

Université de Montréal

Workflow Technology for Complex Socio-Technical Systems

par

Sarita Bassil

Département d'Informatique et de Recherche Opérationnelle

Faculté des Arts et des Sciences

Thèse présentée à la Faculté des Études Supérieures en vue de l'obtention
du grade de Philosophiæ Doctor (Ph.D.) en Informatique

Décembre, 2004

© Sarita Bassil, 2004



QA

76

U54

2005

v. 031

Direction des bibliothèques

AVIS

L'auteur a autorisé l'Université de Montréal à reproduire et diffuser, en totalité ou en partie, par quelque moyen que ce soit et sur quelque support que ce soit, et exclusivement à des fins non lucratives d'enseignement et de recherche, des copies de ce mémoire ou de cette thèse.

L'auteur et les coauteurs le cas échéant conservent la propriété du droit d'auteur et des droits moraux qui protègent ce document. Ni la thèse ou le mémoire, ni des extraits substantiels de ce document, ne doivent être imprimés ou autrement reproduits sans l'autorisation de l'auteur.


Afin de se conformer à la Loi canadienne sur la protection des renseignements personnels, quelques formulaires secondaires, coordonnées ou signatures intégrées au texte ont pu être enlevés de ce document. Bien que cela ait pu affecter la pagination, il n'y a aucun contenu manquant.

NOTICE

The author of this thesis or dissertation has granted a nonexclusive license allowing Université de Montréal to reproduce and publish the document, in part or in whole, and in any format, solely for noncommercial educational and research purposes.

The author and co-authors if applicable retain copyright ownership and moral rights in this document. Neither the whole thesis or dissertation, nor substantial extracts from it, may be printed or otherwise reproduced without the author's permission.

In compliance with the Canadian Privacy Act some supporting forms, contact information or signatures may have been removed from the document. While this may affect the document page count, it does not represent any loss of content from the document.



Université de Montréal
Faculté des Études Supérieures

Cette thèse intitulée:

Workflow Technology for Complex Socio-Technical Systems

présentée par:

Sarita Bassil

a été évaluée par un jury composé des personnes suivantes:

Président-rapporteur: Houari Sahraoui


Directeur de recherche: Rudolf K. Keller

Codirecteur: Peter Kropf

Membre du jury: Yann-Gaël Guéhéneuc

Examineur externe: Kostas Kontogiannis

Représentant du doyen de la FES: Jacques Nantel



Sommaire

La technologie des workflows s'est avérée importante pour des secteurs tels que l'approvisionnement, la logistique et la production. Elle est définie comme un outil informatique dédié à la gestion des procédures d'entreprise. Toutefois, cette technologie ne supporte pas encore de façon adéquate les exigences inhérentes aux systèmes socio-techniques complexes. Les négociations électroniques et le transport sont des exemples de domaines qui font appel à de tels systèmes. L'étude de ces deux domaines nous permet de reconnaître le besoin d'une meilleure technologie des workflows. Par conséquent, un ensemble élucidé et sophistiqué de concepts et de fonctionnalités pour les systèmes de gestion de workflow (WfMSs) est rassemblé, et des solutions appropriées sont proposées pour supporter cet ensemble. Dans ce contexte, un modèle de référence pour les WfMSs est également passé en revue, et une extension de ce modèle est présentée afin d'accommoder ces concepts et ces fonctionnalités.

Dans cette thèse, nous étudions un système de support pour les négociations électroniques (CONSENSUS). Ce système est basé sur un WfMS. Le but de CONSENSUS est d'assister l'utilisateur dans la modélisation et l'exécution d'un certain type de négociation électronique utilisant les workflows. Ce système doit cependant supporter la modification dynamique d'un workflow. Cette fonctionnalité s'avère indispensable pour faire face aux événements imprévus qui peuvent apparaître lors d'une négociation. De nos jours, les WfMSs (par exemple, IBM MQ Series Workflow et WLPI de BEA Systems) supportent seulement de façon limitée ce genre de dynamisme. Par conséquent, les bénéfices de l'approche CONSENSUS se trouvent être réduits.

Une autre application socio-technique non-triviale est étudiée: la planification et le support du transport multi-transfert de conteneurs (MTCT – *Multi-Transfer Container Transportation*). Cette application révèle des besoins en dynamisme pour la composition

des workflows. Nous concevons un système orienté-workflow pour le traitement des requêtes clients. Ce traitement est réalisé par des séquences d'activités interdépendantes qui doivent être créées juste à temps et ensuite adaptées pour répondre aux événements imprévus qui peuvent apparaître. La création et l'adaptation de ces séquences sont basées sur une gestion optimisée des ressources et sur la planification des activités.

Dans le premier système, nous intégrons un prototype de WfMS (ADEPT) qui supporte quelques-unes des modifications dynamiques requises au niveau de l'exécution. D'une part, cette intégration accroît les bénéfices de l'approche CONSENSUS et d'autre part, elle dévoile le besoin de plusieurs autres fonctionnalités qui ne sont pas encore tout à fait supportées par les WfMSs. Dans le deuxième système, le prototype ADEPT est également utilisé. Son API est enrichie avec des fonctionnalités primordiales et des solutions de rechange sont nécessaires pour assurer convenablement la définition d'un modèle de workflow et la gestion (dynamique) des instances.

La réalisation de ces deux applications va au-delà des projets CONSENSUS et MTCT. En effet, la liste des concepts de modélisation de workflow et des fonctionnalités avancées est soigneusement rassemblée. Nous analysons particulièrement les meilleures solutions possibles (l'application appropriée des fonctionnalités offertes par un WfMS, des solutions de rechange, etc.) utilisant trois WfMSs. Nous travaillons également sur une extension formelle d'un méta-modèle de workflow afin de proposer un critère d'exactitude pour l'interruption sans risque d'activités en cours d'exécution. Ceci est une fonctionnalité d'une grande importance. Comme perspectives de recherche, la liste identifiée et les problèmes exprimés peuvent définir un agenda de recherche dans le domaine des workflows. Les solutions déjà étudiées peuvent être applicables dans le contexte d'autres applications poussées. Elles peuvent aussi donner aux développeurs de WfMSs des informations valables pour les futures versions de leurs produits.

Mots clés: technologie des workflows, systèmes de gestion de workflow, concepts de modélisation de workflow, workflows adaptatifs, application de négociation électronique, application de transport, architecture de système.

Workflows are a major enabling technology for areas such as supply chains, logistics and production. They aim to provide computer support to the management of business processes in general. However, this technology offers little adequate support to requirements inherent to complex socio-technical systems. The domains of e-negotiations and transportation are examples that call for such systems. These domains serve us to investigate the need for an enhanced workflow technology. Hence, a clarified and a refined set of concepts and functionality for workflow management systems (WfMSs) is gathered, and appropriate solutions are proposed to deal with this set. In this context, the Workflow Reference Model is also reviewed, and an extension thereof is suggested to accommodate these concepts and functionality.

In this thesis, we study an e-negotiation support system (CONSENSUS) based on a WfMS. CONSENSUS was developed to help the user model and enact a specific kind of e-negotiation using workflows. This system requires, however, support for dynamic modification induced by unexpected events that can occur during negotiation. Current WfMSs (e.g., IBM MQ Series Workflow, BEA's WLPI) support this kind of dynamism in a limited way only, thus reducing the benefits of the CONSENSUS approach to e-negotiations.

Another complex socio-technical application, the multi-transfer container transportation (MTCT) application, exhibits inherently dynamic requirements for workflow modeling. We devise a workflow-oriented system for the processing of customer requests for container transportation. This processing is achieved by specific sequences of interdependent activities that need to be created just-in-time and then adapted to deal with unexpected events that may occur. The creation and the adaptation of activity sequences are based on an optimized resource management and activity scheduling.

In the first system, we integrate a WfMS prototype (ADEPT) that supports some of the required dynamic modifications at the workflow instance level. On one hand, this integration increases the benefits of the CONSENSUS approach. On the other hand, it sheds the light on several workflow requirements not yet fully supported by current WfMSs. In the second system, the ADEPT prototype is also used. Its API is enriched with useful functionality, and workaround solutions are required to properly cope with the definition of a workflow model and with the (dynamic) management of instances.

The realization of these two applications reaches far beyond the CONSENSUS and the MTCT projects. Indeed, the “wish list” of workflow modeling concepts and advanced functionality is carefully gathered. Particularly, we analyze best effort solutions (proper use of WfMS features, workarounds, etc.) applying three state-of-the-art WfMSs. We also work on a formal extension of a workflow meta-model to propose a correctness criterion for safely interrupting running workflow activities. This is a functionality of utmost importance. As research perspectives, the identified “wish list” and the problems expressed while experimenting with current WfMSs may define an agenda for further research in the workflow technology domain. The already investigated solutions may be applicable in the context of other challenging applications, and they may give valuable input to WfMS builders for future versions of their products.

Keywords: workflow technology, workflow management systems, workflow modeling concepts, adaptive workflows, e-negotiation application, transportation application, system architecture.

Table of Contents

Sommaire	iii
Abstract	v
Table of Contents	vii
List of Figures	xii
List of Tables	xv
List of Acronyms	xvi
Dedication	xviii
Acknowledgments	xix
Chapter 1 Introduction	1
1.1 <i>Problem Statement</i>	1
1.2 <i>Research Objectives</i>	4
1.3 <i>Major Contributions</i>	6
1.4 <i>Thesis Structure</i>	8
Chapter 2 Processes, Workflows, and Workflow Management Systems	9
2.1 <i>Workflow Basics and Classifications</i>	9
2.2 <i>Workflow Design</i>	12
2.2.1 <i>Petri Nets and Workflows</i>	15
2.2.2 <i>UML and Workflows</i>	16
2.2.3 <i>WSM-Nets Formalism</i>	17
2.2.4 <i>Workflow Temporal Aspects</i>	19
2.2.5 <i>Organizational Structure</i>	21
2.3 <i>Workflow Enactment</i>	23
2.4 <i>Workflow Management Systems</i>	24
2.4.1 <i>Standardization Effort</i>	24
2.4.1.1 <i>Workflow Enactment Service</i>	25
2.4.1.2 <i>Process Definition Tools</i>	26
2.4.1.2.1 <i>Definition of Processes</i>	26
2.4.1.2.2 <i>Classification of Resources</i>	27
2.4.1.2.3 <i>Analysis</i>	27
2.4.1.3 <i>Workflow Client Applications</i>	27

2.4.1.4	Invoked Applications	28
2.4.1.5	Other Workflow Enactment Services	28
2.4.1.6	Administration and Monitoring Tools	29
2.4.1.6.1	Operational Management Tool	29
2.4.1.6.2	Recording and Reporting Tool	29
2.4.1.7	Discussion of the Workflow Reference Model	29
2.4.2	Current Generation of Commercial WfMSs	30
2.4.2.1	IBM MQ Series Workflow	30
2.4.2.2	BEA WebLogic Integration	32
2.4.2.3	Future Prospects of WfMSs	34
2.5	Summary	36
Chapter 3	Adaptive Workflows	38
3.1	<i>Challenges in Adaptive Workflows</i>	39
3.2	<i>Projects Addressing Adaptive Workflows</i>	41
3.2.1	Workflow Change: Policies and Modalities	42
3.2.1.1	Modification Policies	42
3.2.1.2	Change Modalities	43
3.2.2	Proposed Solutions for Adaptive Workflows	45
3.2.2.1	Description of Key Projects	45
3.2.2.2	Workflow Meta-Model Expressiveness	48
3.2.2.3	Set of Changes Completeness	50
3.2.2.4	Summary of Correctness Verification	51
3.2.2.5	Discussion	53
3.3	<i>Adaptive Workflow Management Systems</i>	56
3.4	<i>Conclusion</i>	57
Chapter 4	Wf Technology Applied to Complex Socio-Technical Systems	59
4.1	<i>Workflow-Oriented Applications</i>	59
4.1.1	E-Business Domain	60
4.1.2	Medical Domain	61
4.1.3	Banking and Insurance Domain	63
4.1.4	Public Administration Domain	63
4.1.5	Why Studying Complex Socio-Technical Applications?	64
4.2	<i>The Combined Negotiation Application</i>	64
4.2.1	Description of the Application	65
4.2.2	Example of Combined Negotiation Packages	69
4.2.2.1	“Flight Connection” Package	69
4.2.2.2	“Importing” Package	72
4.2.3	The CONSENSUS System	74
4.2.3.1	WLPI Studio Unit	75
4.2.3.2	Enactment Unit	76
4.2.3.3	Coordination Unit	78
4.2.4	Towards a Dynamic Version of CONSENSUS	78
4.3	<i>The Multi-Transfer Container Transportation Application</i>	79
4.3.1	Description of the Application	80

4.3.2	Examples of Customer Request Processing Planning	84
4.3.2.1	Customer Request Processing Planning – Simple Example	84
4.3.2.2	Customer Request Processing Planning – Re-planning Example	87
4.3.3	Customer Request Processing	89
4.4	Summary	91
Chapter 5	The Enhanced CONSENSUS System	93
5.1	<i>Dynamic Aspects of the “Importing Package” Example</i>	94
5.2	<i>The CONSENSUS System Based on an Adaptive WfMS</i>	95
5.2.1	Dynamic Modifications Using ADEPT	95
5.2.2	ADEPT in CONSENSUS	98
5.3	<i>Adaptive Workflow Framework</i>	100
5.3.1	Adaptive Workflows and Transaction Management	103
5.4	<i>Summary and Discussion</i>	104
Chapter 6	Workflow Management Requirements	106
6.1	<i>Workflow Technology Enhancement</i>	107
6.2	<i>Enhanced Workflow Concepts and Functionality</i>	108
6.3	<i>Enhanced Workflow Concepts</i>	110
6.3.1	The Activity Template Concept	110
6.3.2	The Template Classification	111
6.3.3	The Activity Temporal Aspects	112
6.3.3.1	The Activity Starting/Finishing Time	113
6.3.3.2	The Activity Duration	114
6.3.3.3	The Activity WUT Concept – Integration of Preparation Activities	114
6.3.3.3.1	Dealing with the 1 st Disadvantage of the “Prep. Act.” Approach: Introduction of an Intermediate Work-list with a Listener Process	118
6.3.3.3.2	Dealing with the 2 nd Disadvantage of the “Prep. Act.” Approach: Defining Preparation Activities in the Background (First Solution)	119
6.3.3.3.3	Dealing with the 2 nd Disadvantage of the “Prep. Act.” Approach: A Layered Workflow Architecture (Second Solution)	120
6.3.3.3.4	Extension of the Warm-Up Time Concept – An Overview	122
6.4	<i>Enhanced Workflow Functionality Applied at the Workflow Instance Level</i>	123
6.4.1	The Dynamic Insertion of an Activity	123
6.4.1.1	The Dynamic Insertion of a New Activity Instance	123
6.4.1.2	The Dynamic Insertion of a Block of Activities	124
6.4.2	The Dynamic Deletion of an Activity	128
6.4.2.1	The Interruption of an Act. Execution While Preserving its Context	128
6.4.2.1.1	Formal Framework	131
6.4.2.1.2	Correctness Criterion	137
6.4.2.1.3	Discussion	140
6.4.3	The Dynamic Move of an Activity	142
6.4.4	The Dynamic Modification of Activity Attributes	142
6.4.4.1	The Dynamic Insertion/Setting/Updating of Input Attributes	142
6.4.4.2	The Dynamic Deletion of Input/Output Attributes	144
6.4.4.3	The Dynamic (Re-)Assignment of Activities to a Participant	145

6.4.4.4	The Dynamic Setting/Updating of Time Attributes	145
6.4.5	The Dynamic Management of Work-lists	146
6.4.6	The Automatic/Manual Modification of Workflow Instances	146
6.5	<i>Conclusion</i>	147
Chapter 7	The MTCT System	148
7.1	<i>The Transportation System Framework</i>	149
7.2	<i>Architecture of the MTCT System</i>	151
7.2.1	System Components	151
7.2.1.1	Build-time Components	152
7.2.1.2	Run-time Components	153
7.2.2	Underlying Management Mechanisms	154
7.2.2.1	Workflow Management	154
7.2.2.2	Resource Management	155
7.2.2.2.1	Static Resource Management	156
7.2.2.2.2	Dynamic Resource Management	156
7.2.2.3	Rule Management	158
7.2.2.3.1	Designing Modification Rules	158
7.2.2.3.2	Implementing Modification Rules	161
7.2.3	Interface of the MTCT System to External Systems	161
7.3	<i>Planning and Modifying the Processing of Customer Requests - Examples</i>	162
7.4	<i>Implementation of the MTCT System</i>	165
7.5	<i>Conclusion</i>	168
Chapter 8	Extension of the Specification of the WfRM	170
8.1	<i>Review of the Workflow Reference Model</i>	171
8.2	<i>The Proposed Extension</i>	174
8.2.1	Extension of Interf. 1 (Process Definition Tools)	175
8.2.2	Extension of Interf. 2 (Wf Client Apps) and Interf. 3 (Invoked Apps)	176
8.2.3	Discussion of Already Existing Components	177
8.3	<i>Functionality Extension of a WfMS</i>	178
8.3.1	Structural Modifications	179
8.3.2	Activity Attributes Modification	179
8.3.3	Work-lists Management	180
8.3.4	Discussion of Current Implementation	181
8.4	<i>Conclusion</i>	181
Chapter 9	Conclusion	183
9.1	<i>Summary and Discussion</i>	183
9.1.1	The CONSENSUS and the MTCT Applications as Drivers of Sophisticated Requirements for Workflow Technology	183
9.1.2	The Identification and the Accommodation of Sophisticated Requirements for Workflow Technology	184
9.1.3	The Extension of the WfRM to Adequately Support Enhanced Workflow Technology	187
9.1.4	Further Discussion	188

