp equal to the time of firing it is available to other transitions immediately. A token that gets a timestamp higher than the time of firing is only available after a *delay*. Although the firing of a transition does not use time, time-consuming processes can be modelled by a transition of which the produced tokens have a delay equal to the time required for the process.

The extension with hierarchy

Although Petri-Nets with colour and time can be used to model very complex processes, the result will always be one large Petri-Net that is often too large to be understood easy. A good approach to deal with complexity is to view a large complex process as composed of smaller sub-processes, which can be divided into smaller sub-processes again. This is why high-level Petri-Nets are extended with hierarchy. A transition can be decomposed into a network of places, transactions and connectors. A transition that can be decomposed into a net is called a

non-elementary or *composite* transition (we will also use the term *system* for composite transitions). Otherwise, a transition is called an *elementary* transition. A Petri-Net is called a *flat net* if all transitions are elementary transitions. In this dissertation, we will use the following conventions for the graphical representation of high-level Petri-Nets.

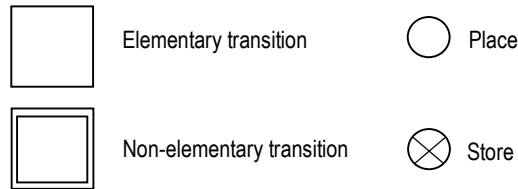


Figure 273 Graphical conventions for Petri-Nets

Furthermore, elementary transitions can have access to a real time clock to create a time stamp on a token and to compare the time stamp on a token in an input place with the current time (precondition). If a transition has access to a real time clock, it is presented as shown in Figure 274.



Figure 274 Graphical conventions for a transition with access to a real time clock

A.3 EBNF

This dissertation uses an Extended Backus-Naur Form (EBNF) notation for the definition of formal grammars. A grammar consists of one or more *rules*, each of which defines a *symbol* in the form of an *expression*.

symbol := expression

Within the expression on the right-hand side of a rule, the following patterns are used. The symbols *A* and *B* represent simple expressions.

Pattern	Matches
A B	A followed by B.
A B	A or B but not both.
[A]	A or nothing.
A+	One or more occurrences of A.
A*	Zero or more occurrences of A.
"string"	A literal string matching that given inside the double quotes.
'string'	A literal string matching that given inside the single quotes.
(expression)	Expression is treated as a unit and may be combined as described in this list.

References

1. AALST, W.M.P. VAN DER, *Timed coloured Petri nets and their application to logistics*, PhD thesis, Eindhoven University of Technology, Eindhoven, 1992.
2. AALST, W.M.P. VAN DER, K.M. VAN HEE AND G.J. HOUBEN, *Modelleren en analyseren van workflow: een aanpak op basis van Petri-netten*, *Informatie*, 37(11), pages 590-599, 1995 (in Dutch).
3. AALST, W.M.P. VAN DER AND K.M. VAN HEE, *Business Process Redesign: A Petri-net-based approach*, *Computers in industry*, 29(1-2), pages 15-26, 1996.
4. AALST, W.M.P. VAN DER, *Three Good Reasons for Using a Petri-net-based Workflow Management System*, in S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179-201, Camebridge, Massachusetts, November 1996.
5. AALST, W.M.P. VAN DER, *Verification of Workflow Nets*, in P. Azéma and G. Balbo, editors, *Application and Theory of Petri Nets 1997*, volume 1248 of *Lecture Notes in Computer Science*, pages 407-426, Springer-Verlag, Berlin, 1997.
6. AALST, W.M.P. VAN DER, *The Application of Petri Nets to Workflow Management*, *The Journal of Circuits, Systems and Computers*, 8(1), pages 21-66, 1998.
7. AALST, W.M.P. VAN DER, *Interorganizational Workflows: An Approach based on Message Sequence Charts and Petri Nets*, *Systems Analysis – Modelling – Simulation*, 34(3), pages 335-367, 1999.
8. AALST, W.M.P. VAN DER, *Process-oriented Architectures for Electronic Commerce and Interorganizational Workflow*, *Information Systems*, 24(8), pages 639-671, 2000.
9. AALST, W.M.P. VAN DER, *Workflow Verification: Finding Control-Flow Errors using Petri-net-based Techniques*, in *Business Process Management: Models, Techniques and Empirical Studies*, volume 1806 of *Lecture Notes in Computer Science*, pages 161-183, Springer-Verlag, Berlin, 2000.
10. AALST, W.M.P. VAN DER, *Inheritance of Interorganizational Workflows: How to Agree to Disagree Without Loosing Control?*, BETA Working Paper Series, WP 46, Eindhoven University of Technology, Eindhoven, 2000
11. AALST, W.M.P. VAN DER AND T. BASTEN, *Inheritance of Workflows: An approach to tackling problems related to change*, BETA Working Paper Series, WP 50, Eindhoven University of Technology, Eindhoven, 2000.