9.2	<i>Research Perspectives</i>	188
References		191
Appendix A Extending the Workflow Reference Model: Workflow Management Application Programming Interface Specification-----I		
A.1	<i>Compressed Summary of the Groups of Operations and Operations</i>	I
A.2	<i>Detailed Summary of the Groups of Operations and Operations</i>	IV
A.3	<i>Description of the Extended WAPI Specification</i>	XII
A.3.1	Inserting Activities	XII
A.3.2	Deleting Activities and Templates	XXVII
A.3.3	Moving Activity Instances	XXIX
A.3.4	Setting and Updating Attribute Values	XXX
A.3.5	Inserting Attributes	XXXII
A.3.6	Deleting Attributes	XXXIV
A.3.7	Role/User Assignment	XXXVI
A.3.8	Time Attributes Assignment	XXXVIII
A.3.9	Keeping Modified Process Instances	XLIV
A.3.10	Inserting Sub-Workflows	XLV
A.3.11	Managing Work-lists	XLVI
A.4	<i>WAPI Data Types Addendum</i>	XLIX
A.5	<i>WAPI Error Return Codes Addendum</i>	LI
References of Appendix A		LII

List of Figures

Figure 2.1. Ad-hoc, Collaborative, Administrative, and Production Workflows-----	12
Figure 2.2. The Process Definition Meta-Model, taken from [WfMC99a] -----	14
Figure 2.3. Medical Treatment Process -----	18
Figure 2.4. Example of an Organizational Meta-model, adapted from [RT02]-----	22
Figure 2.5. Example of an Organizational Model (Tree Structure) -----	22
Figure 2.6. WfRM – Components and Interfaces, taken from [WfMC95]-----	24
Figure 2.7. Groups of Operations Distributed within the Five Interfaces of the Workflow Reference Model, based on [WfMC95] -----	25
Figure 2.8. Interoperability Models, adapted from [WfMC95]. (a) Chained, (b) Nested Sub-Processes, (c) Peer-to-Peer, (d) Parallel Synchronized -----	28
Figure 3.1. Loop Tolerance in ADEPT/WSM-Nets, adapted from [RRD04a] -----	54
Figure 3.2. Markings Adaptation using the SCOC – Syntactic Cut Over Change – in ML-DEWS/Flow Nets, adapted from [RRD04a] -----	55
Figure 4.1. Flight Connection Package Workflow Model in WLPI -----	71
Figure 4.2. Importing Package Workflow Model in WLPI -----	74
Figure 4.3. CONSENSUS based on BEA Systems WLPI, adapted from [BBK01] -----	75
Figure 4.4. WLPI Studio Unit. (a) Workflow Variables, (b) Invoking a Business Operation, (c) List of Business Operations -----	76
Figure 4.5. Agent Control and Monitoring Tool -----	77
Figure 4.6. Example of a Transportation Network, adapted from [Tra04] -----	85
Figure 4.7. Re-planning Example. (a) The Proposed Modifications for the Processing of OR, (b) The Proposed Solution for the Processing of NR-----	89

Figure 5.1. Importing Package during Run-time in ADEPT – Modeled without Decision Branches. Instance State (a) After Creation, (b) After Moving Task F, (c) After Deleting Tasks: T1, T2, T3, and I-----	96
Figure 5.2. “Importing Package” in ADEPT – Modeled with Decision Branches. (a) The Whole Picture, (b) Detailed Part of the Process -----	97
Figure 5.3. WLPI Methods Called by the ADEPT Client Application for the Implementation of Negotiation Activities-----	99
Figure 5.4. Adaptive Workflow Framework-----	100
Figure 5.5. Sequence of Messages Exchanged (a) during a Normal Execution of a Workflow Instance, (b) when an Activity Insertion is Required, and (c) when an Activity Deletion is Required -----	102
Figure 6.1. Workflow Technology Enhancement-----	108
Figure 6.2. Integrating “Preparation Activities” to a Workflow. (a) A Workflow with Two Activities (“a” and “b”) Defined in Sequence, (b) Integrating “Prep: a”, (c) Integrating “Prep: b”-----	116
Figure 6.3. The Mechanism of an Intermediate Work-list with a Listener Process ----	119
Figure 6.4. Sending “Preparation Activities” to the Background -----	120
Figure 6.5. Explanation of the Layered Workflow Architecture for the Support of the WUT Concept -----	120
Figure 6.6. The “Proclat” Idea for the Support of the WUT Concept -----	122
Figure 6.7. Steps for the Dynamic Insertion of a Sub-Workflow -----	126
Figure 6.8. Valid Structure of the Workflow Resulting from Step 1-----	126
Figure 6.9. Valid Structure of the Workflow Resulting from Step 2-----	127
Figure 6.10. Example of a Sub-Workflow Including a Loop-----	127
Figure 6.11. Data Classification Scheme-----	131
Figure 6.12. Medical Treatment Process (Atomic Steps) -----	133
Figure 6.13. Container Transportation Process-----	135
Figure 6.14. Data Classification in the Medical Treatment and Container Transportation Processes-----	137
Figure 6.15. Container Transportation Scenario -----	140

Figure 7.1. Transportation System Framework-----	149
Figure 7.2. Different Steps from the Detection of an Event till the Instantiation/Change of Workflow Instances -----	150
Figure 7.3. Architecture of the MTCT System -----	152
Figure 7.4. Entity-Relation Diagram for the Resource Management in the MTCT System -----	156
Figure 7.5. Example in ADEPT of a Planned Unavailability Workflow Instance for the Two Drivers McCain and Watson -----	156
Figure 7.6. Workflow Instance Creation and Adaptation Following a Request Arrival – State Diagram -----	160
Figure 7.7. A Transportation Network Representation: Resources Represented as Icons in a Simulation Environment -----	162
Figure 7.8. “Request Information” Form-----	163
Figure 7.9. A Modification Rule of the Pool of Workflow Instances-----	164
Figure 7.10. The Added <i>Mediator</i> Component within the ADEPT Structure -----	166
Figure 7.11. Screenshot of the MTCT System Version 0.1. (a) The Environment of the System Administrator, (b) The Environment of the Drivers -----	168

List of Tables

Table 2.1. Wf Modeling Formalisms and Wf Management Systems -----	37
Table 3.1. Adaptive Wfs Key Projects – Wf Meta-Model Expressiveness -----	48
Table 3.2. Adaptive Wfs Key Projects – Correctness Verification of Changes -----	51
Table 4.1. Act. Templates Involved in the Proc. of a Cust. Request for Cont. Transp. --	81
Table 4.2. Duration Between Two Locations (in minutes)-----	85
Table 8.1. Groups of Ops Distributed within Interfaces 1, 2 and 3 of the WfRM-----	173

List of Acronyms

ADEPT: Application Development based on Encapsulated pre-modeled Process Templates

API: Application Programming Interface

B2B: Business to Business

B2C: Business to Consumer

BPEL4WS: Business Process Execution Language for Web Services

BPM: Business Process Management

C2C: Consumer to Consumer

CIRANO: Centre Interuniversitaire de Recherche en ANalyse des Organisations

CN: Combined Negotiation

CONSENSUS, CNSS: Combined Negotiation Support System

CORBA: Common Object Request Broker Architecture

DWM: Dynamic Workflow Model

ECA: Event-Condition-Action

EFT: Earliest Finishing Time

EST: Earliest Starting Time

FDL: Workflow Definition Language

GNP: Generic Negotiation Platform

GPS: Global Positioning System

ICN: Information Control Net

IT: Information Technology

JDBC: Java Database Connectivity

LFT: Latest Finishing Time

LST: Latest Starting Time

MCS: Minimal Critical Specification

ML-DEWS: Modeling Language to support the Dynamic Evolution within Workflow Systems

MR: Modification Rule

MTCT: Multi-Transfer Container Transportation

OM: Optimization Model
OPL: Optimization Programming Language
PDP: Pick-up and Delivery Problem
ST: Starting Time
TSE: TRP (Technical Reinvestment Project) Support Environment
UML: Unified Modeling Language
URL: Uniform Resource Locator
WAP: Wireless Application Protocol
WAPI: Workflow Application Programming Interface
WAR: Write After Read
WARIA: Workflow and Reengineering International Association
WARP: Workflow Automation through Agent-based Reflective Processes
WAW: Write After Write
Wf: Workflow
WfMC: Workflow Management Coalition
WfMS: Workflow Management System
WfRM: Workflow Reference Model
Wf-XML: Workflow Extensible Markup Language
WLPI: WebLogic Process Integrator
WNM: Workflow Net Model
Woflan: WorkFlow ANalyzer
WPDL: Workflow Process Definition Language
WSCl: Web Services Choreography Interface
WSCL: Web Services Conversation Language
WSDL: Web Services Description Language
WSM: Workflow Sequential Model
WSM-Nets: Well-Structured Marking-Nets
WUT: Warm-Up Time
XML: Extensible Markup Language
XPDL: XML Process Definition Language
XRL: eXchangeable Routing Language

*I dedicate this thesis to my parents, Jean and Colette.
This thesis could never have been accomplished
without their unconditional love and support.
You are the warmth inside my heart and the reason in my soul.
You made me what I am now.*

*I dedicate this thesis to Joanna, my dear sister and friend.
To your affection, love and kindness.*

May God bless you and keep you safe.

Acknowledgments

I would like to thank Rudolf K. Keller, associate professor at the University of Montreal, for giving me the chance to join his team at CIRANO (*Centre Interuniversitaire de Recherche en Analyse des Organisations*), and for supervising this research. His determination, his critical sense and his clear sightedness helped me to progress in the best research axes.

My thanks also go to Peter Kropf, professor at the University of Neuchâtel (Switzerland), for co-supervising this thesis. His availability, all the valuable discussions I had with him, and his encouragement were a significant factor in the success of this work. I thank him as well for inviting me to spend six months at the University of Neuchâtel, and for supporting me all along my stay. This stay helped in speeding up the writing process of the thesis.

I would like to thank Houari Sahraoui, professor at the University of Montreal, for presiding my jury. Yann-Gaël Guéhéneuc, professor at the University of Montreal and member of my jury, may also find here the expression of my appreciation. Furthermore, I thank Jacques Nantel, professor at HEC, for representing the Faculty Dean at my thesis defense.

My gratitude also goes to Kostas Kontogiannis, professor at the University of Waterloo, for accepting to be the external examiner. I thank him for having encouraged me once, during our meetings at CSER (Consortium for Software Engineering Research), to work on a Ph.D.

I would also like to thank the NSERC (Natural Sciences and Engineering Research Council of Canada), Bell Canada and CIRANO. Definitely, the completion of this research was made possible thanks to funding provided by the NSERC (CRD-224950-99),

Bell Canada's support through its Bell University Laboratories R&D program, and support by the CIRANO. I thank each of the CIRANO's researchers and employees.

Thanks to Morad Benyoucef, former Ph.D. student at the CIRANO and now professor at the University of Ottawa, for introducing me to the CONSENSUS project and for his collaboration.

I would like to thank Peter Dadam, professor at the University of Ulm (Germany), for making me the honor to be interested in my research and to invite me to spend two weeks at the Database and Information System Department. My special thanks go to Manfred Reichert, former junior-professor at the University of Ulm and now professor at the University of Twente (The Netherlands), for the valuable comments he made during our numerous exchanges and for his friendship. My sincere appreciation and admiration also go to my collaborator and friend Stefanie "Steffi" Rinderle for her contribution to my research. Moreover, I thank Steffi and Manfred for making my stay in Germany an enjoyable experience.

Finally, I thank every person caring about me. In particular, my special and sincere thanks go to Dr. Roger Hakimian, a dear friend, for always being close to me despite the geographical distances between us. I am also grateful to every member of my family, especially those in Lebanon, for their prayers, attitude and nice words.

1.1 Problem Statement

For competition purposes, today's organizations are forced to streamline their way of doing business. In this context, often a process logic is applied. It consists of focusing on the *business processes* described within these organizations. A business process is defined as a set of one or more linked *activities*, which collectively realize a business objective [WfMC99b]. Specifically, these activities are carried out, in a coordinated way, by different processing entities, including humans and software systems, in order to reach a goal, such as delivering merchandise or operating a patient. Since organizations typically work in dynamic environments their business processes require to be just-in-time modified.

Workflows correspond to a technology that aims to provide as much computer support as possible to the management of business processes. This technology has gained great attention in recent years because the success of organizations is more and more associated with the effective use of information technology, mainly to support their business processes. *Workflow Management Systems* (WfMSs) allow for capturing formal descriptions of business processes and for supporting the automatic enactment of processes based on those formal descriptions. We say that WfMSs support the *modeling* (i.e., build-time) and *enactment* (i.e., run-time) of workflows, and we differentiate between a *workflow model* and a *workflow instance* which is the representation of a single enactment of a workflow model. In particular, systems that support the workflows in a specific business situation or that are adapted to a particular application are called *workflow-oriented* (or *workflow-based*) *systems*. They consist of a WfMS in addition to the application-specific modules.

By adopting a specific business solution, such as implementing workflow-oriented systems, organizations are usually interested in optimizing their profits as much as possible. This optimization goal is however sometimes compromised when, for instance, we are forced to follow a predefined set of linked activities without being able to take into account real-time events, possibly coming from the external environment, and to react appropriately with profitable adaptations. The adaptation problem in workflows, also known as the area of *adaptive workflow technology*, was not yet addressed by the business process management community in a significant manner with respect to real-world applications:

- The Workflow Reference Model (WfRM) [WfMC95] developed by the Workflow Management Coalition¹ (WfMC) [WfMC04] as an overall model for workflow management systems does not support refined workflow issues such as adaptive workflows. This lack of support reduces the benefits of workflow technology, and discourages the building of sophisticated workflow-oriented systems.
- *Workflow meta-models* that form an integral part of WfMSs and that (1) include a set of modeling concepts used to define workflow models, and (2) support the specification of workflow aspects that are relevant for enactment (e.g., the control flow, the data flow, and the assignments of activities to processing entities), are not expressive enough to allow practically relevant modifications. As an example, if a workflow meta-model does not explicitly consider data flows, there would be no way to deal with data during workflow modifications: the insertion/deletion of data would not be possible; furthermore, the *correctness verification* regarding the application of modifications on workflow instances will not include the verification of data, i.e., whether data are correctly provided or not.
- The sets of allowed modification operations proposed by current adaptive workflow projects are still incomplete. Particularly, the studied modification opera-

¹ The WfMC, founded in August 1993, is a non-profit, international organization of workflow vendors, users, analysts and university/research groups. Its goal is to develop standards for workflow systems operation, and to promote knowledge of the technology within the industry.

tions are limited to workflow *structural modifications* (i.e., modifications at the control flow level, such as inserting/deleting an activity). Activity *attribute modifications* (i.e., modifications at the data flow level, such as inserting/deleting an activity attribute) are not addressed.

- The correctness criteria defined to verify that a workflow instance is compliant with the proposed modifications (i.e., modifications will not cause inconsistencies or errors for the rest of the workflow instance processing) are sometimes too restrictive.

The study of sophisticated workflow requirements, including in particular the need for modification facilities that allow for adapting workflow instances during run-time, is the principal subject of this work. We address adaptiveness at the workflow enactment level by introducing new modification operations and their corresponding correctness criteria, and at the modeling level by further developing existing workflow concepts, defined by workflow meta-models, to include flexibility.

We realized that only real-world applications reflect the relevant needs for workflow technology. Our research environment, the CIRANO – *Centre Interuniversitaire de Recherche en ANalyse des Organisations*, gathers expertise in economic science and operations research. Hence, this represents a most valuable opportunity to consider applications and to address systems in the context of these specific fields. We chose to talk about *complex socio-technical systems* to reflect the fact that multiple actors are using such systems in a coordinated way requiring the management of shared resources (i.e., social aspect), that the applications addressed by those systems stem from technical fields (i.e., technical aspect), and that these systems need to be reactive, they may also involve a number of technologies, such as optimization engine technology and rule engine technology, in conjunction with the workflow technology (i.e., complexity aspect).

This doctoral research has partly been conducted within the TEM (Towards Electronic Marketplaces) project, a joint industry-university project supported by the Bell University Laboratories, NSERC (National Sciences and Engineering Research Council of Canada), and CIRANO. The objective of this project was to address market design is-

sues in respect to resource allocation and control, and reward mechanisms, to investigate open protocols for e-marketplaces, and to explore concepts and tools for e-business. The work that we carried out was partly associated with the last topic. In particular, a workflow-based support system for e-business application has been studied (i.e., the CONSENSUS application). Our work went however beyond the only e-business application by exploring an application from the transportation domain as well (i.e., the MTCT application). CONSENSUS and MTCT are from quite different domains. This adds to the generality of our research.

1.2 Research Objectives

The objectives of this research project are summarized as follows:

(1) **The extension of the WfRM to adequately support adaptive workflows.**

Currently, the WfRM is a generic, domain-independent model. It only supports basic workflow aspects. However, successful workflow-oriented systems are the ones that are tailored for particular applications stemming from specific domains (e.g., the medical domain):

- We decided in our research to focus on specific domains. This approach has the potential of being a constructive method for deriving, as an extension of the WfRM, an architectural framework for adaptive workflows.
- We will demonstrate that the construction of effective workflow-oriented systems requires an extended WfRM. For this purpose, an already existing workflow-oriented system will be reviewed and extended for better supporting an e-negotiation application.

(2) **The identification and specification of the extension requirements for the next generation of adaptive workflows.** The identified list of requirements shall enable better specifications to be developed within the context of the extended WfRM:

- We will address these requirements in the *best* appropriate manner to make them available from an existing Workflow Management System (WfMS), thus following a best practice policy.
- We will demonstrate that a formal specification can be provided for the support of such requirements. This specification will extend a state-of-the-art workflow modeling language. In particular, we will present a novel solution to the problem of workflow activity interruption. We will show how to preserve the context of interrupted activities.

(3) **The design of an adaptive workflow system architecture that respects the extended WfRM:**

- We will show that the extended model encourages the construction of effective adaptive workflow-oriented systems. For this purpose, a concrete architecture that stems from the extended WfRM will be devised. As a proof of concept, this architecture will be adapted to a transportation application.
- We will implement an adaptive workflow-oriented system prototype to evaluate the quality and scope of the model extension and of the derived architecture.

A conventional research approach would suggest to formally study workflow requirements and then validate the requirements in respect to specific applications. In our research, we took a slightly different approach. We set off by investigating a number of specific applications from where we derived workflow requirements as input for our research. This application-driven approach proved quite effective in identifying realistic requirements and in providing solutions that are readily applicable to real-world problems.

1.3 Major Contributions

In the course of this research project, we have extended the WfRM. A new architectural framework for adaptive workflows has been proposed (cf. Chapter 5):

- The model allows for designing concrete workflow-oriented system architectures in the context of specific applications stemming from specific domains.
- The model supports adaptive workflows.

We have identified a list of requirements for adaptive workflows. This list includes new concepts, such as the activity template concept, the atomic step concept and the activity warm-up time concept, and enhanced functionalities, such as the interruption of an activity execution, the dynamic move of an activity, and the dynamic modification of activity attributes. As a proof of concept, we have proposed best effort solutions (proper use of WfMS features, workarounds, etc.) to address these requirements based on three state-of-the-art WfMSs. The problems encountered while experimenting with those systems may give WfMS builders valuable input for future versions of their products, and may define an agenda for further research in the domain (cf. Chapter 6).

Particularly, we have proposed a formal framework to correctly address the issue of safely interrupting running workflow activities in case of exceptional situations. This issue is a major requirement for the next generation of adaptive workflows. In the context of this framework, we have introduced a lower level of granularity to the modeling of activities by defining the atomic step concept, and we have elaborated a data classification scheme that puts the frequency of updating activity data and the relevance of these data into relation. We have formally derived a correctness criterion for the safe interruption of a running activity (i.e., interrupting a running activity by keeping its context) (cf. Chapter 6).

We have extended the combined negotiation support system CONSENSUS², an e-

² CONSENSUS is a workflow-based system that helps the user model and enact a combined negotiation. Combined negotiations are defined as a novel and a general type of negotiation in which the user is interested in a package of goods or services and consequently engages in many negotiations at the same time [BAV+01].

business application that was developed by Benyoucef [BAV+01] (cf. Chapters 4 and 5):

- The extended CONSENSUS supports dynamic modifications induced by unexpected events possibly occurring during negotiations. It allows for moving/deleting an already scheduled e-negotiation activity, for inserting a new activity, and for changing the attributes of an activity.
- We used the WfMS prototype ADEPT³ [RRD03a, RT02] in order to accommodate these modifications. We have proposed an extension to ADEPT to support the whole set of modification operations required by CONSENSUS, and to allow the “automatic call” feature for the implementation of e-negotiation activities.

Finally, based on the designed architectural framework for adaptive workflows and taking into account the identified list of workflow requirements and the proposed solutions to address these requirements, we have proposed an original adaptive workflow-oriented system applied in the transportation domain: the multi-transfer container transportation system MTCT. It allows for the processing of customer requests for container transportation. In this context, an innovative integration problem involving workflow technology, optimization engine technology and rule engine technology, was studied. This should give interesting input for the development of new solutions and tools in the transportation domain (cf. Chapter 7).

Various aspects of this work have already been published in the proceedings of the international conferences and workshops: BPM'2004 (*International Conference on Business Process Management*) [BKK04], ICECR-4 (*International Conference on Electronic Commerce Research*) [BBK01], ODYSSEUS'2003 (*International Workshop on Freight Transportation and Logistics*) [BBK+03] and DEXA'2002 (*International Workshop on Database and Expert Systems Applications, International Workshop on Negotiations in e-Markets*) [BBK+02a]. A further paper has recently been accepted for publications in

³ ADEPT (Application Development based on Encapsulated pre-modeled Process Templates) is one of the few available WfMS research prototypes dealing with adaptive workflows. It offers temporal constraint management, workflow modifications, synchronization of inter-workflow dependencies, and scalability [RRD03a].

the proceedings of the international conference ICEIS'2005 (*International Conference on Enterprise Information Systems*) [BRK+05].

1.4 Thesis Structure

The remainder of this thesis is structured as follows. Chapter 2 gives a general overview of related work in the area of processes, workflows, and workflow management systems. Particularly, we review the WfRM as defined by the WfMC. In Chapter 3 we focus on adaptive workflows. We provide a state-of-the-art assessment on existing workflow modification projects and shed the light on extension points. Chapter 4 presents two different applications (the CONSENSUS application and the MTCT application) that outline (1) the need for an adaptive workflow framework, and (2) the requirements to address new concepts and functionality in workflow technology. In Chapter 5 we discuss an extended version of CONSENSUS, and we provide an overall architecture as an extension to the WfRM for supporting adaptive workflows. The CONSENSUS application, its extension, and the description of the MTCT application serve as a motivation to Chapter 6, which presents possible solutions to specific problems in adaptive workflow systems. In particular, this chapter presents a novel solution to the problem of workflow activity interruption. Chapter 7 introduces a system architecture as a solution to the MTCT problem using adaptive workflows. This architecture stems from the overall architecture provided in Chapter 5. Chapter 8 proposes a detailed extension to a workflow application programming interface (WAPI) to support adaptive behavior within the context of the WfRM. Further details about the extension of the WfRM WAPI specification are annexed to the thesis in Appendix A.

Each of these chapters is ended either by a “Conclusion” section, a “Summary and Discussion” section or simply a “Summary” section. A “Conclusion” section refers to an analytic conclusion where a qualitative analysis of the chapter or of some issues related to the chapter is given. The last chapters of the thesis are ended with a “Conclusion” section (Chapters 3, 6, 7 and 8), while the first chapters are rather ended with a “Summary” or a “Summary and Discussion” section (Chapters 2, 4 and 5).

Chapter 2 Processes, Workflows, and Workflow Management Systems

Complex tasks must be structured with some model representation to facilitate their management as well as the automation of their execution. Workflow technology has been proposed to deal with this kind of tasks. The WfMC proposes a definition of workflow that is widely used within the literature. A workflow is considered as “the automation of a *business process* – defined as a set of one or more linked activities, which collectively realize a business objective –, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules” [WfMC99b]. From a more general perspective, not necessarily related to the business world, a process is defined as a set of partially ordered steps involved in reaching a goal [CKO92].

To support automation, a Workflow Management System (WfMS) can be defined as a software that manages a workflow efficiently by tracking and controlling its execution. It supports the definition, the execution, and the monitoring of a workflow [WfMC99b].

This chapter is structured as follows. Section 2.1 explains the terminology related to processes and workflows and reviews workflow classifications. Sections 2.2 and 2.3 consider in detail the two main constituents of workflow management: workflow design and workflow enactment. Section 2.4 addresses WfMSs: an emerging standard is reviewed and specific WfMSs are studied. The chapter is summarized in Section 2.5.

2.1 Workflow Basics and Classifications

In order to set up a nomenclature for specifications, as well as for discussions among users, analysts, and researchers, the basic terms related to processes and workflows need to be defined. Many papers propose a terminology relating concepts as well as relation-

ships among them [DNR90, FH92, LS97, WfMC99b]. The concepts defined by the WfMC [WfMC99b], and then refined by van der Aalst and van Hee in [AH02], are the most widely applied ones within the business process management community. The following list presents the basic workflow concepts and structures for workflow design, workflow enactment, and the organizational configuration, as suggested by the WfMC:

- An *activity* (node, task) is a description of a piece of work that forms one logical step within a workflow. It can be manual or automatic [AH02]. A manual activity is entirely performed by one or more people, without any use of an application. By contrast, an automatic activity is performed without any intervention by people; an application – a computer program – carries out the activity entirely based upon previously recorded data. Activities are ordered based on the mutual dependencies imposed by structural and data aspects (control flows and data flows between activities). Various configurations cover the structural aspects: sequence, selection, iteration, and concurrency. Two approaches are most commonly used for the representation of data: either through data flows between activities, or through data provision services from/to which activities read/write.
- An *instance* (workflow instance (case), or activity instance) is the representation of a single enactment of a workflow, or activity within a workflow.
- A *participant* (actor, agent, user, processing entity, resource) is the construct that performs an activity instance. It may range from humans to software systems.
- A *work-item* is the representation of the work to be processed (by a participant) in the context of an activity within a workflow instance. A list of work-items associated with a given workflow participant (or group of workflow participants) is called a work-list.
- A workflow (resp., activity) *state* is related to the internal conditions defining the status of a workflow (resp., activity) instance at a particular point in time. In the case of a workflow, the state could be “initiated”, “running”, “active”, “suspended”, “completed”, “terminated”, and “archived”. In the case of an activity, it could be “inactive”, “active”, “running”, “suspended”, “skipped”, and “completed”. Variants of these terms are found within the literature, as well as when considering specific workflow products.

- An *organizational model* is a model that represents organizational entities and their relationships; it may also incorporate a variety of attributes associated with the entities, such as skills or role.
- An *organizational role* is a group of participants exhibiting a specific set of attributes, qualifications and-or skills. A workflow participant assumes a role given that she has the appropriate skill set.

Two major workflow classification schemes have been proposed in the literature [McC92, LR99, GT98, GHS95]:

- (1) *Ad-hoc, collaborative, administrative, and production workflows (Figure 2.1) [McC92, LR99, GT98]*. These four kinds of workflows are categorized according to their business value and their repetitiveness. Ad-hoc workflows and collaborative workflows involve participants collaborating to reach a certain goal. Usually, no workflow model is defined in advance because of little repetitiveness. Collaborative workflows (e.g., preparation of product documentation) have a higher business value than the ad-hoc workflows (e.g., meeting scheduling). Administrative workflows and production workflows have a high repetitiveness. Workflow models can be predefined for them. Production workflows (support of an organization's core business; e.g., claims-handling in an insurance company) have a higher business value than administrative workflows (e.g., processing a salary calculation). In this thesis, we address workflows with little repetitiveness (and low business value) but which can be instantiated from a basic workflow model (refer to the multi-transfer container transportation application presented in Section 4.3). Combined (business) negotiation workflows (cf. Section 4.2) can be considered either as collaborative workflows (if B2C/C2C) or production workflows (if B2B, e.g., support of the main business of a travel agency or of an import/export company).

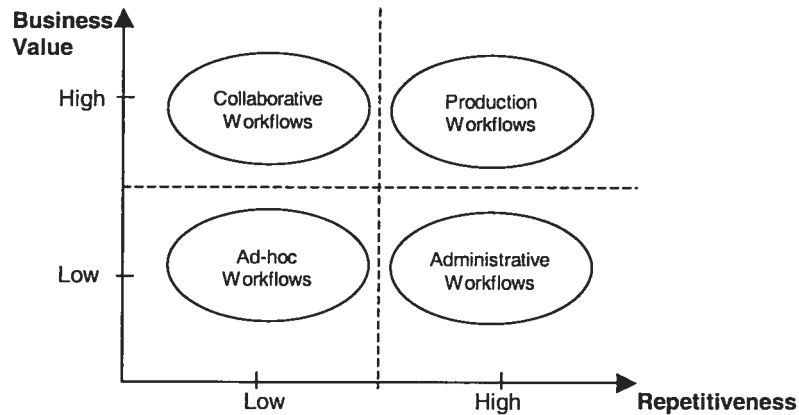


Figure 2.1. Ad-hoc, Collaborative, Administrative, and Production Workflows

(2) *Human-oriented, system-oriented, and transactional workflows [GHS95].* The activities in human-oriented workflows are carried-out by humans. Human-oriented workflows are comparable to ad-hoc and collaborative workflows. System-oriented workflows involve computer systems that perform computation-intensive operations and specialized software tasks. Transactional workflows [AAE+96, WS97] are a special kind of system-oriented workflows. The database community focuses on this kind of workflows. The main motivation for introducing the concept of transactional workflows was to address the WfMS's inability to ensure the correctness and reliability of workflow executions in the presence of concurrently executing workflows and failures. In this thesis we take an abstract, conceptual view of workflows with no emphasis on transactional workflows. A workflow is simply considered to consist of a set of activities with data and control flow dependencies among them, where the activities are executed by participants that may include humans as well as software agents.

2.2 Workflow Design

Two types of methodologies are basically used to design or model a workflow: communication-based methodologies and activity-based methodologies. The former focus on modeling the communications among workflow participants while the latter focus on

modeling activities. WfMSs typically adopt activity-based methodologies, and in this thesis, this type of methodologies will mainly be considered.

In spite of the standardization efforts taking place in the WfMC (cf. Section 2.4.1), no generally accepted workflow meta-model has been defined so far. Textual or graphical workflow modeling languages provide concrete constructs for the concepts of an underlying meta-model. In the context of activity-based methodologies, most of the workflow modeling languages discussed in the literature are based on formalisms such as Petri nets [Petri04] and UML [UML04] (including state and activity charts). Petri nets are known for their rigorous semantics, and UML is widely used these days because of its object-oriented paradigm. However, a workflow modeling language based on one of these formalisms provides users, especially non-computer experts, hardly an intuitive and structured representation of a business process [RD98]. Furthermore, these formalisms do not offer a detailed structure for the definition of workflow aspects.

To facilitate specific purposes, e.g., to address adaptive workflows (cf. Chapter 3), some researchers developed their own workflow modeling languages that rely neither on Petri nets nor on UML. An example of such a workflow modeling language is the ADEPT model (WSM-Nets) based on the concept of symmetrical control structures [RD98]. The Workflow Process Definition Language (WPDL) defined by the WfMC [WfMC99a], remains the only consortium-led language providing constructs that focus specifically on workflow aspects. Its related process definition meta-model has been specified to capture the highest-level objects and relationships that should be defined to support process automation (Figure 2.2). An extension of this meta-model to support dynamic inter-organizational workflow management has been proposed by Meng in her Ph.D. thesis [Men02].

Finally, XML-based representations are often discussed in workflow-based inter-organizational e-business applications (cf. Section 4.1.1) [KZ02, LO01, AK00]. XML (Extensible Markup Language) [XML04] is a document declaration standard proposed by the WWW Consortium [W3C04] that allows the electronic exchange of semantic information. XML on its own does, however, not provide support for document routing and data interchange between the organizations involved. Lenz and Oberweis propose

XML nets – a new kind of high-level Petri nets – that allow to model both the flow of XML documents and the business process [LO01]. Van der Aalst and Kumar propose XRL (eXchangeable Routing Language) for document routing [AK00, VHA02]. XRL is also expressed in terms of Petri nets. Another XML-based representation to support inter-organizational applications, published by the WfMC, is Wf-XML [WfMC01]. It is intended as a basis for concrete implementations of the WfMC's Interface 4 (cf. Section 2.4.1). Wf-XML relies on WPDL for routing issues. The XML version of the WfMC's WPDL is called the XPDL (XML Process Definition Language) [WfMC98].

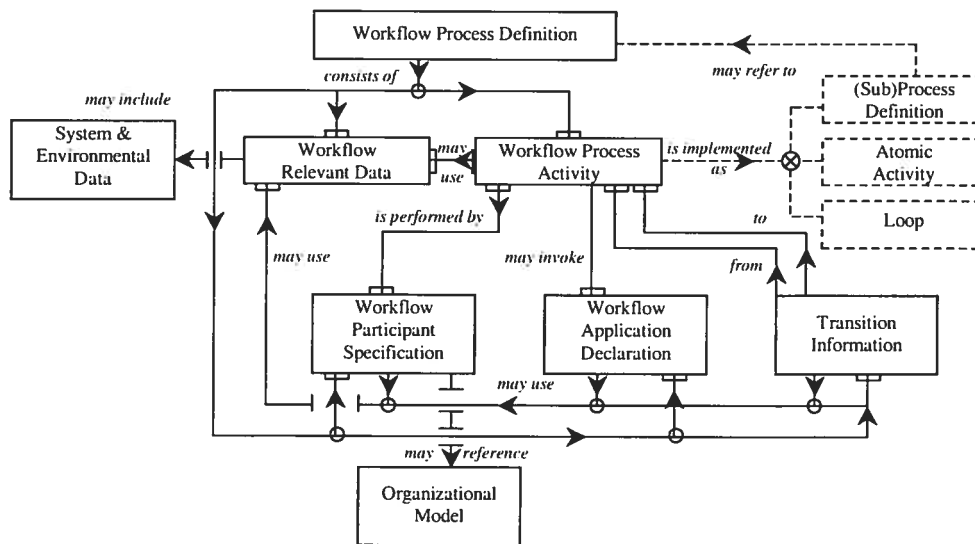


Figure 2.2. The Process Definition Meta-Model, taken from [WfMC99a]

In the context of e-business applications, choreography languages have been proposed by imposing companies and organizations (e.g., IBM, Microsoft, W3C) for the composition or orchestration of Web Services. The emerging of such languages underlines the timeliness of this research project. Examples of choreography languages include Web Services Conversation Language (WSCL) [WSCL02], Web Services Choreography Interface (WSCI) [WSCI02] and Business Process Execution Language for Web Services (BPEL4WS) [BPEL03]. The latter, developed by IBM and considered as a standard, seems to have a lot of momentum. The Web Services Description Language (WSDL) [WSDL01] is the XML-based specification used for describing the operational information of Web Services (e.g., input and output messages) (i.e., function logic) and

BPEL4WS allows for defining business processes letting several Web Services from different service providers work together (i.e., flow logic). BPEL4WS provides a long-running transaction model that allows increasing consistency and reliability of Web Services applications. A collection of “primitive” activities (e.g., invoke, receive, reply) – that we may also call “activity templates” – and “structure” activities (e.g., sequence, switch, while) is defined. The invocation of services is done using the “invoke” activity, while the reception of an invocation from a client is done using the “receive” and “reply” activities. Hence, the WfMC’s Interface 3 (Invoked Applications; cf. Section 2.4.1) is well defined by BPEL4WS.

In the next sections, we begin by briefly describing both the Petri net and UML formalisms with respect to workflows (Section 2.2.1 and Section 2.2.2). Then, we present the WSM-Nets formalism on which we rely to formally introduce in Chapter 6 new workflow concepts and functionality (Section 2.2.3). Workflow temporal aspects will be discussed thereafter as a further issue in workflow design (Section 2.2.4). The organizational structure is part of workflow design and will be introduced as well (Section 2.2.5).

2.2.1 Petri Nets and Workflows

“Petri nets” [Petri04] is a major formalism for modeling workflows. One of the strengths of Petri nets is the strong mathematical basis they offer along with a graphical representation. In this section, we summarize the mapping between workflow concepts and Petri nets [JB96, AAH98, AH02].