12. AALST, W.M.P. VAN DER AND K.M. VAN HEE, *Workflow Management: Models, Methods and Systems*, MIT Press, Cambridge, MA, 2001.
13. AALST, W.M.P. VAN DER AND M. WESKE, *The P2P approach to Interorganizational Workflows*, in Proceedings of the 13th International Conference on Advanced Information Systems Engineering (CAiSE'01), volume 2068 of Lecture Notes in Computer Science, pages 140-156, Springer-Verlag, Berlin, 2001.
14. ARIBA, *cXML User's Guide version 1.2*, www.cxml.org, February 2001.
15. BERGSTRA, J.A. AND P. KLINT, *The toolbus: a component interconnection architecture*, Technical report P9408, University of Amsterdam, 1994.
16. BLOMMENSTEIN, F. VAN, *Electronic Commerce, Internet en EDI*, Informatie, January 1998 (in Dutch).
17. BOERTIEN, N., M. BIJLSMA, W. JANSSEN, P. VAN DER STAPPEN, AND M. STEFANOVA, *State of the art in e-business services and components*, TI/RS/2000/024, Telematica Instituut, Enschede, 2000.
18. BOLLIER, D., *The Future of Electronic Commerce*, The Aspen Institute, 1996.
19. BONNS, R.W.H., *Designing trustworthy trade procedures for open Electronic Commerce*, PhD dissertation, Erasmus University Rotterdam, 1997.
20. BONNS, R.W.H., R.M. LEE AND R.W. WAGENAAR, *Obstacles for the Development of Open Electronic Commerce*, International Journal of Electronic Commerce, Special Issue, Vol. 2, No. 3, pages 61-83, 1998.
21. BONNS, R.W.H., F. DIGNUM, R. LEE AND Y-H. TAN, *A Formal Analysis of Auditing Principles for Electronic Trade Procedures*, in International Journal of Electronic Commerce, Vol. 5, No. 1, pages 57-82, 2000.
22. BOOCH, G., *Object-oriented analysis and design with applications*, 2nd edition, The Benjamin/Cummings Publishing Company Inc., 1994.
23. BRINKSMA, E., *Tussen droom en daad: Formele methoden en gereedschappen bij specificatie en implementatie van open systemen*, Informatie, Vol. 35, No. 9, pages 537-546, 1993 (in Dutch).
24. CASTELFRANCHI, C., F. DIGNUM, C. JONKER AND J. TREUR, *Deliberate Normative Agents: Principles and Architectures*, in N. Jennings and Y. Lesperance (editors), Intelligent Agents VI (LNAI-1757), pages 364-378, Springer-Verlag, 2000.
25. CHAVEZ, A. AND P. MAES, *Kasbah: An Agent Marketplace for Buying and Selling Goods*, Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, London, 1996.
26. CHEN, P.P.S., *The entity-relationship model - toward a unified view on data*, ACM Transactions on Database Systems, Vol. 1, No. 1, pages 9-36, March 1976.
27. CHOI, S.Y., D.O. STAHL, A.B. WHINSTON, *The Economics of Electronic Commerce*, Macmillan Technical Publishing, 1997

28. CREEMERS, M.R., *Transacties, bedrijfsprocessen en informatietechnologie*, Informatie en Informatiebeleid, Spring 1992 (in Dutch).
29. CREEMERS, M.R., *Transaction Engineering - Process design and information technology beyond interchangeability*, PhD dissertation, 1993.
30. CROSSFLOW, *CrossFlow Project Overview*, www.crossflow.org, 2001.
31. DALMEIJER, M., D.K. HAMMER AND A.T.M. AERTS, *Mobile Software Agents*, Computers in Industry, Vol. 41, No. 3, pages 251-260, 2000.
32. DAVENPORT, T.H., *Process innovation: reengineering work through information technology*, Harvard Business School Press, 1993.
33. DAVIDOW, W.H. AND M.S. MALONE, *The virtual corporation*, HarperBusiness, 1992.
34. DIETZ, J.L.G., *Introduction to DEMO*, Samson Bedrijfsinformatie, 1996.
35. DIGNUM, F. AND H. WEIGAND, *Communication and deontic logic*, in R. Wieringa and R. Feenstra, editors, Information Systems, Correctness and Reusability, pages 242-260, World Scientific, Singapore, 1995.
36. DIGNUM, F., E. VERHAREN AND H. WEIGAND, *A Language/Action perspective on Cooperative Information Agents*, in N. van der Rijst, E. Verharen and J. Dietz, editors, International Workshop on Communication Modelling (LAP-96), pages 40-53, Oisterwijk, 1996.
37. DIGNUM, F., *Social interactions of autonomous agents; private and global views on communication*, in J.J. Meyer and P-Y. Schobbens, editors, Formal models of agents (LNCS-1760), pages 103-122, Springer-Verlag, 1999.
38. DIGNUM, F., *Software agents en Electronic Commerce*, Informatie, pages 18-22, April 1998 (in Dutch).
39. DIGNUM, F., *FLBC: From messages to protocols*, In Y-H. Tan and W. Thoen (editors), Proceedings of the International workshop on Formal Methods in Electronic Commerce, pages 15-29, Rotterdam, June 1999.
40. DIGNUM, F., *Agent Communication and Cooperative Information Agents*, in M. Klusch and L. Kerschberg (editors), Cooperative Information Agents IV – The Future of Information Agents in Cyberspace (LNCS-1860), pages 191-207, Springer-Verlag, 2000.
41. EBXML, *Technical Architecture Specification V1.0.4*, ebXML Technical Architecture Project Team, www.ebxml.org, 16 February 2001.
42. EBXML, *E-commerce patterns V1.0*, www.ebxml.org, 11 May 2001.
43. EIU, *Competing in the digital age (How the Internet will transform global business)*, Research report by The Economist Intelligence Unit and Booz • Allen & Hamilton, 1999.
44. EUROPEAN BOARD FOR EDI STANDARDS / EUROPEAN EXPORT GROUP 4 – FINANCE (EBES/EEG4), *A guide to financial EDI – Strategy and Implementation*, 1996.
45. EDIFORUM, *De Nationale EDI-Gids*, 1994 (in Dutch).