A process defines tasks as well as the conditions for their execution. Using Petri nets, a process is represented by mapping its only entrance (i.e., start node) into a place without incoming arcs, and its only exit (i.e., end node) into a place without outgoing arcs. Conditions are mapped into places, and tasks into transitions. Usually, a process specified using Petri nets should fulfill two requirements: (1) it should at any time be possible to reach a state in which there is a token in “end”, and (2) when there is a token in “end”, all the other tokens should have disappeared.

Different instances of the same process can be translated into Petri net models in two ways: (1) produce a separate copy of the Petri net (i.e., process) for each instance, (2)

use just one Petri net by making use of the color extension [Petri04]. Each token will then be provided with a color or value from which it is possible to identify the instance to which the token refers.

Tasks may need to be carried out for certain instances and not for others. The order in which tasks are performed may also vary from one instance to another. Routing permits to determine which tasks need to be carried out and in what order. Basic constructions for routing (sequential, selective, iterative, and parallel routing) are associated with specific Petri net compositions such as “two transitions linked using a place” to represent a sequence of two activities, “two transitions” to model the And-split and the And-join of a parallel routing, and “a place” to model a condition for a selective/iterative routing.

In a process modeled with a Petri net, an enabled transition corresponds to a work-item, and the firing of a transition to an activity instance. Certain work-items can only be transformed in an activity instance once they are triggered. A trigger could correspond to a participant initiative, to an external event or to a time signal coming from the environment. To each transition belonging to a task requiring a trigger an extra input place is added. A trigger occurrence brings a token in that extra input place. The token is consumed once the appropriate transition fires. A failure while performing a task requires a rollback (i.e., go back to the state prior to the start of the activity). When an activity has been successfully completed, a commit occurs and changes become definitive.

2.2.2 UML and Workflows

State and activity charts are another major formalism for the modeling of workflows. They were originally invented by Harel [Har87], and have been incorporated into UML (Unified Modeling Language) [UML04] in a slightly different form. Weissenfels *et al.* [WMW98] have investigated the use of state and activity charts to model workflows (the Mentor WfMS project), while Blake [Bla02, Bla00] presents a systematic approach to the modeling of workflows using UML (the WARP project).

In the Mentor WfMS project [WMW98], activities reflect the functional decomposition of a system and denote the active components of a specification; they correspond directly to the activities of a workflow. An activity chart specifies the data flow between

activities, in the form of a directed graph with data items as arc annotations. State charts capture the behavior of a system by specifying the control flow between activities. A state chart is a finite state machine with an initial state and transitions driven by Event-Condition-Action (ECA) rules. Each transition between states is annotated with an ECA rule. A transition from state X to Y, annotated with an ECA rule, fires if event E occurs and condition C holds. The effect is that state X is left, state Y is entered, and action A is executed. Conditions and actions are expressed in terms of variables, for example, those that are specified for the data flow in the corresponding activity chart. In addition, an action A can start an activity, and can generate an event E or set a condition C.

Turning to the WARP (Workflow Automation through Agent-based Reflective Processes) project [Bla02, Bla00], the approach used distinguishes between structural, functional, non-functional, and operational views. The structural views show information about the activities, definition of the roles, and composition of the workflow. They are represented in UML class diagrams. The functional views show the data and control flow of the workflow by using UML activity diagrams. The non-functional concerns (error-handling, concurrency, atomicity, etc.) use data and control flow models and can be modeled with activity diagrams as well. Finally, the operational views are related to the initiation of workflow instances, and the coordination for the completion of the workflow. Operational views can be modeled using UML sequence diagram.

2.2.3 WSM-Nets Formalism

The Well-Structured Marking-Nets (WSM-Nets) approach is used in the ADEPT WfMS [RD98]. As it has been summarized in [RRD03b], WSM-Nets are serial-parallel, attributed graphs on which control and data flow of a process schema can be described. More precisely, different node and edge types are provided for modeling control structures like sequences, branchings, or loops. Branchings and loops are modeled in a block-oriented fashion (block structure). This structure is relaxed by offering *sync edges*, which allow defining precedence relations between activities of parallel branches. Self-explanatory definitions for WSM-Nets and for workflow instances based on WSM-Nets have been given in [RRD03b]. WSM-Nets are somewhat comparable to BPEL4WS (cf. Section 2.2). The latter uses a block structure for defining processes. WSM-Nets provide, how-

ever, a better understanding and formal foundation regarding the use of links (called sync links in WSM-Net).

In the following, we provide the definitions of a WSM-Net and of a workflow instance based on WSM-Net. We will apply these definitions to formally introduce new concepts and functionality in Chapter 6.

Definition 2.1 (Well-Structured Marking-Net, WSM-Net) A tuple $S = (N, D, NT, CtrlE, SyncE, LoopE, DataE)$ is called a WSM-Net if the following holds:

- N is a set of activities and D a set of process data elements
- $NT: N \mapsto \{StartFlow, EndFlow, Activity, AndSplit, AndJoin, XOR-Split, XORJoin, StartLoop, EndLoop\}$
 NT assigns to each node of the WSM-Net a respective node type.
- $CtrlE \subset N \times N$ is a precedence relation
- $SyncE \subset N \times N$ is a precedence relation between activities of parallel executed branches
- $LoopE \subset N \times N$ is a set of loop backward edges
- $DataE \subseteq N \times D \times \{read, write\}$ is a set of read/write data links between activities and data elements

As an example of a process schema modeled by a WSM-Net, Figure 2.3 depicts a simplified medical treatment process. The control and data flow are clearly shown. For example, activities “admit patient”, “inform patient”, and “prepare patient” are arranged in sequence whereas activities “monitor” and “operate” are executed in parallel. “Weight” and “temperature” are examples of data involved in a data flow modeled between activities “prepare patient” and “operate”.

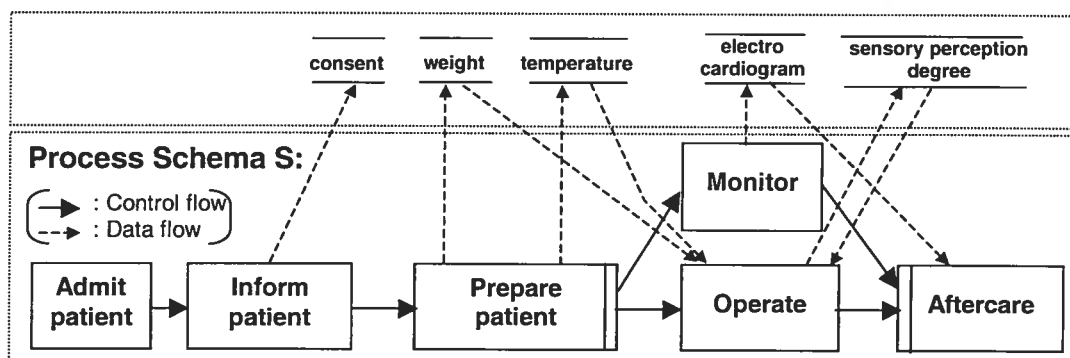


Figure 2.3. Medical Treatment Process

Process instances can be created and executed at run-time. As defined in [RRD04c], a process instance references the process schema it was created on. Furthermore, specific execution states of a process instance are given by model-inherent activity and edge markings. An activity which can be worked on is thus labeled *Activated*. As soon as the activity execution is started the marking changes to *Running*. Finally, a finished activity is marked as *Completed* and an activity, which belongs to a non-selected, alternative execution branch, is marked as *Skipped*. Once an activity is completed, its outgoing edge is set to *TrueSignaled*. When an activity is marked as *Skipped* its outgoing edge is set to *FalseSignaled*, which may lead to the skipping of succeeding activities.

Definition 2.2 (Workflow Instance based on WSM-Net) A workflow instance I is defined by a tuple $(S, M^S, Val^S, \mathcal{H})$ where:

- $S = (N, D, NT, CtrlE, SyncE, \dots)$ denotes the WSM-Net the execution of I is based on.
- $M^S = (NS^S, ES^S)$ describes node and edge markings of I :
 - $NS^S: N \mapsto \{NotActivated, Activated, Running, Completed, Skipped\}$
 - $ES^S: (CtrlE \cup SyncE \cup LoopE) \mapsto \{NotSignaled, TrueSignaled, FalseSignaled\}$
- Val^S is a function on D . It reflects for each data element $d \in D$ either its current value or the value *UNDEFINED* (if d has not been written yet).
- $\mathcal{H} = \langle e_0, \dots, e_k \rangle$ is the execution history of I . e_0, \dots, e_k denote the start and end events of activity executions. For each started activity X the values of data elements read by X and for each completed activity Y the values of data elements written by Y are logged.

2.2.4 Workflow Temporal Aspects

The workflow model should be capable of capturing different aspects of the business process [JB96] including structure, data, and resources properties, but also temporal properties. Time modeling in workflows has been investigated in the context of some (few) workflow research projects. Marjanovic and Orłowska specify that basically three main time constraints can be specified [MO99]: (1) a duration constraint that models the expected duration of an activity in a workflow (a single relative time value or an interval of two relative time values); (2) a deadline constraint that can be specified in terms of absolute time limits when an activity should start or finish during workflow execution; (3) an interdependent temporal constraint that limits when an activity should start/finish relative to the start/finish of another activity (a relative time value). Temporal consis-

tency plays a crucial role in the modeling of time constraints. It must be verified several times during the workflow lifetime: during the workflow modeling and then again during the workflow enactment at several control points (usually after each decision node) to make sure that activities are executing as planned. An algorithm for the verification of temporal consistency in workflows is introduced in [MO99].

Dadam *et al.* introduce temporal aspects by defining time edges between activities [DRK00]. They discuss a minimum and a maximum duration of an activity, and an earliest and a latest relative starting/finishing time of an activity.

Eder *et al.* [EPP+99] worked on a method to enrich a workflow specification by time information for activities, and to translate such a workflow description into a PERT-diagram that shows for each activity the time when the activity must be at a specific state to satisfy the overall time constraints of the workflow. They put the assumption that the end event of an activity corresponds to the start event of all its successor activities. The extension of the PERT-net technique (ePERT [PEL97]) consists in associating relative time information with the end of an activity A. As an example, the earliest point in time A may end corresponds to an execution where optional activities are not executed and the fastest alternative in all selective routings is always selected. A *forward traversal* of the workflow model is required for computing the *earliest* point in time activities may end. A *backward traversal* of the workflow model is required for computing the *latest* possible point in time activities can finish to ensure minimal execution time for the entire workflow.

As argued by Marjanovic in [Mar01], the three approaches introduced above [MO99, EPP+99, DRK00] follow the paradigm of modeling temporal aspects “on top” of a “control-flow” oriented workflow model: to assign temporal attributes to individual activities whose order has been predetermined by control flows. In [Mar01], a two-levels approach for workflow modeling is motivated: a control flow level and an operational level. Thus, a separation is done between the modeling of control flow and the modeling and verification of temporal constraints. At the operational level, an analysis of the accumulated workflow instances stored in a workflow log is made to detect cases where duration of an activity is a function of an instance type (i.e., identification of imprecise activities –

an imprecise activity is an activity with different duration for different instance types). Decision nodes are introduced at the operational level to distinguish these cases. Hence, temporal properties determine modeling of control flows at this level. As a consequence, workflow models at the two different levels are syntactically different but semantically equivalent, and the workflow model at the operational level provides a more *precise* modeling of temporal aspects than the workflow model at the control-flow level. Due to the improved precision in modeling, it is possible to predict more accurately, during workflow execution, *when a specific activity is likely to occur* and to dynamically verify temporal constraints based on the actual execution (i.e., the real duration) of individual activities.

Assigning time to activities in a workflow is a task similar to scheduling in real-time systems. A differentiation is done between time management at build-time and time management at run-time. At build-time, using the workflow model and the durations assigned to the activities in the model, the relative start and end times for all activities are calculated (with respect to the beginning of the workflow). Such calculations are carried out using a forward traversal and a backward traversal of the workflow model. At workflow instantiation time, a calendar is used to convert all relative time information specified during build-time to absolute time points.

If a deadline is missed, a time failure is generated and special actions may be triggered, referred to as escalation actions [EPP+99]: deadline extension, alternative selection, optional removal, and time error.

Despite the importance of time for the coordination and the execution of business processes, the currently available time management support in workflows is rather rudimentary. As pointed out in [Mar01], the requirements for time modeling and visualization far exceed the capabilities provided by today's (commercial) WfMSs.

2.2.5 Organizational Structure

Participants' specification in the workflow usually requires beforehand a specification of an organizational structure (roles, capabilities, positions, hierarchies, etc.). One important aspect of an organizational structure is the division of authorities and responsibili-

ties. An example of authority is to assign work to other members of staff. The most widely used form of organizational structure is the hierarchical organization characterized by a “tree” structure where each node shows either (1) the person who is responsible for all the people below her in the tree, or (2) the department (i.e., organizational unit) that gathers sub-departments defined below it in the tree down to reach individual staff at the leaves (cf. Figure 2.5). A simple example of an organizational meta-model based on the second definition could be the one captured by the entity-relation diagram of Figure 2.4. Based on this meta-model, a tree such as the one shown in Figure 2.5 can be defined.

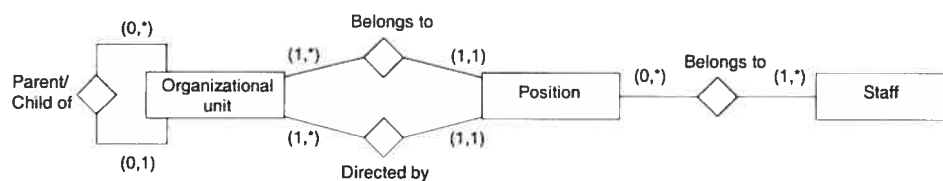


Figure 2.4. Example of an Organizational Meta-model, adapted from [RT02]

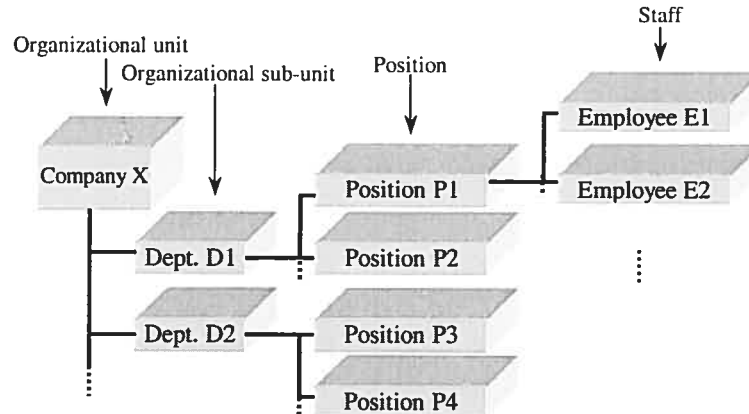


Figure 2.5. Example of an Organizational Model (Tree Structure)

The WfMC defines a simplistic organizational model [WfMC99b]. It specifies that a reference to an external model can also be done. In fact, depending on the internal structure of a company and on the workflow application to be developed, the definition of a company-specific or application-specific organizational model may be necessary. The

complexity of the meta-model on which this definition should be based is a subject of discussion. The more complex the meta-model is, the more detailed the organizational model can be defined, and the more specific the actor assignment to activities can be derived.

2.3 Workflow Enactment

Workflow enactment consists mainly in coordinating the execution of activities according to a predefined workflow model. More precisely, since activities are carried out by workflow participants, workflow enactment actually requires coordination among the participants in executing the activities.

Architectures for the scheduler-based workflow enactment range from a highly centralized to a fully distributed coordination [CHR+98]. In the centralized approach, there is a single workflow engine (or *scheduler*) that controls and coordinates the execution of the activities for all workflow instances. The advantages of the centralized approach include easy monitoring and auditing, simpler synchronization mechanisms, and overall design simplicity [Men02]. However, there are also many shortcomings: a single point of failure, performance limitations, scalability problems, etc. [AM97]. Scalability problems involve workflow engine robustness problems, e.g., a workflow may crash when hundreds or thousands of workflow instances are concurrently running. To solve these problems, the distributed approach is proposed. A summary of some architectures for this alternative approach is given in [SAA99]. A distinction is done between the partially distributed approach, where each workflow has its own scheduler, and the fully distributed approach, where there is no scheduler and the task managers (that could be software agents) coordinate the execution of activities by communicating among them.

In addition to the basic workflow enactment, a “dynamic” workflow enactment is necessary to deal with the dynamic nature of today’s business environments. Adaptive workflows will be introduced in Chapter 3. At this point, we only emphasize the need for a “dynamic” workflow engine to support changes brought to the execution course of workflow instances at run-time.

2.4 Workflow Management Systems

We now study how we can manage processes using information technology. Recently, the availability of tools to help in the definition and control of the various activities associated with a process considerably increased. These tools are known as Workflow Management Systems (WfMSs). Section 2.4.1 thoroughly reviews the Workflow Reference Model (WfRM), an emerging standard from the Workflow Management Coalition (WfMC). Section 2.4.2 addresses current generation of commercial WfMSs.

2.4.1 Standardization Effort

The WfMC has developed a reference model for workflow technology (WfRM) [WfMC95] (Figure 2.6). The major goal of the model is to provide a standard for interoperability among workflow subsystems. It consists of a general description of the structure of a WfMS, in which five main components are presented (Process Definition Tools, Workflow Client Applications, Invoked Applications, Other Workflow Enactment Service(s), Administration and Monitoring Tools).

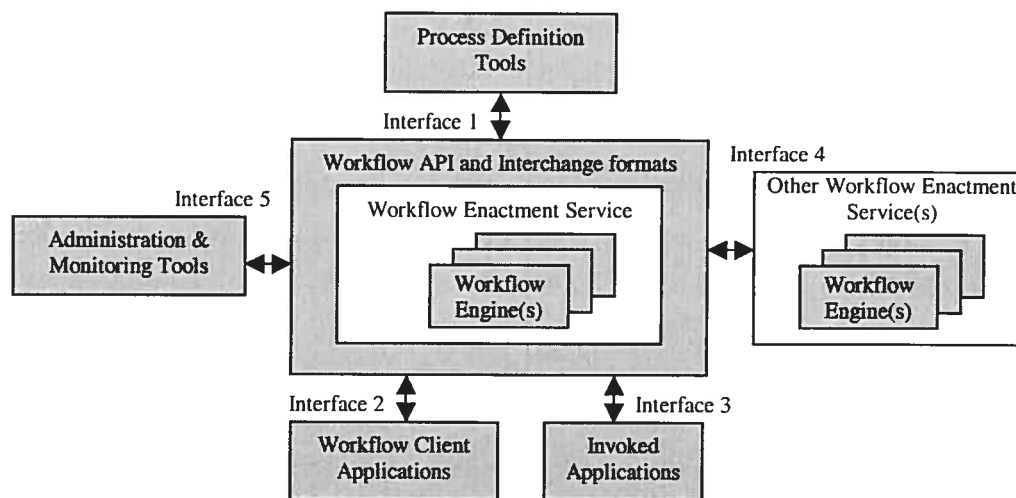


Figure 2.6. WfRM – Components and Interfaces, taken from [WfMC95]

These components are related to the Workflow Enactment Service via interfaces, which are supported by a set of API calls (Workflow Application Programming Interface

WAPI). Many operations are identified across the five interface areas. These operations are gathered within a number of groups represented by the 14 ellipses in Figure 2.7. Based on [AH02, WfMC95], the following sections present details on each component of the WfRM.

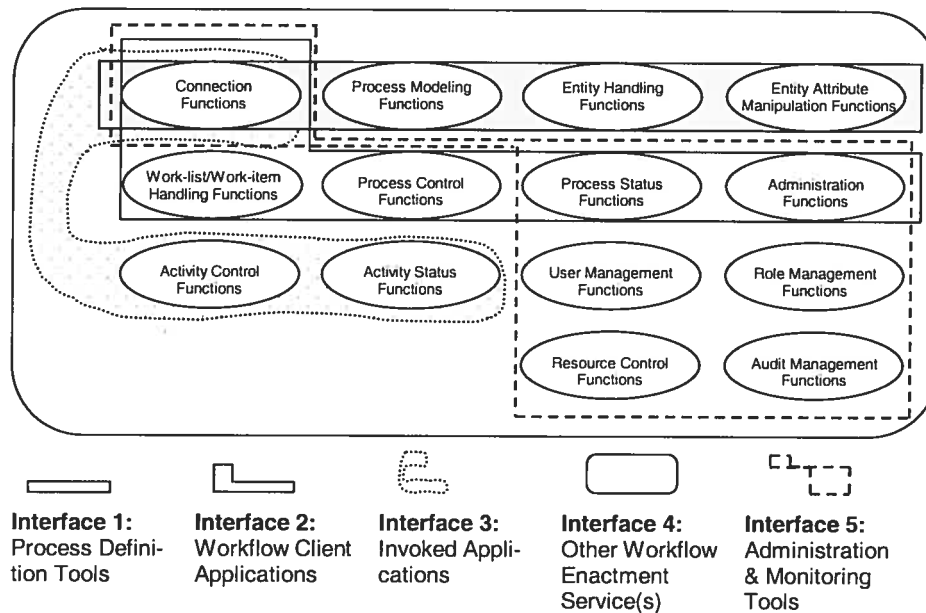


Figure 2.7. Groups of Operations Distributed within the Five Interfaces of the Workflow Reference Model, based on [WfMC95]

2.4.1.1 Workflow Enactment Service

The Workflow Enactment Service is the core of a WfMS. It provides the run-time environment for the execution of workflow instances. It comprises at least one workflow engine. The latter has the functionality for creating, managing, and executing workflow instances. It allows mainly to:

- Sign-on/sign-off specific participants
- Create/delete an instance
- Control an instance (creation, activation, suspension, termination, etc.)
- Route an instance by interpreting the process model definition
- Manage attributes

- Submit work-items to the correct resources based upon the classification of the different resources (cf. Section 2.4.1.2.2)
- Handle triggers
- Call and start up IT applications related to a specific task and link any workflow relevant data
- Record historical data (audit), provide a summary of the workflow, and monitor the consistency of the workflow (these are supervisory actions)

2.4.1.2 Process Definition Tools

In order to accomplish the aspects covered by the engine, process definitions and resource classifications are used. They are preliminarily produced by Process Definition Tools. In addition to illustrating the process and the organization, these tools offer sometimes analysis techniques (e.g., simulation). These techniques are still limited in current WfMSs. In the following sections (2.4.1.2.1, 2.4.1.2.2, and 2.4.1.2.3), we will describe each of the three aspects addressed by the Process Definition Tools.

2.4.1.2.1 Definition of Processes

In a Process Definition Tool, the process model is defined either in a graphical or textual way. Aspects such as the name, the description, the date, the version, and the components of the process are specified. Such a tool allows also to model different types of routing by means of components such as And-split/And-join and Or-split/Or-join. It supports version management for a same process, the definition of attributes used in the process, the specification of tasks, the checking of the (syntactical) correctness of a process definition, and the tracing of any omissions or inconsistencies. Finally, a number of characteristics need to be established for each task:

- The name and description of each task
- The associated user (role/organizational unit) or IT application that should carry out the task (or should be started)
- Supporting information (resp., instructions) for the user performing the task (resp., IT application)
- Task routing characteristics

- Specification of triggers
- Instructions to the workflow engine (e.g., priorities)
- Specification of the related attributes
- Rules that determine how the tasks progress across the workflow and which controls are in place to govern each task

2.4.1.2.2 Classification of Resources

When a process is defined, it is better to couple tasks with resources instead of a specific user. A Resource Classification Tool, considered as part of the Process Definition Tool, allows to find relationships among various resource classes (roles and organizational units). Roles are based upon qualifications, functions, and skills, while organizational units are rather based upon regrouping into teams, branches, and departments. Specific characteristics are affected to a specific resource class. A hierarchy of roles or organizational units may exist (this defines a relationship).

2.4.1.2.3 Analysis

An Analysis Tool can be embedded in the Process Definition Tool. It allows workflow simulation or creates prototype and-or pilot versions of a particular workflow such that this workflow can be tested on a limited basis before it goes into production. Such analysis can encompass checking the semantic correctness of a process definition.

2.4.1.3 Workflow Client Applications

When a defined process is initiated by a workflow engine, the appropriate user and IT applications are scheduled and engaged to complete each activity as the process progresses. The contact the humans have with the workflow is done via the Workflow Client Applications. Work-lists that are part of the Workflow Client Applications, are used by workflow engines to show which work items need to be carried out. Each user has her personal work-list to quickly identify her current tasks along with such things as due date, priority, state, etc. We distinguish between a standard and a customized work-list handler. In a standard work-list handler, the functions provided are generic. They are not customized to any application. By contrast, the customized work-list handler can be adapted to a specific application.

2.4.1.4 Invoked Applications

Performing a task may require the workflow engine to execute one or more external applications. A distinction is done between interactive and fully automatic applications. Interactive applications are initiated when we select a work item from the work-list (e.g., a form that needs to be completed). However, a fully automatic does not require any user intervention (e.g., a program that performs a calculation).

Some applications are workflow-enabled and can be invoked directly by the workflow engine. However, other applications are not compatible with the standardized interface related to the Invoked Applications component. Their integration into the business process is possible only via a software agent that takes the role of an actor and enables indirect interaction of the workflow engine and the application in question. The actor agent is encountered in projects such as the MALL2000 project [HH99], the TSE project (Andersen Consulting) [CS96], and CONSENSUS project [BAV+01].

2.4.1.5 Other Workflow Enactment Services

It is possible to link several autonomous WfMSs with one another. Instances (or part thereof) can be distributed among these WfMSs. This distribution may be based upon the characteristics of the instance, the task, or the resource. Four possible interoperability models are identified in [WfMC95]: connected discrete (chained), hierarchical (nested sub-processes), connected indiscrete (peer-to-peer), and parallel synchronized. Refer to Figure 2.8 for a visual representation of these models.

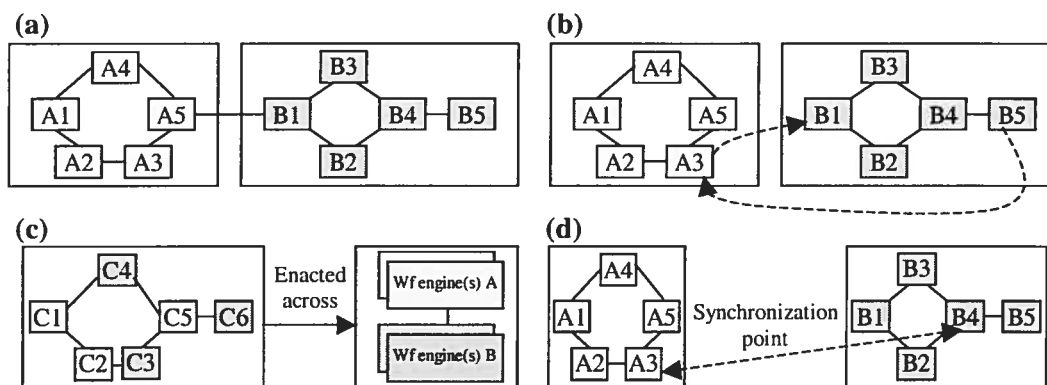


Figure 2.8. Interoperability Models, adapted from [WfMC95]. (a) Chained, (b) Nested Sub-Processes, (c) Peer-to-Peer, (d) Parallel Synchronized

2.4.1.6 Administration and Monitoring Tools

The Administration and Monitoring Tools can be divided into two types of tools, those used for the operational management of the workflows, and those used for recording and reporting.

2.4.1.6.1 Operational Management Tool

Three types of information are managed by the Operational Management Tool: resource-related, system-related, and instance-related information. The tool functions for resource-related information allow the addition or deletion of staff and the input or updating of user details such as the user name, the user role, and the user availability. The functions for system-related information allow the reconfiguration of the workflow system. Finally, the functions for instance-related information permit the inspection and the manipulation of the logistical state of an instance when an exception occurs.

2.4.1.6.2 Recording and Reporting Tool

Some of the WfMSs give the opportunity to measure and analyze the execution of the process so that continuous improvements can be made. A number of aspects can be saved during execution. These are historical data that gather, for instance, information about the execution (e.g., completion/waiting/processing time of an instance) and properties of completed workflows (e.g., bottlenecks, overcapacity). Such aspects may lead to revise the current process (e.g., reallocation of tasks, redefining portion of the workflow model).

2.4.1.7 Discussion of the Workflow Reference Model

It is often argued that workflow technology is still young and not yet fully developed. The WfRM just presented reduces the confusion that may arise as to what is expected from the basic functionality of a WfMS. Indeed, it defines the different components of a WfMS as well as the API that supports the interfaces among these components and the Workflow Enactment Service. However, workflow management has many facets other than the basic ones already supported by the current WfRM. Indeed, in the context of specific complex applications, WfMSs are often expected to support (1) *advanced con-*

cepts such as concepts relative to the temporal aspects of a workflow or to the standard definition of activities, and (2) *enhanced functionality* such as operations for the dynamic change of workflow instances. Unfortunately, it is still unclear which components and-or API calls should be added to the existing WfRM so that such concepts and functionality could be provided.

2.4.2 Current Generation of Commercial WfMSs

A number of WfMSs are available on the market. The number of suppliers offering WfMSs is estimated at two hundred [AH02]. Staffware from TIBCO Software Inc. and Staffware Plc [Tib04] is one of the most widespread WfMSs in the world. Therefore it may serve as a nice illustration of the capabilities of the current generation of commercial WfMSs. A detailed description of Staffware is given in [AH02]. In an initial phase of our work, we experimented with two other mainstream commercial WfMSs: the IBM MQ Series Workflow [Ibm04] and the WLPI (WebLogic Process Integrator) from BEA Systems [Web04]. In the following, we will first briefly describe each of these two WfMSs. Then, we will discuss future prospects of these systems.

2.4.2.1 IBM MQ Series Workflow

IBM MQ Series Workflow (now, IBM WebSphere MQ Workflow) [Ibm04] is a “flow-chart style” WfMS. It consists of the following components (designed as a three-tier structure):

- (1) The *Buildtime GUI* offers a graphical editor to create workflow models. Other features allow one to define the organization (staff, roles, etc.) and the implementations (data structures and programs to use in the workflow), as well as the network definition.
- (2) The *MQ Workflow Client* used to start/stop the execution of workflow instances and to manage work-items within work-lists. Process monitoring functions are also part of this component.

- (3) The *Administration Utility* used to start/stop the MQ Series Workflow system, to list the defined resources specified in build-time. It regularly checks the state of all servers and it can be used to list the current state of any server.
- (4) The *server components* include four types of servers: the Execution Server (process instances management), the Administration Server (server components management), the Scheduling Server (activities control), and the Cleanup Server (finished process instances deletion).

The MQ Series Workflow components can be easily mapped onto the WfRM: the *Build-time GUI* corresponds to the process definition tools (Interface 1), the *MQ Workflow Client* corresponds to the client applications (Interface 2), the *Administration Utility* corresponds to the administration and monitoring tools (Interface 5), and the *server components* provide the workflow enactment service of MQ Series Workflow.

MQ Series Workflow requires a relational database (DB2 or Oracle). Process development information and process run-time information are stored in two separated databases. MQ Series Workflow uses a Workflow Definition Language (FDL) for the exchange of process models between build-time and run-time.

During build-time, activities read data from an input data structure, take some action, and then write data to an output data structure. A mapping tool is provided to map items in the output data structure to the input data structure of the next step. Each activity has an exit condition that is set by the activity program and is saved into the output data structure of the activity. A process will not move on until the activity exit condition is met. When this exit condition is met, MQ Series Workflow evaluates the conditions on all the subsequent control connectors from that activity. It then activates zero, one, or many subsequent activities depending upon whether the control connectors to those activities evaluate to true or not. MQ messages are used to pass control from one activity to the next.

At run-time, MQ Series Workflow uses a Program Execution Agent and a Program Execution Server to invoke application programs in a workflow. Note that the MQ Workflow Client has a poor usability. For instance, it does not provide the ability to re-order

work-lists. The poor usability of the client windows incites users to build a custom client.

An interesting comparison of MQ Series Workflow and Staffware – considered as the leader of the business process management (BPM) market – is given in [Wat01]. In short, Staffware is more suitable than MQ Series Workflow when dealing with simple workflows. Staffware does not require any infrastructure to operate in a basic mode. It has its own integrated form builder and uses its own file format to store workflow definitions and run-time information, while MQ Series Workflow requires a relational database and a programming language for building programs that implement activities. In addition, MQ Series Workflow provides more benefit over Staffware for complex workflows. Workflow definitions and run-time statistics in MQ Series Workflow are already available in a relational database for reporting purpose, while in Staffware, extra work is required to load this information into a database.

2.4.2.2 BEA WebLogic Integration

BEA WebLogic Integration [Web04] allows for connecting applications, databases, enterprise information systems, processes, and business partners; it gathers a set of functionality in the following areas: application integration, BPM, B2B integration, and data integration. Since we are interested in BPM, we will describe the BPM functionality provided by BEA WebLogic Integration. This corresponds to the former WebLogic Process Integrator (WLPI) product. WLPI consists of the following components:

- (1) The *WebLogic Integration Studio* (formerly, WLPI Studio) used to design processes. It provides a graphical interface in which flowchart elements are available for workflow modeling. It is also used to define users and roles as well as to monitor workflow instances.
- (2) The *BEA WebLogic Server* includes a process engine used to manage the execution of business processes.
- (3) The *WebLogic Integration Worklist* (formerly, WLPI Worklist) is the client application used to start/stop processes and to interact with a running process.

It also allows users to handle business process tasks assigned to them (e.g., specify the value of a variable).

WebLogic Integration can be mapped onto the WfRM as follows. The *WebLogic Integration Studio* forms the process definition tools (Interface 1). This Studio as well as the *WebLogic Integration Worklist* forms the workflow client applications (Interface 2). The Studio corresponds to the administration and monitoring tools as well (Interface 5). The process server provides the workflow enactment service of WebLogic Integration. We observe that the BPM functionality of the WebLogic Integration system is mainly encompassed in the Studio. This does not show in a clear manner the separation of the different interfaces defined in the WfRM. IBM MQ Series Workflow provides a better separation as we already saw in the previous section.

WebLogic Integration requires a relational JDBC database (Oracle or SQL Server). Workflow template definitions and running instances of a workflow are respectively saved in a template store and an instance store.

At build-time, public and private business processes can be developed using the Studio (B2B integration environment). Nodes such as Start, Done, Task, Decision, Event, And-join, and Or-join are used to design workflow models. *Actions* are defined within these nodes and they are performed when a node is activated in the workflow. A wide variety of actions are provided (e.g., call an Enterprise JavaBeans (EJB) method, send an XML message to an application).

During execution, XML is used for data representation, and JMS is used for messaging between workflows and other applications. Business processes can be started in a number of ways: called by an application or another workflow (sub-flow), invoked manually (e.g., using WebLogic Integration Worklist), triggered by the reception of an event notification (XML message), or timed to start automatically at a predefined date and time. During run-time, statistics can be collected for reports (evaluation of processes, optimization of performance and throughput).

2.4.2.3 Future Prospects of WfMSs

Van der Aalst and van Hee examined the future prospects for WfMSs in terms of seven areas of functionality [AH02]:

- **Modeling:** WfMSs should acquire more *repository functions* in the future, or improved interfacing with such tools. Repositories should record much more on an organization's data and they should offer good query opportunities through which all the connections relevant to the management of the organization can be analyzed. Another aspect is the *expressive power* of the modeling function. Common constructions in business processes should be well handled. A final aspect is that today's WfMSs are mainly suited to *standard* processes where the number of workflow instances is large compared with the number of workflow models (i.e., production workflows). WfMSs should offer functionality for so-called *one-of-a-kind* processes (ad-hoc workflow), with a separate process defined for each case. WfMSs should integrate process definition functionality with the workflow engine.
- **Analysis:** Simulation and formal verification techniques are used to perform workflow analyses. An expansion of simulation abilities includes to ease the use of historical data from the WfMS for testing modified business processes. Simulation tools can easily evolve into workflow engines because it is not a great leap from simulating workflows to coordinating real ones. An expansion of formal verification techniques – mainly developed for Petri nets – would incorporate correctness tests into the process definition tools.
- **Planning:** Today's WfMSs sometimes offer a limited ability to allocate resources to tasks and to schedule tasks using the same resources. Timetabling problems are not solved by today's WfMSs, though these problems are becoming more and more significant in organizations. Planning support may be offered by the application of modern operations research methods in preparing schedules (e.g., taboo search, constraint satisfaction). What we just introduced is known as operational planning problems. Tactical planning problems should also be considered. As an example, decisions are taken about how much of the capacity of a

particular resource will be required during the period being planned for. Although a WfMS does in fact contain all the relevant information needed to solve such problems, none yet actually offers the facilities to do so. Also, it is still unclear whether producers of these systems should develop such functionality themselves, or whether it would be better for them to try to integrate planning software into their programs.