46. EDIT, *EDI Development and Implementation Tool*, Version 2.7, User Manual, Bakkenist Management Consultants, Diemen, 1996.
47. EIJK, P.H. J. VAN, C.A. VISSERS AND M. DIAZ, *The Formal Description Technique LOTOS*, North-Holland, 1989.
48. ELLIS, C.A., *Information Control Nets: A mathematical model of office information flow*, in Proceedings of the Conference on Simulation, Measurement and Modelling of Computer Systems, Boulder, Colorado, pages 225-240, ACM Press, 1979.
49. ELLIS, C.A. AND G.J. NUTT, *Modelling and enactment of Workflow Systems*, in M. Ajmone Marsan, editor, Application and Theory of Petri Nets 1993, volume 691 of Lecture Notes in Computer Science, pages 1-16, Springer-Verlag, Berlin, 1993.
50. EXSPECT, *User manual*, Bakkenist Management Consultants, Diemen, 1997.
51. FERREIRA, J., B. DEROCHER AND L. PROKOPETS, *Leveraging the e-Business Marketplace, Business-to-business e-Procurement Trends, Opportunities, and Challenges*, Deloitte Consulting (www.dc.com), 1999.
52. FININ, T., J. WEBER ET AL, *Specification of the KQML Agent-Communication language*, The DARPA Knowledge Sharing Initiative, 1993.
53. FININ, T., R. FRITZSON, D. MCKAY AND R. MCENTIRE, *KQML as an Agent Communication Language*, The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM'94), pages 456-463, ACM Press, 1994.
54. FLORIJN, G. AND M. VAN ELSWIJK, *OO ontwerptrend: Design Patterns*, Informatie, Vol. 38, No. 2, February 1996 (in Dutch).
55. GAMMA, E., R. HELM, R. JOHNSON AND J. VLISSIDES, *Design patterns: elements of reusable object-oriented software*, Addison Wesley, 1994.
56. GOLDKUHL, G., *Generic Business Frameworks and Action Modelling*, Proc. of 1st International workshop on Communication Modelling, Springer-Verlag, 1996.
57. GREFEN, P., K. ABERER, Y. HOFFNER AND H. LUDWIG, *CrossFlow: Cross-Organizational Workflow Management in Dynamic Virtual Enterprises*, International Journal of Computer Systems Science & Engineering, Vol. 15, No. 5, pages 277-290, 2000.
58. GRIFFEL, F., M. BOGER, H. WEINREICH, W. LAMERSDORF AND M. MERZ, *Electronic Contracting with COSMOS – How to Establish, Negotiate and Execute Electronic Contracts on the Internet*, in 2nd International Enterprise Distributed Object Computing Workshop (EDOC'98), 1998.
59. GUTTMAN, R.H AND P. MAES, *Agent-mediated Integrative Negotiation for Retail Electronic Commerce*, Proceedings of the Workshop on Agent Mediated Electronic Trading (AMET-98), Mineapolis, 1998.
60. HAMMER, M. AND J. CHAMPY, *Reengineering the corporation*, HarperBusiness, 1993.
61. HEE, K.M. VAN, *Information System Engineering: a Formal Approach*, Cambridge University Press, 1991.

62. HOFMAN, W.J., *EDI handbook, electronic data interchange between organisations*, Tutein Nolthenius, 1989 (in Dutch).
63. HOFMAN, W.J., *A conceptual model of a Business Transaction Management System*, PhD dissertation, Tutein Nolthenius, 1994.
64. HOFFNER, Y., *Supporting Contract Match-Making*, Proceedings of the Ninth International Workshop on Research Issues in Data Engineering, IEEE Computer Society, 1999.
65. HOOGEWEEGEN, M.R., R.W. WAGENAAR, W.E.J.M. BENS AND J.A.E.E VAN NUNEN, *Het bepalen van kosten en baten van EDI-investeringen*, Informatie, Vol. 37, No. 1, 1995 (in Dutch).
66. HOOGEWEEGEN, M.R. AND F. DE JONG, *Naar een zakelijke inschatting van het nut van EDI*, Informatie Management, Vol. 12, No. 11, pages 39-43, 1996 (in Dutch).
67. IAB, *Architectural Principles of the Internet*, RFC 1958, The Internet Society, www.ietf.org June 1996.
68. IAB, *Internet Open Trading Protocol - IOTP*, Version 1.0, RFC 2801, The Internet Society, www.ietf.org, April 2000.
69. ISO, *EDIFACT - Syntax rules common to both batch and interactive EDI*, IS 9735-1, 1996.
70. ISO, *EDIFACT - Syntax rules specific to batch EDI*, IS 9735-2, 1996.
71. ISO, *EDIFACT - Syntax rules and controls specific to Interactive EDI*, IS 9735-3, 1996.
72. ISO, *The Open-Edi Reference Model*, IS 14662, ISO/IEC JTC1/SC30, 1997.
73. ITU-TS, *ITU-TS Recommendation Z.120: Message Sequence Chart 1996 (MSC96)*, Technical report, ITU-TS, Geneva, 1996.
74. JABLONSKI, S. AND C. BUSSLER, *Workflow Management, Modelling Concepts, Architecture and Implementation*, International Thomson Computer Press, 1996.
75. JENNINGS, N.R., P. FARATIN, M.J. JOHNSON, P. O'BRIEN AND M.E. WIEGAND, *Using Intelligent Agents to Manage Business Processes*, Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology (PAAM 96), pages 345-360, London, 1996.
76. JENNINGS, N.R., K. SYCARA, M. WOOLDRIDGE, *A Roadmap of Agent Research and Development*, Journal of Autonomous Agents and Multi-Agent Systems, Vol. 1, pages 7-38, 1998.
77. JENSEN, K., *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, EATC Monographs on Theoretical Computer Science, Springer-Verlag, 1992.
78. JOOSTEN, S.M.M., *Werkstromen: een overzicht*, Informatie, Vol. 36, No. 9, pages 518-528, 1995 (in Dutch).
79. KALATOKA, R. AND A.B. WHINSTON, *Frontiers of the Electronic Commerce*, Addison-Wesley Publishing Company, 1996.

80. KIMBROUGH, S.O. AND R.M. LEE, *On Illocutionary Logic as a Telecommunications Language*, Proceedings of the International Conference on Information Systems, San Diego, December 1986.
81. KIMBROUGH, S.O. AND S.A. MOORE, *On Automated Message Processing in Electronic Commerce and Work Support Systems: Speech Act Theory and Expressive Felicity*, ACM Transactions on Information Systems, Vol. 15 No. 4, pages 321-367, October 1997.
82. KLINT, P. AND G. WIJERS, *De renovatie van systemen*, Informatie, June 1996 (in Dutch).
83. KLINT, P. AND C. VERHOEF, *Evolutionary Software Engineering A Component-based Approach*, Report P9801, Programming Research Group, University of Amsterdam, 1998.
84. KINDLER, E., A. MARTENS AND W. REISIG, *Inter-operability of Workflow Applications: Local Criteria for Global Soundness*, in Business Process Management: Models, Techniques and Empirical Studies, volume 1806 of Lecture Notes in Computer Science, pages 235-253, Springer-Verlag, Berlin, 2000.
85. LAZCANO, A., G. ALONSO, H. SCHULDT, C. SCHULER, *The WISE approach to Electronic Commerce*, International Journal of Computer Systems Science & Engineering, special issue on Flexible Workflow Technology Driving the Networked Economy, Vol. 15, No. 5, pages 345-357, 2000.
86. LEE, R.M., *A Logic Model for Electronic Contracting*, Decision Support Systems, Vol. 4, No. 1, pages 27-44, 1988.
87. LEE, R.M., R.W.H BONS, C.D. WRIGLEY AND R.W WAGENAAR, *Automated Design of Electronic Trade Procedures Using Documentary Petri Nets*, Proceedings Fourth International Conference on Dynamic Modeling and Information Systems, Noordwijkerhout, September 1994.
88. LEE, R.M. AND R.W.H. BONS, *Soft-Coded Trade Procedures for Open-edi*, International Journal of Electronic Commerce, Vol. 1, No. 1, pages 27-49, 1996.
89. LEE, R.M., *CANDID - A Logical Calculus for Describing Financial Contracts*, PhD Dissertation, Department of Decision Sciences, The Wharton School, University of Pennsylvania, 1980.
90. LEE, R.M., *Interprocs: A java-based Prototyping Environment for Distributed Electronic Trade Procedures*, Proceedings of the Hawaii International Conference on System Sciences, pages 202-209, January, 1998.
91. LEE, R.M., *Towards Open Electronic Contracting*, Journal of Electronic Markets, Special issue on Electronic Contracting, Vol. 2, No. 1, 1998.
92. LEE, R.M., *Distributed Electronic Trade Scenarios: Representation, Design, Prototyping*, International Journal of Electronic Commerce, Vol. 3, No. 2, pages 105-120, 1999.

93. MERZ, M., F. GRIFFEL, T. TU, S. MÜLLER-WILKEN, H. WEINREICH, M. BOGER AND W. LAMERSDORF, *Supporting Electronic Commerce Transactions with Contracting Services*, International Journal of Cooperative Information Systems, Vol. 7, No. 4, World Scientific, 1998.
94. MICROSOFT CORPORATION, *The Component Object Model Specification*, www.microsoft.com, 24 October 1995.
95. MICROSOFT CORPORATION, *DCOM Technical Overview*, www.microsoft.com, November 1996.
96. MICROSOFT CORPORATION, *BizTalk Framework 2.0: Document and Message Specification*, www.microsoft.com, December 2000.
97. NORMANN, R. AND R. RAMIREZ, *Designing interactive strategy: from value chain to value constellation*, John Wiley & Sons Ltd, 1994.
98. OBI, *Open Buying on the Internet Technical Specifications*, Version 2.0, The OBI Consortium, www.openbuy.org, 1999.
99. OFX, *Open Financial Exchange*, specification 2.0, Open Financial Exchange, www.ofx.net, July 2000.
100. OMG, *The Common Object Request Broker: Architecture and Specification*, Revision 2.5, Object Management Group, www.omg.org, September 2001.
101. OMG, *Unified Modelling Language Specification*, version 1.4, Object Management Group, www.omg.org, September 2001.
102. PATTISON, T., *Programming Distributed Applications with COM and Microsoft Visual Basic 6.0*, Microsoft Press, 1998.
103. PETRI, C.A., *Kommunikation mit Automaten*, PhD dissertation, Institut für Instrumentelle Mathematik, Bonn, 1962.
104. PORTER, M.E., *Competitive Advantage: Creating and Sustaining Superior Performance*, Free Press, 1998.
105. RODDY, D.J., *Online B2B Exchanges, the new economics of markets*, Deloitte Consulting, www.dc.com, 1999.
106. RODDY, D.J., *The New Economics of Transactions, evolution of unique e-business Internet market spaces*, Deloitte Consulting, www.dc.com, 1999.
107. RUMBAUGH, J., M. BLAHA, W. PREMERLANI, F. EDDY AND W. LORENSEN, *Object-Oriented Modelling and Design*, Prentice-Hall International, Inc., 1991.
108. SEARLE, J.R., *Speech Acts: an essay in the philosophy of language*, Cambridge University Press, 1969.
109. SEARLE, J.R. AND D. VANDERVEKEN, *Foundations of illocutionary logic*, Cambridge University Press, 1985.
110. SOKOL, P.K., *EDI: the competitive edge*, Intertext Publications, McGraw-Hill Book Company, New York, 1989.