- **Transaction management:** This requires an appropriate communications process (e.g., XML for supporting e-business transaction processing).
- **Interoperability:** Restrictions regarding the monitoring of protocols and the support of data conversion among communicating applications, should be overcome.
- **Internet/Intranet:** WfMSs should allow the use of a web browser as a Workflow Client Application (Interface 2). On the one hand, this makes it possible for users to access the workflow system through the Internet, and hence to perform work from anywhere. On the other hand, the combination of workflow and the WWW opens up new application opportunities: e-business.
- **Logistical management:** It is provided by Enterprise Resource Planning (ERP) systems. One of the most important functions of these systems is the calculation of the required resources for a specific enterprise project. The scheduling of these resources in time is addressed, and the process is deduced consequently. It is of interest to incorporate such functions from ERP systems into WfMSs.

Van der Aalst and van Hee argue that it is unlikely that workflow product manufacturers incorporate all these functionalities because they would never be able to remain up to date in every one of these fields. A better solution is for the architecture of their systems to be left sufficiently open so that it is easy to integrate other manufacturers' software packages with specific functions from the range described. A great work of standardization is required.

The BEA WebLogic Integration product (whose BPM functionality is described in Section 2.4.2.2) and the IBM WebSphere MQ Workflow (described in Section 2.4.2.1) can be considered as an attempt to integrate functionalities in many areas. These areas do

not, however, completely cover all the ones introduced above. They are mostly oriented towards B2B applications (e.g., “IBM e-business solutions”), and they lack to address in an appropriate manner all the modeling, planning, and analysis issues.

In this thesis, we address mainly aspects related to the modeling area (expressive modeling functions, functionality for one-of-a-kind processes), and to the planning area (integration of a planning software into WfMSs). The two examined commercial WfMSs are not quite appropriate to address advanced needs in today’s complex applications. Indeed, functionality stemming from areas such as “adaptive workflows”, “workflow temporal constraints” and “workflow data management” should be supported. Nevertheless, some researchers in the workflow domain are studying these areas – part of the “modeling” area of functionality in the classification of van der Aalst and van Hee exposed above. ADEPT [RRD03a] is an example of a WfMS prototype that addresses adaptive workflows and workflow temporal issues. In our work we rely on ADEPT as a well-founded basis, to address and to test new workflow concepts and functionality, but also to try to integrate functionalities provided by external tools/systems.

2.5 Summary

In this chapter, workflow management concepts have been reviewed and the terminology that will be used in the rest of the thesis has been introduced. Workflow management involves the design and enactment of workflows. Workflow design consists of creating a workflow model, which is a description of several aspects of a workflow: the activities to be fulfilled, the assignment of activities to participants that are either humans or software systems, the control and data flow between activities. Workflow design requires a set of modeling concepts including temporal issues and organizational structure issues. Modeling concepts may be based on formalisms such as Petri nets and UML, or on formalisms that focus specifically on workflow aspects. In this thesis, we are interested by the latter. We consider the WSM-Nets formalism to introduce new workflow modeling concepts and functionality. Workflow enactment refers to the execution of the activities comprised in a workflow, as prescribed by the corresponding workflow model. The WfRM is a standardization effort for the development of WfMSs. This model does

not accommodate in an appropriate manner concepts and functionality inherent to complex socio-technical systems. These concepts and functionality relate to modeling, analysis as well as planning. Table 2.1 lists the workflow modeling formalisms that were presented in this chapter, as well as the enactment engines presented and their corresponding specification languages. Workflow modeling formalisms concerning adaptive workflows are not listed in the table, since they will be covered in detail in Chapter 3.

Table 2.1. Workflow Modeling Formalisms and Workflow Management Systems

Workflow modeling formalisms-languages	Workflow Management Systems	
	Enactment engines	Specification languages
Petri nets	IBM MQ Series	Workflow Definition Language (FDL)
UML (state and activity charts)	BEA WebLogic Integration	XML-based language (e.g., WSCI)
WSM-Nets		
WPDL		
XML-based formalisms (e.g., XML nets, XRL, XPDL, WSCL, WSCI, BPEL4WS)		

Chapter 3 Adaptive Workflows

The capability to dynamically adapt in-progress workflows is an essential requirement for any workflow management system [RRD04a]. This requirement is mainly motivated by the need to react to external or unexpected events. Furthermore, as pointed out by Horn and Jablonski [HJ98], *adaptive workflows* are interesting in the context of specific applications because it may be impossible to identify all the elements of a workflow model (i.e., workflow or process schema/type) before run-time. Furthermore, modeling all alternative paths in advance might decrease its readability. Domain experts sometimes prefer therefore to model paths that are used frequently only.

A distinction is made between (1) *ad-hoc changes* or *punctual changes* [EK00] which are workflow changes applied to a single workflow instance, and (2) *evolutionary changes* consisting of adapting a collection of workflow instances due to a permanent change of a workflow model [HS98]. The latter includes propagating changes on workflow instances or migrating workflow instances running on an old schema S to the new schema S' . Evolutionary changes are relevant for instance when new laws come into effect, when the maintenance or when the optimization of a workflow model is required. Ad-hoc changes are of interest when exceptional situations occur that influence a single workflow instance.

Adaptive workflows are currently studied by a number of researchers in the workflow community [AM00, CCP+98, EK00, HS98, HJ98, KBB98, RD98, Kra00, Wes01, AB02, SMO00, RRD04b]. Problems and challenges behind this topic are featured within the literature, and solutions based on specific approaches are proposed as well. In this chapter, we first review the challenges and problems related to adaptive workflows as exposed in the literature (Section 3.1). Then, we present and discuss a selection of important research projects and their related approaches (Section 3.2). In Section 3.3, adap-

tive WfMSs are discussed. The final section puts the chapter into perspective by highlighting our interests regarding adaptive workflows.

3.1 Challenges in Adaptive Workflows

While not distinguishing between ad-hoc and evolutionary changes, we focus on three main issues related to adaptive workflows:

The *expressiveness of the workflow meta-model used* [HS98]. The workflow meta-model should provide appropriate modeling constructs to support the dynamic requirements of business processes. There are two different interpretations of this statement:

- The first interpretation consists of a workflow meta-model that is expressive enough to let workflow instances react *automatically* to specific events which corresponds to a process-driven approach for workflow changes. As an example, the Dynamic Workflow Model (DWM) extension of the Workflow Process Definition Language (WPDL) provides dynamic properties needed to support the requirements of inter-organizational business processes [Men02]. Constructs such as “events” (e.g., before-activity event, after-activity event, external event), “rules” and “triggers” are defined within DWM. Another example is given in [NDS96]. A workflow modeling approach using transactions and tasks is described: transactions specify the contents of the workflow, and tasks specify the scheduling and execution of transactions and also provide reactivity to failures. In such frameworks, the workflow already comprises the adaptations required for accommodating pre-defined, potential failures. In fact, automatic adaptations depend usually on the outcomes of previous activity executions, and they restrict in advance the possible workflow changes.
- The second interpretation consists of a workflow meta-model that is expressive enough at the control and data flow level to allow *practically* relevant changes: if loops for instance are not tolerated by a specific workflow meta-model, there would be no way to bring a change by inserting/deleting a cyclic structure. WASA₂ Activity Nets [Wes01], MILANO Nets [AM00], and TRAMs Graphs [Kra00, KG99] provide examples of adaptive workflow models based on meta-

models excluding loops. If data flow issues were excluded from a workflow meta-model, there would be no way to deal with data during workflow changes; e.g., the insertion/deletion of data would not be possible. Petri Net-based adaptive workflow models do not usually explicitly consider data flows. Examples of such models are Workflow Nets [AB02] and MILANO Nets [AM00]. In this case, the *correctness verification* of changes (refer to the third issue below) does not include the verification of data, i.e., whether data is correctly provided or not.

The *completeness of the set of change operations* allowed [RD98, RRD04a, SMO00].

The set of offered change operations should be complete in the sense that starting from a basic workflow model with only a begin node and an end node, any workflow model can be built using this set of change operations. Completeness and minimality are well discussed in [RD98]. It has been argued that for practical purposes, as a minimum, change operations for inserting and deleting activities as well as control/data dependencies among them are needed [RRD04a]. In [SMO00], the authors also discuss change operations for modifying activity properties (data requirements, underlying application, temporal constraints, resource allocation) and for modifying the order of activity execution. Sometimes, an adaptive workflow meta-model (e.g., MILANO Nets [AM00]) allows for structural changes such as *parallelization* to change sequential activities into parallel activities, *sequentialization* to change parallel activities into sequential activities, and *swapping* to change the order of activities. However, it fails to allow for fundamental changes such as the insertion of a new activity and the deletion of an existing one. A complete yet *minimal* set of change operations is desired [RD98].

The *correctness verification* regarding the application of changes on instances. Correctness criteria for verifying if a workflow instance is compliant with the proposed changes are required. It must be ensured that those changes will not cause inconsistencies or errors for the rest of the workflow instance processing. Rinderle *et al.* [RRD04a] point out that correctness criteria should not be too restrictive, i.e., no workflow change should be needlessly refused. As surveyed in [RRD04a], such criteria are addressed by the following researchers: Agostini and De Michelis (MILANO [AM00]), Casati *et al.* (WIDE [CCP+98]), Ellis *et al.* (ML-DEWS [EK00, EKR95]), Reichert and Dadam

(ADEPT [RD98]), Kradolfer and Geppert (TRAMs [Kra00, KG99]), Weske (WASA₂ [Wes01]), van der Aalst and Basten (Woflan [AB02]), Sadiq *et al.* (BREEZE [SMO00]). To this list, the work of Rinderle *et al.* is added [RRD03b, RRD04c, RRD04b].

The second and third issues discussed above have been identified in [RRD04a] as fundamental issues. Moreover, another interesting issue has been added in [RRD04a], namely change realization. From a *workflow evolution* perspective, it should be possible to *automatically* migrate workflow instances to a new schema. One challenge is to correctly and efficiently adapt instance states [RRD04a]. This challenge holds in the context of ad-hoc changes as well.

It must be mentioned that the four issues cited above are discussed in the literature almost exclusively in the context of *structural* workflow changes. Regarding other kind of changes such as workflow attribute changes, changes in the workflow temporal aspect, and organizational model changes, they are still not studied and discussed in an appropriate manner. Attribute changing operations are evoked in [RRD04c]. It consists of changing the *value* of an activity attribute or of an edge attribute. In [SMO00], it is argued that “time” is an element that makes workflows dynamic, and that temporal uncertainty during workflow modeling calls for verifying the consistency of temporal constraints during execution, at the workflow instance level. Applying changes on an organizational model and correctly propagating those changes on workflow instances have not been addressed till now in the literature.

3.2 Projects Addressing Adaptive Workflows

Even though the need for adaptive workflows is apparent, solutions are not obvious. In this section, we review relevant research projects in relation with adaptive workflows. Change policies and modalities are first reviewed and discussed (Section 3.2.1). Then, in Section 3.2.2, key projects proposing solutions to the challenges identified in Section 3.1 are surveyed and discussed.

3.2.1 Workflow Change: Policies and Modalities

We review in this section two main research projects that classify various elements related to workflow change. We begin by presenting modification policies [Sad99] (Section 3.2.1.1); then we expose a set of factors to be taken into account when specifying a change [EK00] (Section 3.2.1.2).

3.2.1.1 Modification Policies

In the context of evolutionary changes, Casati *et al.* present a set of workflow changes referred to as Case Evolution Policies [CCP+98]. They identify “abort”, “flush” and a set of progressive policies that allow instance or case dependent evolution management of a workflow. Sadiq [Sad99] identifies a larger number of workflow modification policies, which can be adopted by the workflow administrator:

- Flush: All current instances are allowed to complete according to the original process model, but new instances are set to follow the new model.
- Abort: Active workflow instances may be aborted when the process model is changed, and then restarted (or not) according to the new model.
- Migrate: The change affects all current instances but it has to be introduced without allowing current instances to abort or flush. This policy calls for the “correctness verification” issue discussed in Section 3.1.
- Adapt: The process model may not change permanently, but some instances have to be treated differently because of some exceptional and unexpected circumstances (i.e., ad-hoc changes).
- Build: The starting point is not a detailed pre-existing model, but an elementary description, which captures only the basics. For instance, workflow activities are identified, but the order of execution is mostly unknown. In [KBB98], the authors use the terms “partial” or “just-in-time” execution.

These policies cover evolutionary changes (flush, abort, migrate), and ad-hoc changes (adapt). In [HS98], Han and Sheth specify that a strong association can exist between ad-hoc and evolutionary changes: if ad-hoc changes are to be made permanent, we are confronted with a problem of workflow evolution. Moreover, we may add that if a workflow

evolution is to be applied on running instances, we are confronted with a problem of ad-hoc changes.

From these policies, two different facets are also identified with respect to adaptive workflows: the dynamic *change* and the dynamic *definition* of workflows. We observe that the “build” policy tackles exclusively the dynamic definition of workflows. Hence, adaptive workflow technology takes a broader perspective than the one that is restricted to “dynamic workflow changes”.

When compared to the modification policies presented above, the requirements discussed in [KBB98] address *advanced* elements regarding adaptive workflows. The authors describe for instance “reflexivity” where a process has the ability to re-model itself, and the “late binding of resources” where the completion of activities uses the resources at hand at a specific point in time.

3.2.1.2 Change Modalities

When we want to specify a change, there are many factors that must be taken into account. Ellis and Keddara [EK00] present eight important change modalities to be specified so that an unambiguous change will be implemented:

- Change duration: Instantaneous versus time interval versus indefinite.
- Change lifetime: This specifies the amount of time that the change is in effect. It could be permanent or temporary.
- Change medium: Manual versus automatic versus mixture. Usually, when the number of instances that must change is small, this could be manually done *by a human*. If the number of instances is big, then they should be *automatically* updated. This has been discussed in [RRD04a] under the “change realization” issue.
- Change time-frame: Past versus present versus future. The instances to which a change is applicable are typically restricted to the ones in progress. However, there are situations where instances that have not yet begun are excluded, or instances that have already terminated are included.

- **Change continuity:** Preemptive versus integrative. Here we specify the planning and the implementation work of the changes, e.g., should we disrupt (preempt) currently running instances or not? Preemptive strategies include abort, rollback, restart, checkpoint, and flush schemata. Integrative strategies include versioning, and other gradual instance migration schemata.
- **Change agents:** Here we specify which participants play which organizational roles within the change process, e.g., who has the right to specify, enact, and authorize what types of changes.
- **Change rules:** There are participatory rules that define the participation aspect of a change process, integrity rules that define the various constraints of a change, (e.g., temporal, data integrity, and flow constraints), and situated rules that specify how to react in the face of exceptional situations, e.g., constraint violation and system failure.
- **Change migration:** This refers to the ability to bring filtered-in instances into compliance with the new schema in accordance with the migration policies.

Note that the “change lifetime” specification refers to both ad-hoc and evolutionary changes. The “change continuity” is obviously defined by policies such as the ones presented in 3.2.1.1, and the “change migration” overlaps with the “migrate” policy.

Regarding the “change medium” modality that introduces the automatic change aspect, we think that the specified context (i.e., big number of instances) within which automatic changes are interesting is too restrictive. Automating workflow changes is indeed desirable in many other contexts. The type of application studied may for instance require *automatic* workflow changes (e.g., with a rule-based approach). This may be realized on top of adaptive WfMSs (cf. Section 3.3).

As a final note concerning change modalities, “change duration”, “change time-frame”, and “change agents” are not well studied in the literature. Researchers are rather interested in “change rules”, e.g., the “correctness verification”, as discussed in Section 3.1.

3.2.2 Proposed Solutions for Adaptive Workflows

A survey and an interesting classification of key projects in the adaptive workflow domain [AM00, CCP+98, EK00, RD98, Kra00, Wes01, AB02, SMO00], and specifically of the approaches adopted and the correctness criteria developed within these projects, are provided in [RRD04a]. In this section, we review these key projects with respect to the three adaptive workflow issues discussed in Section 3.1.

We begin by describing each of these projects (Section 3.2.2.1). Then, in Section 3.2.2.2, the expressiveness of the workflow meta-model used within each project is studied. Afterwards, in Section 3.2.2.3, the completeness of the set of changes provided by these projects is reviewed. A summary of the different projects' correctness verification of changes is given in Section 3.2.2.4. Finally, Section 3.2.2.5 puts these three adaptive workflow issues into perspective, by discussing five typical problems regarding dynamic workflow change.

3.2.2.1 Description of Key Projects

Woflan [AB02]. In this project, the Workflow Nets, a Petri Nets-based model, is introduced. Transformation rules based on inheritance concepts (cf. Section 3.2.2.4) are developed to avoid problems such as the “dynamic-change bug”. The latter refers to errors such as duplication of work, skipping of tasks, and deadlocks introduced when migrating an instance from an old schema to a new one or when ad-hoc changes are applied on an instance. A tool that supports the inheritance notions has been developed (Woflan – WorkFlow ANalyzer [VA04]). It can analyze workflows designed with various workflow products. This tool has been successfully tested with Staffware.

WASA₂ [Wes01]. The workflow model Activity Nets using an object-oriented activity-based workflow meta-model, is defined. It comprises one generic class *Workflow* of which *Workflow schema* and *Workflow instance* are instances. Within a workflow schema, activity nodes, control connectors, and data connectors are defined. Data connectors map output and input parameters of subsequent activities (data flow). The workflow model used is comparable to Activity Nets applied in IBM MQ Series Workflow. A

WfMS architecture based on the CORBA object-oriented middleware has been elaborated. The system, including dynamic adaptations, has been implemented.