111. SUN, *Java Beans API specification*, version 1.01, Sun Microsystems, www.java.sun.com/beans, 24 July 1997.
112. SZYPERSKI, C., *Component Software, beyond Object-Oriented Programming*, Addison-Wesley, 1998.
113. TAPSCOTT, D. AND A. CASTON, *Paradigm Shift – The New Promise of Information Technology*, McGraw-Hill, 1993.
114. UDDI, *UDDI Executive White Paper*, www.uddi.org, 6 September 2000.
115. UDDI, *UDDI Technical White Paper*, www.uddi.org, 6 September 2000.
116. UNITED NATIONS, *The Business and Information Modelling Framework for UN/EDIFACT*, working draft UN/ECE TRADE/WP.4/R.1212, 1996.
117. UNITED NATIONS, *UN/EDIFACT Message Type Directory Batch*, Version D.01B, www.unece.org/trade/untdid, 2001.
118. VANDENBULCKE, J., *Met componentensoftware naar de wendbare onderneming*, Informatie, February 1998 (in Dutch).
119. VERBEEK, H.M.W. AND W.M.P. VAN DER AALST, *Woflan 2.0: A Petri-net-based Workflow Diagnosis Tool*, in M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000*, volume 1825 of *Lecture Notes in Computer Science*, pages 475-484, Springer-Verlag, Berlin, 2000.
120. VERHAREN, E.M. AND F. DIGNUM, *Cooperative information agents and communication*, In M. Klusch and P. Kandzia, editors, *Proceedings of the first international workshop on cooperative information agents*, LNAI-1202, pages 195-209, Kiel, Germany, Springer-Verlag, 1997.
121. VERHAREN, E.M., *A Language-Action Perspective on the Design of Cooperative Information Agents*, PhD dissertation, 1997.
122. VLIST, P. VAN DER ET AL, *EDI in trade*, Samson Bedrijfsinformatie, Alphen aan den Rijn, 1992 (in Dutch).
123. VLIST, P. VAN DER ET AL, *EDI in transport*, Samson Bedrijfsinformatie, Alphen aan den Rijn, 1994 (in Dutch).
124. VLIST, P. VAN DER AND A. VAN DIJK, *EDI and Internet*, *Informatie en Informatiebeleid*, Vol. 15, No. 4, 1997 (in Dutch).
125. W3C, *Extensible Markup Language (XML) 1.0 (Second Edition)*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 6 October 2000.
126. W3C, *Document Object Model (DOM) Level 1 Specification*, Version 1.0, W3C Recommendation, World Wide Web Consortium, www.w3.org, 1 October 1998
127. W3C, *Namespaces in XML*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 14 January 1999.
128. W3C, *XSL Transformations (XSLT) Version 1.0*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 16 November 1999.

129. W3C, *XML Path Language (XPath) Version 1.0*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 16 November 1999.
130. W3C, *XML Schema Part 0: Primer*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 2 May 2001.
131. W3C, *XML Schema Part 1: Structures*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 2 May 2001.
132. W3C, *XML Schema Part 2: Datatypes*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 2 May 2001.
133. W3C, *HTML 4.01 Specification*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 24 December 1999.
134. W3C, *Extensible Stylesheet Language (XSL) Version 1.0*, W3C Proposed Recommendation, World Wide Web Consortium, www.w3.org, 28 August 2001.
135. W3C, *Canonical XML Version 1.0*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 15 March 2001.
136. W3C, *XHTML 1.0: The Extensible HyperText Markup Language*, W3C Recommendation, World Wide Web Consortium, www.w3.org, 26 January 2000.
137. W3C, *Simple Object Access Protocol (SOAP) 1.1*, W3C Note, World Wide Web Consortium, www.w3.org, 8 May 2000.
138. WEIGAND, H. AND F. DIGNUM, *Formalization and rationalization of communication*, in F. Dignum and J. Dietz, editors, *Second International Workshop on Communication Modelling (LAP-97)*, pages 71-86, Veldhoven, 1997.
139. WEIGAND, H., W-J. VAN DEN HEUVEL AND F. DIGNUM, *Modelling Electronic Commerce Transactions; A layered approach*, In G. Goldkuhl, editor, *Third International Workshop on Communication Modelling (LAP-98)*, pages 47-58, Stockholm, 1998.
140. WFMC, *The Workflow Reference Model, (TC00-1003)*, Workflow Management Coalition, www.wfmc.org, 19 January 1995.
141. WFMC, *Terminology & Glossary, (TC00-1011)*, Workflow Management Coalition, www.wfmc.org, February 1999.
142. WFMC, *Workflow Standard - Interoperability Abstract Specification, (TC00-1012), Version 2.0b*, Workflow Management Coalition, www.wfmc.org, 30 November 1999.
143. WFMC, *Workflow and Internet: Catalysts for Radical Change*, Workflow Management Coalition white paper, www.wfmc.org, June 1998.
144. WFMC, *Workflow Interoperability – Enabling E-Commerce*, Workflow Management Coalition white paper, www.wfmc.org, April 1999.
145. WFMC, *Workflow Standard - Interoperability Wf-XML binding, (TC-1023)*, Workflow Management Coalition Specification, www.wfmc.org, 1 May 2000.
146. WOMACK, J.P., *The Machine That Changed the World: The Story of Lean Production*, HarperCollins, 1991.