MILANO [AM00]. Two different representations of a workflow model are possible: (1) The Workflow Net Model (WfNM) – a local state representation making explicit the independence between the actions, and (2) the Workflow Sequential Model (WSM) – a global state representation where the path followed during the execution of an instance is made immediately visible. MILANO provides a specification module that supports the users when changing a workflow model. The correctness of changes is verified to ensure a *safe enactment* of these changes on instances. This enactment is postponed in instances that are in an unsafe state until they reach a safe one. The theoretical framework of Milano allows three patterns of change: parallelization, sequentialization, and swapping. A *Minimal Critical Specification* (MCS) is defined, and must be satisfied by the workflow model and its changes. It is used as a reference to guide changes.

ICN and ML-DEWS [EK00]. An approach to provide dynamic changes by replacing a given sub-workflow by another completely specified sub-workflow is introduced in [EKR95]. Notions of dynamic change and correctness as allowed by the Petri Net formalism – the Information Control Net model (ICN) – are defined. ICN is used to analyze structural changes. In [EK00], the eight change modalities presented in Section 3.2.1.2 are defined, but also a workflow Modeling Language (ML-DEWS) that supports the Dynamic Evolution within Workflow Systems is elaborated. ML-DEWS is an extension of UML. It provides two meta-models: one to specify a workflow, and the other to specify a change within this workflow. The language is based on concepts as proposed by the WfMC. All workflow model elements (workflows, activities, rules, events, and flow) are modeled as classes. Pre-defined change schemata are supported by ML-DEWS. Within these schemata, the ad-hoc schema supports ad-hoc changes. The idea is to complete the change specification at run-time when the change process is enacted. The authors discuss change design and enactment that either alternate or are done in parallel.

WIDE [CCP+98]. One of the first approaches dealing with dynamic workflow changes was offered by the WIDE project. It provides a generic correctness criterion for process schema change propagation (the so called *compliance criterion*). This criterion is suit-

able when *instance execution histories* are logged. An instance *I* created from a schema *S* is *compliant* with a changed schema *S'* if the execution history of *I* can be correctly replayed on *S'*. In WIDE, workflow schemata can be described either graphically or by using predecessor and successor functions. In brief, as stated in [RRD04a], WIDE has offered a cornerstone for many other approaches – the intuitive history-based compliance criterion.

TRAMs [KG99]. Within this project, the workflow meta-model developed provides support for the versioning of process schemata and explicitly defines correctness criteria for the model as well as for the workflow instances. A taxonomy of modification operations has been developed. These operations address changes at the process schema level (add/drop a process schema), at the version level (add/drop a version, changes to a version state), and at the version content level (attribute and activity changes). The migration of workflow instances is studied. Another aspect addressed in [Kra00] is the reuse of process schemata, which is a process consisting of finding, understanding, adapting, and integrating process schemata instead of developing process schemata from scratch.

ADEPT [DR98]. The WSM-Nets workflow model developed within the ADEPT project has already been introduced in Chapter 2. The research efforts in this project were initially concentrated on the support of *ad-hoc deviations* at the workflow instance level without violating data consistency, temporal constraints, and robustness of the system [RD98]. Data dependencies and the data flow between steps are analyzed to decide which dynamic modifications can be granted and which have to be refused. Lately, evolutionary changes have been addressed in a significant manner within the ADEPT project [RRD03b, RRD04c, RRD04b]. When compared to other projects that address workflow evolution, the ADEPT approach is considered as a farsighted approach since it studies and proposes correctness criteria for propagating process type changes not only to instances that are still running according to their original schema (i.e., unbiased instances), but also to those instances that have been individually modified (i.e., biased instances).

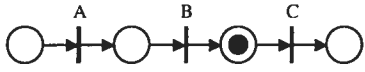
BREEZE [SMO00]. A three-phase modification methodology that consists of (1) defining the modification, (2) verifying the compliance of the workflow instance with the

proposed modification, and (3) realizing the modification is proposed in [SO99a, Sad99]. This methodology handles the modification policies presented in Section 3.2.1.1. A WfMS architecture providing fully automated support for the process of dynamic changes has been elaborated. It allows the automatic compliance verification of the instance to migrate. A “compliance module” component generates graphs (called “compliance graphs”) that define a bridge between a workflow model version k , and a workflow model version $k+1$.

3.2.2.2 Workflow Meta-Model Expressiveness

Assume a workflow composed of a sequence of three activities A, B, and C. We apply workflow models respectively developed within the different projects (P) introduced in Section 3.2.2.1 to represent this workflow (S_P). Table 3.1 shows workflow instances I_P respectively issued from S_P . In each instance the activities A and B were completed. Information behind the execution phase is included (e.g., execution history, markings). Some notes regarding the expressiveness of the workflow meta-model used are given. We consider that a workflow model allows the modeling of sequential, parallel, conditional, and iterative activity branches, and that data flow is supported. Otherwise, i.e., if it is not the case for a specific workflow model, we state it clearly in what follows.

Table 3.1. Adaptive Workflows Key Projects – Workflow Meta-Model Expressiveness

Workflow instance taking into account a specific workflow modeling language	Execution phase information	Meta-model expressiveness
<p><i>Woflan: Workflow Nets</i></p> 	Petri nets rules apply.	Data flow issues are excluded.

WASA₂: Activity Nets



○ : NotActivated
 ◐ : Completed

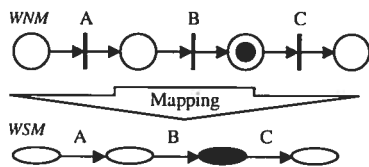
-----> : Data flow

Possible markings: NotActivated, Activated, Running, Completed, Skipped.

Loops are excluded. The acyclic graph structure is even a correctness criterion for a workflow schema – no deadlocks.

Activity output parameter and its corresponding activity input parameter should be type-conform. This is one of the correctness criteria for a workflow schema.

MILANO Nets



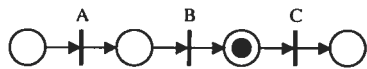
Petri net rules apply in WNM.

Loops are excluded – acyclic Free-Choice Petri Nets are used.

● : Current state in WSM

Data flow is not explicitly considered.

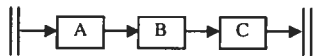
ML-DEWS: Flow Nets



Colored tokens are used to distinguish different instances.

Comparable to Workflow Nets, but places can hold more than one token.

WIDE Graphs



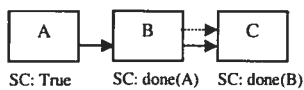
Only activity completion events are logged within the *History* (E = End).

||| : Start/End

$Vars^S = \{d_1\}$
 History = ((E_{start}), (E_A), (E_B, <d₁, "value">))

Workflow instance state can be deduced from the history logs.

TRAMs Graphs



SC: Start Condition

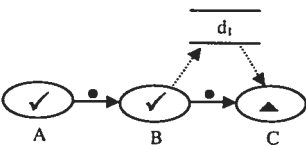
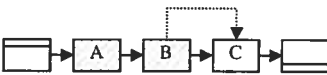
-----> : Data flow

Possible markings: Activated, Running, Completed.

$Vars^S = \{d_1\}$
 History = ((S_A), (E_A), (S_B), (E_B, <d₁, "value">))

Activity start and completion events are logged within the *History* (S = Start, E = End).

Workflow instance state can be deduced from the history logs.

<p>ADEPT: WSM-Nets</p>  <p>History=((S_A), (E_A), (S_B), (E_B, <d₁, "value">))</p>	<ul style="list-style-type: none"> ✓ : Completed ▲ : Activated • : True-Signaled <p>Possible markings: NotActivated, Activated, Running, Completed, Skipped.</p> <p>Activity start and completion events are logged within a reduced <i>History</i> (S = Start, E = End).</p>	<p>→ : Data flow</p> <p>Synchronization edges between activities in different parallel branches are included.</p>
<p>BREEZE</p>  <p>Vars^S={ d₁ }</p> <p>History=((S_A), (E_A), (S_B), (E_B, <d₁, "value">))</p>	<ul style="list-style-type: none"> ▨ : Completed <p>Possible markings: Scheduled, Active, Suspended, Completed, Terminated.</p>	<ul style="list-style-type: none"> □ : Initial □ : Final → : Data flow

3.2.2.3 Set of Changes Completeness

In [RRD04a], it has been specified that the set of changes defined within each of the projects introduced in Section 3.2.2.1 is complete except in the Woflan project where the order changing operations are explicitly excluded, and in the MILANO project where only the parallelization, the sequentialization, and the swapping of activities are allowed. Indeed, most of the projects allow for the serial and parallel insertion and deletion of an activity, and the change of an activity attribute value (variable, in-/out-parameter). In addition, ADEPT allows moving an activity, inserting and deleting a synchronization edge, and changing an edge attribute value. However, the insertion and deletion of a workflow data element is evoked but not discussed in detail. In TRAMs, the start and end conditions of an activity can be changed, and changes on data issues are addressed: insertion and deletion of an in-/out-parameter, of a workflow variable declaration, and of a data flow. In WIDE, the insertion and deletion of variables is possible as well.

3.2.2.4 Summary of Correctness Verification

A summary of the change correctness criteria elaborated within each of the key projects is given in Table 3.2. A detailed survey of these criteria is given in [RRD04a].

Table 3.2. Adaptive Workflows Key Projects – Correctness Verification of Changes

Approach	Correctness verification
<p><i>Woflan: Workflow Nets</i> (Ad-hoc and evolutionary changes)</p>	<p>A workflow instance I on schema S is compliant with the modified schema S', if S and S' are related to each other under inheritance.</p> <p>There are two kinds of basic inheritance relations. S is a subclass of S' if the behaviors of S and S' cannot be distinguished when:</p> <ul style="list-style-type: none"> • <i>only</i> executing tasks of S, which are also present in S'. I.e., blocking a sub-set of tasks of S. • <i>arbitrary</i> (all) tasks of S are executed but only effects of tasks that are present in S' as well are taken into account. I.e., hiding a sub-set of tasks of S. <p>(S and S' could be reversed.)</p> <p>There exist automatic transfer rules for adapting markings.</p>
<p><i>WASA₂: Activity Nets</i> (Ad-hoc and evolutionary changes)</p>	<p>I is compliant with S' (i.e., I can be migrated to S') if a valid mapping exists between I and S', i.e., if I is a prefix of S'. In this case, all completed activities of I and all control and data dependencies in I are also contained in S'.</p> <p>I is a <i>purged instance graph</i>: I is derived from S by deleting all activities which have not been started yet and by removing all associated control and data edges.</p>
<p><i>MILANO Nets</i> (Evolutionary changes only)</p>	<p>I is compliant with S' if I is in a safe state on S regarding S'. A safe state on S regarding S' is a state that is present in S' as well.</p> <p>If I is in an unsafe state, the migration of I is postponed until a safe state is reached.</p>
<p><i>ML-DEWS: Flow Nets</i> (Ad-hoc and evolutionary changes)</p>	<p>Suppose that m is the marking of I on S. It is supposed that the marking m' of I migrated to S' is known.</p> <p>I is compliant with S' if for each of the possible firing sequences leading from m' to the final marking of S':</p> <ul style="list-style-type: none"> • This sequence is producible on S starting from m. What will potentially be done on S' could be done on S as well. <p>OR</p> <ul style="list-style-type: none"> • The firing sequence that led to m on S can be reproduced on S' and hence it can be continued on S' by this sequence. What has been done on S can be reproduced on S'. <p>Marking adaptations are always correctly performed since the old change region is completely contained in the new net.</p>

<p>WIDE Graphs (<i>Evolutionary changes only</i>)</p>	<p>I is compliant with S' if the execution history on S can be "replayed" on S' as well. All events stored in the execution history could also have been logged by an instance on S' in the same order. <i>Obvious approach problems:</i> (1) Possibly extensive volume of history data, which is normally not kept in main memory [KG99]. (2) Restrictions in conjunction with change operations within cyclic structures [RRD04a].</p>
<p>TRAMs Graphs (<i>Evolutionary changes only</i>)</p>	<p>Similar to WIDE Graphs. In addition, migration conditions are verified. This provides a solution for problem (1) exposed in WIDE Graphs.</p>
<p>ADEPT: WSM-Nets (<i>Ad-hoc and evolutionary changes</i>)</p>	<p>Similar to WIDE Graphs. However, a <i>reduced</i> execution history is used, and migration conditions are verified. This provides a solution respectively for problem (2) and problem (1) exposed in WIDE Graphs.</p>
<p>BREEZE (<i>Evolutionary changes only</i>)</p>	<p>Similar to WIDE Graphs. A compliance graph is generated. It defines a bridge between S and S'. A compliance graph <i>i</i> related to an instance <i>i</i>, allows the latter to follow a unique path defining actions or compensations necessary to achieve compliance for this instance, and a suitable "plug" point in S'.</p>

In the context of the Woflan project, van der Aalst and Basten used the concept of bi-simulation [Mil80, Par81] in order to verify inheritance of process schemata [AB02]. Specifically, branching bi-similarity was used as an equivalence relation on Petri Nets schemata: two Petri Nets schemata S and S' are bi-similar if S can simulate every behavior of S' and vice versa, i.e., starting from the initial marking every firing sequence of S must be executable on S' and vice versa. Bi-simulation is a well-founded concept for correctness verification that can also be applicable on process specification formalisms other than Petri Nets. Other techniques coming from process algebra [Hen88] were successfully used for the verification of systems [GR01]. In fact, process algebra can be used as a specification formalism for workflows since they yield elements for the modeling of sequential, alternative and parallel processes but also a rule framework for the verification of these processes. Unfortunately, there is no implementation of WfMSs based on process algebras. A possible explanation could be the lack of understandability (i.e., user friendliness) of the formalism. Featuring more details regarding bi-simulation and process algebra goes beyond the scope of this work.

3.2.2.5 Discussion

Five typical problems regarding dynamic workflow change have been reviewed in [RRD04a]: changing the past, loop tolerance, dangling states, order changing, and parallel insertion. They denote correctness criteria problems. It has been argued that: (1) the past of an instance should not be changed; (2) it should not be needlessly impossible to bring changes within loops; (3) it is sometimes problematic not to distinguish between activated and running activities; (4) a potential problem of order changing is to correctly adapt instance markings; and (5) inserting a new parallel branch is problematic in Petri Net-based approaches (e.g., Workflow Nets, Flow Nets) – new tokens may have to be added to avoid deadlocks or livelocks.

The approaches adopted by specific projects for the correctness verification of changes either deal or do not deal with each of these five problems. Furthermore, in the case where they deal with a problem, they may not do it correctly. This mainly depends on the *expressiveness of the workflow meta-model* used, on the *completeness of the set of change operations* allowed for, as well as on the *approach* adopted itself:

Changing the past. A possible problem of changing the past is to miss input data of subsequent activity execution. As highlighted in [RRD04a], it is interesting to see that the Flow Nets approach forbids changes that affect both already passed regions and regions which will be entered in the sequel. This can be easily explained when considering the two exclusive criteria of the Flow Nets correctness verification of changes. Indeed, when a change is applied on a schema S leading to a schema S' , in the best case, either the statement “what will potentially be done on S' could be done on S as well” is verified or the statement “what has been done on S can be reproduced on S' ” is verified, but never both. Suppose both statements are verified, this means that $S = S'$, which is not the case. Obviously, the verification ensured by the Flow Nets approach guarantees correct data provision.

Loop tolerance. It has been argued in [RRD04a] that a *reduced* (i.e., consolidated) view of the execution history as in ADEPT/WSM-Nets relieves the restrictive aspect in conjunction with change operations within cyclic structures. A reduced execution history is derived from an execution history by discarding all the history entries related to other

loop iterations than the last completed or currently running loop. As shown in the example of Figure 3.1, the execution history on S cannot be replayed on S' , but the reduced execution history on S can be replayed on S' . The *reduced pre-change firing sequence* which is a Petri Net-based approach adopted in Flow Nets and that corresponds to the sequence of transitions that have been fired before the change arrives, and the *purged instance graph* approach adopted in $WASA_2$ are both comparable to the reduced execution history approach in the sense that they relieve the restrictive aspect in conjunction with change operations within cyclic structures. Nevertheless, this problem is factored out in $WASA_2$ since the workflow meta-model used excludes loops.

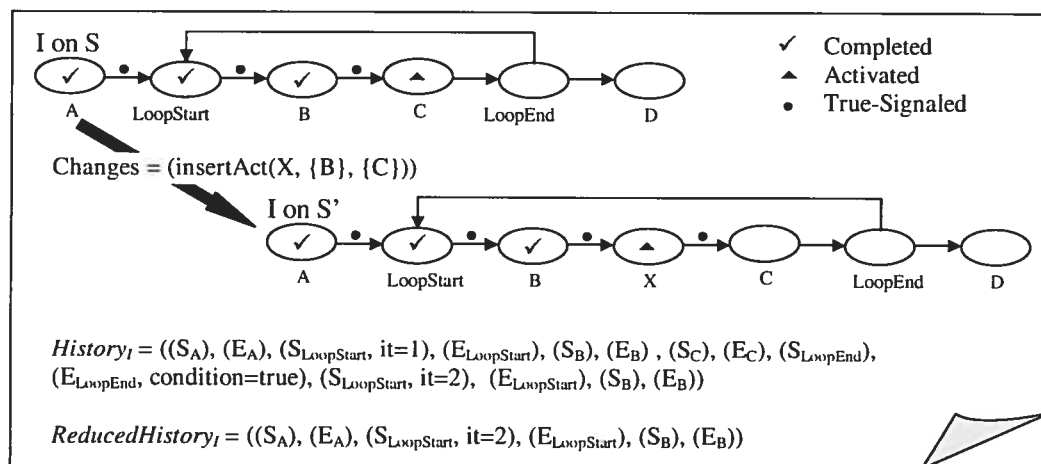


Figure 3.1. Loop Tolerance in ADEPT/WSM-Nets, adapted from [RRD04a]

Dangling states. We observe that the approach of execution histories that contain only “end” entries of activities – such as in WIDE Graphs – does not help to distinguish between an activated state and a running state for a specific activity. The activity state (i.e., the markings) should be specified explicitly to cope with this problem, which however is not done in WIDE Graphs. Workflow modeling languages based on Petri nets (WF Nets, MILANO Nets, and Flow Nets) abstract as well from internal activity states, i.e., they only differentiate between activated and non-activated transitions [RRD04a]. As it has been shown in [RRD04a], this coarse differentiation of activity states is unfavorable in conjunction with the deletion change operation. As an example, the deletion of activated

activities is forbidden which is too restrictive, or the deletion of running activities is allowed which leads sometimes to loss of work.