147. WOOLDRIDGE, M. J. AND N.R. JENNINGS, *Intelligent Agents: Theory and Practice*, The Knowledge engineering Review, Vol. 10, No. 2, pages 115-152, 1995.
148. WURMAN, P. R., M. P. WELLMAN, W. E. WALSH, *The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents*, Proceedings of the Second International Conference on Autonomous Agents (Agents-98), Minneapolis, 1998.

Abbreviations and acronyms

API	Application Program Interface
BTMS	Business Transaction Management System
COM	Component Object Model
CORBA	Common Object Request Broker Architecture
COTS	Commercial Off The Shelf
CXML	Commerce XML
DCOM	Distributed Component Object Model
DOM	Document Object Model
EAN	European Article Numbering
EBES	European Board for EDI Standards
EBXML	Electronic Business XML
ECR	Efficient Consumer Response
EXSPECT	Executable Specification Tool
EDI	Electronic Data Interchange
EDIT	Edi Documentation and Implementation Tool
EDIFACT	EDI For Administration Commerce and Transport
GUID	Globally Unique Identifier
HTML	HyperText Markup Language
HTTP	HyperText Transfer Protocol
ICT	Information and Communication Technology
IP	Internet Protocol
IAB	Internet Architecture Board
IDL	Interface Description Language
IESG	Internet Engineering Steering Group
IETF	Internet Engineering Task Force
IOTP	Internet Open Trading Protocol
ISO	International Standardisation Organisation
LAP	Language Action Perspective
MIME	Multipurpose Internet Mail Extension
MSC	Message Sequence Chart
OBI	Open Buying on the Internet
OFX	Open Financial Exchange
OMG	Object Management Group
OO	Object-Oriented
ORB	Object Request Broker
OTP	Open Trading Protocol

POP	Post Office Protocol
RFC	Request For Comment
SCM	Supply Chain Management
SGML	Standard Generalised Markup Language
SMTP	Simple Mail Transfer Protocol
STD	State Transition Diagram
TCP/IP	Transmission Control Protocol / Internet Protocol
TDID	Trade Data Interchange Directory
TSD	Time Sequence Diagram
UML	Unified Modelling Language
URL	Uniform Resource Locator
VAN	Value Added Network
W3C	World Wide Web Consortium
WFM	Work Flow Management
WFMC	Work Flow Management Coalition
WFMS	Work Flow Management System
WWW	World Wide Web
XML	eXtensible Markup Language
XSL	Extensible Stylesheet Language

Samenvatting (Dutch)

Bedrijfsprocessen beperken zich zelden tot de grenzen van één organisatie. In plaats daarvan is in toenemende mate sprake van inter-organisatiele bedrijfsprocessen. Voor deze bedrijfs-overstijgende processen zijn verschillende vormen mogelijk, bijvoorbeeld gebaseerd op het delen van kennis en informatie, het delen van verwerkingscapaciteit, het uitwisselen van casussen en het uitbesteden van werk door middel van het contracteren van diensten van derden. Dit onderzoek richt zich op inter-organisatiele processen die gebaseerd zijn op uitbesteding van werk via het elektronisch contracteren van diensten van derden. Alhoewel er meestal een verschil wordt gemaakt tussen een ‘product’ en een ‘dienst’ zullen we in dit onderzoek het begrip ‘dienst’ gebruiken als synoniem voor beide.

Afbakening

Het Engelstalige begrip ‘electronic contracting’ wordt gebruikt voor een variëteit aan verschijnselen. Dit onderzoek richt zich op een specifiek deel hiervan, dat wordt afgebakend door de volgende karakteristieken. Ten eerste richten we ons op een omgeving waarin tussen de deelnemende organisaties een ‘losse’ koppeling bestaat, dat wil zeggen: alle communicatie vindt plaats door uitwisseling van gestructureerde berichten, waarvan de inhoud (semantiek en syntax) en het protocol (volgorde van de berichttypen) is overeengekomen. Deze ‘losse’ vorm van koppeling veronderstelt dat geen gebruik wordt gemaakt van een centrale ‘broker’ component, en dat partijen geen kennis (hoeven) hebben van elkaars bedrijfsprocessen.

Een contracteringsproces vergt altijd de deelname van minstens twee partijen: één in de rol van koper en één in de rol van verkoper. Dit onderzoek beperkt zich tot het deel van het contracteringsproces dat door de *koper* wordt uitgevoerd. Het begrip ‘service contracting’ wordt daarom gebruikt voor de acties uitgevoerd door de inkoper van de dienst:

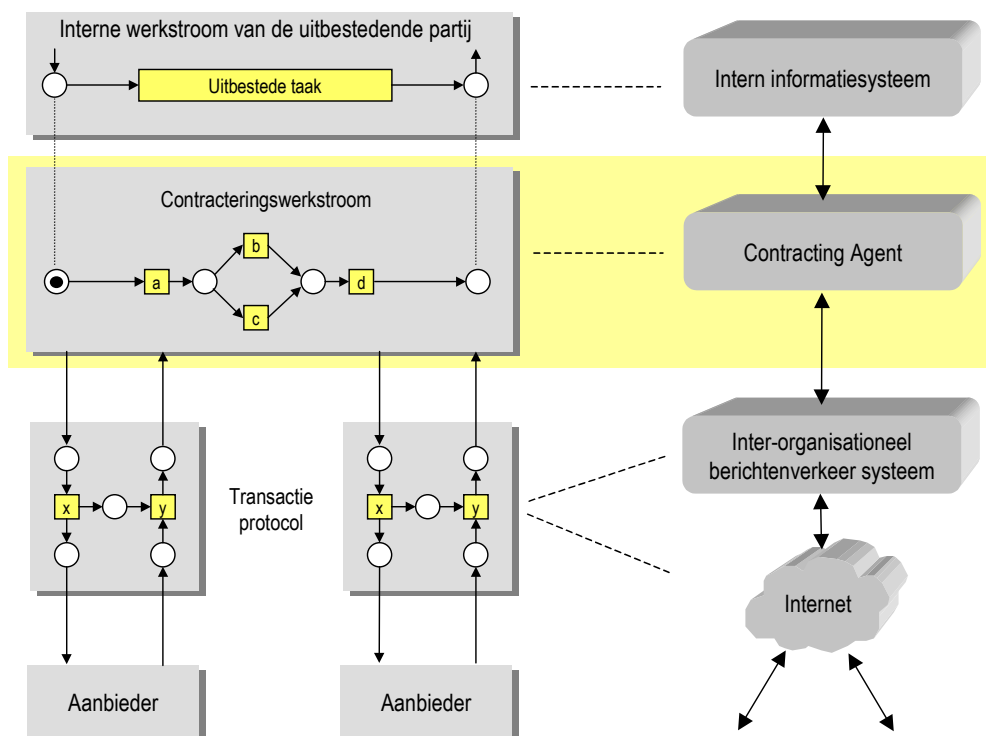
- *specificeren* van de details van de benodigde dienst;
- *onderhandelen* van een contract met een aanbieder;
- *bewaken* van de uitvoering van het contract;
- *aanvaarden* van het resultaat.

We geven nu een nadere toespitsing van het type contracteringsproces waarop dit onderzoek zich richt. Allereerst richten we ons op de complexere processen, waarbij voor iedere casus N verschillende diensten moeten worden gecontracteerd en waarbij voor iedere dienst M verschillende aanbieders beschikbaar zijn. Voor de N benodigde diensten veronderstellen we verder *randvoorwaarden* op de volgorde waarin ze gecontracteerd worden. Ook veronderstellen we *afhankelijkheden* tussen de details van de te contracteren diensten. Bijvoorbeeld, wanneer een vlucht naar New York en een huurauto op het vliegveld van bestemming moeten worden gecontracteerd kan het contracteren van de laatste pas beginnen wanneer het vliegveld van bestemming bekend is. Daarom is het nodig om eerst de vlucht te boeken en daarna – als het vliegveld bekend is – de huurauto. Een andere veronderstelling die we maken is dat partijen hoogstens gedeeltelijke kennis hebben van elkaars beschikbare capaciteit. Dit, gecombineerd

met de autonomie van aanbieders, maakt het onmogelijk om de uitvoering van uitbesteed werk *toe te wijzen* aan een externe partij. In plaats daarvan is het nodig om een *onderhandeling* aan te gaan met één of meer aanbieders. Tenslotte richt dit onderzoek zich op *geheel geautomatiseerde* contracteringsprocessen, wat een in hoge mate gestructureerde specificatie van het contracteringsproces vraagt die geïnterpreteerd kan worden door computer-toepassingen.

Aanpak

De gevolgde aanpak is geïllustreerd in Figuur 275. Het linkerdeel van de figuur illustreert het conceptuele domein en het rechterdeel illustreert het technische (software infrastructuur) domein. In het conceptuele domein onderkennen we de interne werkstroom van een organisatie, die één of meer uitbesteede taken kent. De uitvoering van die uitbesteede taken vergt het contracteren van één of meer diensten van externe aanbieders. Communicatie met deze aanbieders vindt plaats via transacties die bestaan uit gestructureerde berichten waarvan de statische en dynamische aspecten zijn gedefinieerd in het transactieprotocol. De activiteiten gericht op de onderhandeling en de totstandkoming van het contract, het bewaken van de uitvoering van het contract en het aanvaarden van het resultaat kunnen gemodelleerd worden in een contracteringswerkstroom. De specificatie van contracteringseisen en de constructie van deze contracteringswerkstroom is het onderwerp van dit onderzoek. In aanvulling daarop richt het onderzoek zich ook op het ontwerp en de constructie van een softwarecomponent, de Contracting Agent, waaraan het interne informatiesysteem de uitvoering van contracteringsprocessen, beschreven door contracteringswerkstromen, delegeert. Voor het uitwisselen van berichten maakt de Contracting Agent gebruik van standaard software voor inter-organisatieel berichtenverkeer, dat onder andere de functies conversie, authenticatie en communicatie biedt.



Figuur 275 Illustratie van de gevolgde aanpak

Doelstelling en bijdrage

De doelstelling van dit onderzoek is om een bijdrage te leveren aan de efficiency van organisaties door optimale ondersteuning te bieden voor de afgebakende klasse van contracteringsprocesses met behulp van informatie-technologie. De bijdrage van het onderzoek bestaat uit:

- het expliciet maken van het afgebakende gebied door middel van het *conceptuele raamwerk*;
- een *specificatietaal* waarin contracteringseisen voor uitbestede taken kunnen worden uitgedrukt;
- een verzameling *standaard transities* waaruit contracteringswerkstromen kunnen worden samengesteld;
- een mechanisme waardoor ‘sound’ contracteringswerkstromen kunnen worden *gegenereerd* uit contracteringseisen;
- een *architectuur* (logisch en technisch) van een softwarecomponent die de afgebakende klasse van contracteringsprocessen ondersteunt;
- een ‘*proof of concept*’ van het conceptuele raamwerk, de specificatietaal en de architectuur in de vorm van een operationele software component.

Resultaten

De resultaten van het onderzoek bestaan uit (i) een conceptueel raamwerk voor de klasse van contracteringsprocessen, (ii) een logische architectuur van de Contracting Agent, (iii) een technische architectuur van de Contracting Agent en (iv) een implementatie van de Contracting Agent.

• Conceptueel raamwerk (Hoofdstuk 2)

Het conceptueel raamwerk begint met een definitie van onderliggende concepten zoals werkstromen, diensten en transacties. Daarna introduceren we het begrip ‘contracteringsproces’ en beschrijven raamwerken voor contracteringsprocessen uit de literatuur: Action Workflow, DEMO en BAT. Een synthese van deze raamwerken leidt tot het onderscheid in vier achtereenvolgende fasen: specificatie, onderhandeling, uitvoering en aanvaarding. Daarna definiëren we de concepten die nodig zijn voor het specificeren van interfaceafspraken tussen partijen in een contracteringsproces: ‘dienst type’, ‘aanbieder’, ‘transactie protocol’ en ‘bericht type’. Voor iedere fase die berichtuitwisseling met de aanbieder omvat geven we een aantal ‘patterns’ voor het transactieprotocol in die fase. Hierna richten we ons op de specificatie van contracteringseisen voor uitbestede taken. Deze eisen bepalen *welke* diensten moeten worden gecontracteerd en *hoe* deze moeten worden gecontracteerd (bijvoorbeeld de te volgen onderhandelingsstrategie). Met de interfaceafspraken aan de ene kant en de contracteringseisen aan de andere kant hebben we de basis voor het definiëren van de contracteringswerkstroom. Als voorbereiding hiervoor definiëren we eerst de toestandsdata van het contracteringsproces, de standaard bewerkingen op deze toestandsdata en de configuratieparameters die deze standaard bewerkingen gebruiken. Daarna gaan we uit van een contracteringswerkstroom die gemodelleerd is als hoog niveau gekleurd Petri net, van waaruit de standaard bewerkingen in de juiste volgorde worden aangeroepen. Voor deze werkstromen stellen we een verzameling van standaard transities voor waaruit zij samengesteld kunnen worden. Deze standaard transities worden weer gebruikt om standaard transities te maken met een lagere granulariteit voor bijvoorbeeld een complete onderhandelingsfase. Daarna bespreken we de regels volgens welke de contracteringswerkstroom kan worden samengesteld uit de interfaceafspraken aan de ene kant en de contracteringseisen aan de andere kant. Tenslotte illustreren we het conceptuele raamwerk door een casus

waarin diensten voor een zakenreis moeten worden gecontracteerd: twee vluchten, een hotelovernachting en een huurauto.

- **Logische architectuur (Hoofdstuk 3)**

De logische architectuur beschrijft de structuur en het gedrag van de componenten waaruit de Contracting Agent bestaat. De structuur van persistente data en van data die wordt uitgewisseld op de koppelvlakken van een component beschrijven we via functionele datamodellen. Het gedrag van iedere component op zijn koppelvlakken, en de samenwerking met andere componenten via de koppelvlakken beschrijven we via hoog niveau gekleurde Petri netten. Op het hoogste niveau onderscheiden we drie componenten: Server, Configurator en Monitor. De Server component is ontworpen volgens de principes van ‘workflow management systemen’: scheiding van besturing en uitvoering. Persistente data wordt opgeslagen in een relationele database, standaard bewerkingen op de persistente data wordt geïmplementeerd in relatief kleine applicaties, die op hun beurt worden aangeroepen door de ‘workflow engine’ die is geconfigureerd met de definitie van de contracteringswerkstroom. De Configuratie component biedt een opslag van interface-afspraken met aanbieders en ondersteunt de gebruiker in het definiëren van contracteringseisen voor uitbestede taken. Met deze gegevens als invoer genereert de Configuratie component de gehele contracteringswerkstroom, samen met alle andere configuratieparameters die door de Server component worden gebruikt. Tenslotte biedt de Monitor component functionaliteit om de Server component te bevragen op zijn toestandsdata en deze te presenteren aan de gebruiker via een grafisch gebruikersinterface.

- **Technische architectuur (Hoofdstuk 4)**

Dit hoofdstuk geeft de vertaling van de logische architectuur in een technische architectuur met de bedoeling een operationele versie van de Contracting Agent te maken. Omdat de bedoeling van deze component is te dienen als ‘proof of concept’ ligt de nadruk meer op het efficiënt realiseren van de functionaliteit dan op aspecten als schaalbaarheid en inzetbaarheid op verschillende platforms. We kiezen daarom voor Windows als besturingsstroom en COM als componentenraamwerk. Verder modelleren we hiërarchische datastructuren als XML document en gebruiken we afgeleide standaarden voor bewerkingen op XML documenten: validatie (XML-Schema), transformatie (XSLT) en presentatie (XSL). Deze standaarden worden ondersteund door de standaard MSXML4 component. Een tweede standaard component die we gebruiken is de ExSpect engine als werkstroombesturingssysteem. De mogelijkheid die ExSpect biedt om hoog niveau gekleurde Petri netten te executeren laat ons de conceptuele contracteringswerkstroom bijna zonder wijziging omzetten naar de werkstroom die wordt geëxecuteerd in ExSpect. Tenslotte gebruiken we ook het hulpmiddel WOFLAN voor het analyseren van eigenschappen van werkstromen.

- **Evaluatie (Hoofdstuk 5)**

Het laatste hoofdstuk beschouwt de behaalde resultaten en de bijdrage van het onderzoek aan al bestaand onderzoek. We komen tot de conclusie dat het toepassen van domeinkennis van contracteringsprocessen in de Configuratie component, en het genereren van contracteringswerkstromen uit eenvoudiger contracteringseisen een grote efficiëntieverbetering brengt in het implementeren van contracteringsprocessen.