Order changing. Flow jumpers used in the Flow Nets approach help to correctly adapt instance markings when changing the order of two activities (Figure 3.2). Markings adaptation is considered as a challenging problem in workflow changes, and very little approaches address this issue. Furthermore, in the case where an approach addresses this issue, it may not do it correctly.

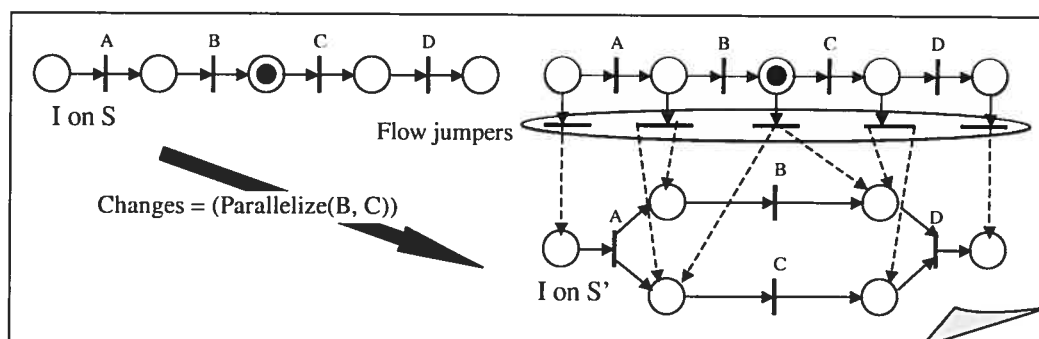


Figure 3.2. Markings Adaptation using the SCOC – Syntactic Cut Over Change – in ML-DEWS/Flow Nets, adapted from [RRD04a]

Parallel insertion. As opposed to Petri Net-based approaches, when the correctness verification is based on “compliance criteria” (e.g., WIDE Graphs, TRAMs Graphs, WSM-Nets, BREEZE), the parallel insertion – if not mixed with the “changing the past” problem – can obviously be easily solved.

As a final note in this section, the correctness verification in the context of “evolutionary changes” or compliance verification may lead to non-compliant instances where instances cannot be migrated to the new workflow schema. Current adaptive workflows projects deal with these instances either by proposing compensation activities so that rolling back non-compliant instances into a compliant state becomes possible [SMO00, Sad00], or by delaying the migration until a compliant state has been reached [EKR95].

3.3 Adaptive Workflow Management Systems

Most current WfMSs support process versioning where multiple versions of a workflow may be active at the same time. Few commercial products provide, however, support for adaptive workflows. WfMSs like Staffware, WebSphere MQ Workflow (reviewed in Section 2.4.2.1), and WLPI (reviewed in Section 2.4.2.2) tend to be very inflexible. Mainly, the adaptation of in-progress instances is not allowed. By contrast, InConcert [Inc02] and FileNet Ensemble [Ens98] allow workflow instance adaptation during runtime, namely the dynamic insertion and deletion of an activity. InConcert [Inc02] for example supports ad-hoc workflows by using *Process Design by Discovery*, a method which allows customers to deploy workflows without a preliminary design phase: the process is built by doing the tasks, may be changed on the fly by users, and saved as a template when completed. As argued in [RRD04a], though ad-hoc WfMSs provide flexibility, they have failed to adequately support end users. Particularly, they do not support them in defining changes and in dealing with potential side effects such as missing input data of an activity due to the deletion of a preceding data provider activity. Since one cannot expect from the end user to cope with such problems, this increases the number of errors and therefore limits the practical usability of respective WfMSs.

Turning now to academic WfMSs, prototypes exist for some of the projects introduced and discussed in the previous sections. We are aware of the following WfMS implementations: WASA₂, BREEZE, Chautauqua [EM97] which offers an implementation where Flow Nets are generalized to Information Control Networks (ICN), and ADEPT which offers an implementation based on WSM-Nets. As for other projects, the basic mechanisms of the framework have been implemented and simulated (e.g., MILANO, TRAMS, Woflan). However, no complete WfMS prototype has been developed.

Automatic workflow changes may be realized on top of adaptive WfMSs. Examples include the dynamic workflow system discussed by Müller and Rahm [MR99]. This system has been tailored on top of ADEPT. It implements an automatic rule-based approach for the detection of semantic exceptions (e.g., drug-side effects) in cancer therapy workflow scenarios, and for the dynamic changes of patient treatment workflow instances. Exception events are filtered out of “normal” events, then affected workflow instances

and control flow areas are automatically identified, and finally affected areas are automatically adjusted. Control flow modification algorithms are provided for this purpose. They allow to drop, to replace, to check, to delay, and to insert a node. Such a rule-based approach for automatic workflow changes is usually based on the availability of knowledge bases, as it is for example the case for medical domains. In fact, a lot of declarative knowledge is needed to derive modification implications when events occur.

3.4 Conclusion

We consider the problems and solutions reviewed in this chapter as groundwork, first, for the categorization of limitations in adaptive workflow technology, and second, for the formalization of related solutions. The expressiveness of a workflow meta-model and the completeness of a set of change operations are debatable issues. Indeed, it has already been highlighted in Section 3.1 that the expressiveness is measured taking into account the *practically* relevant changes allowed by the workflow meta-model. To the extent that “practicality” is involved in the measurement of expressiveness, a “new” workflow concept may appear of great practicality when studying a specific application. Moreover, the completeness of a set of change operations can be judged mostly from the application for which it is going to be used. The proposed and studied change operations are still almost only limited to workflow *structural* changes. The completeness of a set of change operations should be measured beyond structural changes only.

Indeed, issues such as “time” and “workflow attributes” are relevant as well. Beyond the structural specification, the temporal aspects are relevant since they add another dimension to the scheduling of workflow activities [SO99b]. If temporal constraints exist, they should be specified as a complement to the control flow, and they should be addressed in conjunction to changes. Workflow attributes are relevant as well since the successful execution of an activity may require the availability of specific attributes [SO99b]. A data dependency may exist between activities (i.e., data flow). The workflow attributes and the structural aspects of a workflow model are dependent on each other: a condition needs the attributes provided by a data flow to select the alternative path, and a control flow path must also exist to satisfy a data flow constraint between two activities

[SO99b]. Mainly, because of this dependency, structural changes may require changes at the workflow attribute level and vice versa.

Once limitations of current adaptive workflow technology are identified, it becomes necessary to propose solutions to cope with these limitations. Under this perspective, the correctness verification issue should be considered. Proposed solutions need to be based as much as possible on formal definitions, and correctness criteria need to be developed.

Chapter 4 Workflow Technology Applied to Complex Socio-Technical Systems

The domains of e-negotiations and transportation are examples that call for non-trivial socio-technical systems. These systems need to be “dynamic” mainly because of the underlying application environment; they need to be “reactive” in the sense that they should be able to automatically react to internal and-or external events; they involve multiple actors which implies their “social” aspect; they sometimes require the management of shared resources; and finally, we define a “human in the loop” which means that the user must take decisions and she should be able to intervene (manually) with the system to communicate results, to bring modifications to what already exists, and so on. Studying complex systems will serve us to investigate refined aspects of workflow technology and to trigger more adequate support for building such systems.

This chapter begins by briefly reviewing state-of-the-art workflow-oriented applications and by motivating the need to study complex socio-technical applications. Then, in Section 4.2, an e-negotiation application is reviewed in detail and a combined negotiation support system (CONSENSUS) is presented. In Section 4.3, another complex socio-technical application, the multi-transfer container transportation (MTCT) application, is detailed and a workflow-oriented system for the processing of customer requests for container transportation is motivated.

4.1 Workflow-Oriented Applications

For a number of years, workflow technology has been embedded within IT products, instead of being a standalone technology. Hollingsworth discusses the emergence of workflow within the market by analyzing some of the application domains where this technology has been successfully applied [Hol97]: image processing, document man-

agement, electronic mail and directories, groupware applications, and project support software.

Today's literature gathers a number of workflow applications. Some of these applications must be considered as sample examples, while others come from the real world where workflow systems are sometimes specifically tailored to cope with them. The most cited applications are related to domains such as e-business (electronic business), medicine, banking, insurance, public administration, and software development. A number of other workflow case studies can be found at the WARIA website [WARIA04]. These case studies are gathered under different topics: academic, financial, government, healthcare, industry, technology, transportation, and utilities.

In the following, we first review four application domains that are well supported by workflow technology. We focus on the e-business domain, the medical domain, the banking and insurance domain, and the public administration domain. Then, we motivate our involvement in complex socio-technical applications.

4.1.1 E-Business Domain

In e-business, purchasing of goods (e.g., computers, books), buying/selling of stocks, e-procurement of materials (e.g., raw materials) and outsourcing (i.e., supply chains of services) are examples of applications that can be supported by workflow technology [MWW98]. These applications involve inter-organizational workflows [Aal00] because they require communication between multiple parties. They are used as typical distributed examples in [Bla00, SRK+01, AK00, WI98]. Express mail services over the Internet is another e-business application where a company such as Federal Express can offer a workflow-based service to notify the customer (both sender and receiver) as soon as a package is delivered at a site [KZ02]. We may also think of a tracking service that tracks the routing of the package.

Recent research projects in the e-business domain are oriented towards e-services applications (e.g., Web services). We discuss below two recent projects in the context of e-services.

Blake worked on a workflow architecture, called WARP (Workflow Automation through agent-based Reflective Processes), that supports Web services [Bla02]. This architecture consists of software agents that can be configured to control the workflow operation of distributed services.

Meng worked on a dynamic inter-organizational WfMS [Men02]. The sharable tasks performed by people or automated systems in a virtual enterprise are treated as e-services, and e-services requests are specified in the activity definitions of a process model. Four dynamic properties of the proposed WfMS are discussed: the flexible property (the dynamic binding of e-services to service providers), the active property (a result of the integration of business events and business rules with business processes), the customizable property (the processing of business rules may enforce customized business constraints and policies), and the adaptive property (the processing of business rules may dynamically alter the process model at run-time).

The two workflow-based projects described above allow for the dynamic binding of e-services during run-time. This can be considered as a specific type of dynamism in workflows. Meng discusses also dynamic aspects stemming from the enactment of business processes, and she proposes a rule-based approach to deal with such dynamism.

Finally, the BPEL4WS language introduced in Section 2.2, has exceptions built into the language via the “throw” and “catch” constructs. It also supports the notion of compensation, and introduces the notion of scope (comparable to spheres [Ley95]). There is some work on exception handling in BPEL4WS [CKL+03]. However, dynamic change issues have not been addressed yet.

4.1.2 Medical Domain

In the medical domain, most applications are implemented using adaptive workflows. Dadam and Reichert discuss, under the ADEPT project, the management of a hospital’s “day clinic” division by means of workflows [DR98]. They analyze relevant processes, and evaluate to which degree these processes could be supported by current workflow technology. Workflow evolution, exception handling, flexibility, and temporal aspects are among the many important aspects addressed within the project. These aspects are

mainly motivated by medical cases: acute emergency, violation of prerequisites for medical intervention, incomplete medical orders, etc.

A dynamic workflow system tailored for an oncology application is discussed by Müller and Rahm [MR99]. Cancer therapy is characterized by a long-term treatment based on standardized plans that can be modeled using workflows. However, one major problem of cancer treatment is the enormous amount of diagnosis, which may create, for instance, specific drug-side effects to a significant number of patients. The treatment workflows of these patients will have to be modified partially. We already explained this application previously (cf. Chapter 3, Section 3.3).

At another level, the management of food services and dietetics departments of a hospital can be done using workflow technology [NDS96]. Ngu *et al.* describe a workflow modeling approach using transactions and tasks to manage the meal production and the distribution of meals to patients in a number of hospitals (CBORD system). Transactions specify the contents of the workflow, and tasks specify the scheduling and execution of transactions, and they provide reactivity to failures. However, these failures need to be known in advance so that appropriate adaptations are defined. For example, in case the production unit (i.e., “prepare meal” task) of the CBORD system cannot satisfy the required number of meals due to unforeseen disruption in the kitchen, a contingency task “acquire meal” can be invoked to order the shortfall from the external kitchen.

In spite of the fact that the ADEPT project was initiated to address clinical processes, the result of this project was an adaptive WfMS that could be applied to many types of applications. The dynamic workflow system addressing the oncology application uses ideas relative to dynamic workflows that are behind the ADEPT system. In addition, a rule-based approach is elaborated for the detection of semantic exceptions and for applying the changes to workflow instances. This approach is based on the availability of knowledge bases. Finally, the CBORD system puts the emphasis on the fact that when we talk about adaptive workflow management, we refer not only to a workflow system that is flexible enough to support adaptation, but also to modeling languages that provide the appropriate constructs to support dynamism. In this framework, the workflow already comprises the required adaptation for pre-defined possible failures. The dynamism

provided by this framework as well as the one supported by the rule-based approach are similar in the sense that they introduce the notion of “automatic” dynamism (reactivity).

4.1.3 Banking and Insurance Domain

The banking domain and the insurance domain provide classical examples of workflow applications. Examples include credit processing applications and insurance claims processing applications [Man99]. The literature related to the Mentor project [MWW+98] uses the credit processing application as a sample example. The work of van der Aalst *et al.* [ABE+00] considers the insurance claims processing application, as well as applications related to software development, to the hiring of new employees, and to the organization of a conference, as typical examples to expose their new workflow control approach based on *proclats* (used to let instances interact among themselves via channels).

4.1.4 Public Administration Domain

A public administration represents a huge enterprise dealing with different tasks. Well-structured models are usually defined for processes like vacation or travel requests. However, complex and long-running workflows depending on events exist as well. Some parts of these workflows cannot be planned in advance and have to be modified during execution.

An example of such workflows is given by Siebert in [Sie96]. It consists of the planning phase for the construction of a new building triggered by the relocation of some authorities from one city to another. The need for inter-organizational and for adaptive (ad-hoc changes) workflows is highlighted. A “workflow modification services” component is provided to check and perform modifications. Adaptivity rights (i.e., which adaptations may be performed by which actors), and structural integrity/consistency rules are considered by this component to decide whether a requested modification is allowed or not. An original idea concerning the “on-the-fly” editing has been discussed by Siebert [Sie96]. New activities need to be defined and modeled at run-time.

Van der Aalst and van Hee discuss in detail another public administration application based on workflows: the custom service application. They describe the workflow-based

Sagitta 2000 declaration-processing system of the Dutch Customs Service [AH02]. One of the topics addressed is the requirement to separate management and application. This creates an opportunity to improve the control of business processes, it eases the ability to adapt business processes to changes in the law, and it gives a guarantee that formal steps do take place in accordance with the law.

4.1.5 Why Studying Complex Socio-Technical Applications?

The approach of focusing on specific applications has the potential of being a constructive method for building a list of (new) aspects that need to be addressed by workflow technology so that these applications can be properly supported. Some applications, such as the ones cited in previous sections, are considered as workflow-oriented by their nature. Even if studying these applications can help identify some interesting workflow aspects, these aspects may still lack some innovation because the applications introducing them are initially meant to be supported by workflows. However, when studying applications that do not readily lend themselves to workflow-oriented applications, the requirements inherent to workflow-oriented systems supporting them can be profitably used to enhance what workflow technology is currently offering. In this perspective, we study complex socio-technical applications asking for dynamic and reactive systems where multiple actors are involved and where the user should be able to interact with the system.

4.2 The Combined Negotiation Application

Combined negotiations (CNs) are a novel and general type of e-negotiation, in which the user is interested in a package of items (goods and services) and consequently engages in separate negotiations for these items [BAV+01]. The negotiations are independent of each other, whereas the goods or services are typically interdependent. As examples of CNs, two packages are described respectively in [BBK01, BBK+02a]: a flight connection package and an importing package. The flight connection package may consist of three items: a plane ticket from place₁ to place₂, another plane ticket from place₂ to place₃, and a hotel room for one night in place₂. The importing package involves a number of activities/services such as the purchase, the shipment, the insurance and the for-

warding of goods. The items within each package are interrelated since for instance, in the first package, the hotel room should be reserved at place₂ on the date of the trip, and in the second package, the shipment of goods should be scheduled after the goods are delivered. Furthermore, many constraints exist such as the maximum price to pay, the preferable date to travel or to receive goods and so on.

A CN Support System (CNSS) based on a WfMS is proposed in [BAV+01] to help the user (consumer or business) model the CN by specifying the sequencing of the individual negotiations and the dependencies between them. The CNSS is then used to enact the CN and to allow the user to control, track and monitor the progress of the CN as well as the individual negotiations.

In the following, we begin by describing the CNs application. Then, we discuss in detail the modeling of the two packages introduced above. The CNSS, called CONSENSUS, is finally presented.

4.2.1 Description of the Application

Negotiation takes place when an agreement between the consumer and the provider has potential for optimization, and the parties intending to carry out the transaction are willing to discuss (i.e., negotiate) their offers [Str99]. Auctions are a special case of negotiations as they represent a more general approach to price determination, admitting a range of protocols, including fixed-price as a special case [WWW98]. The benefit of dynamically negotiating a price for a product instead of fixing it is that it relieves the merchant from needing to determine the value of the good a priori. Rather, this burden is pushed to the marketplace [MGM99]. On the Internet, negotiation often amounts to one party (typically the seller) presenting a take-it-or-leave-it offer (e.g., a sale price), but in the last few years auctions have become more and more popular especially in the C2C market.

In the context of negotiations, the consumer may be interested in many items that form a whole (i.e., a package of items). The negotiation of these items cannot be conducted separately. Indeed, if conducted separately, it can happen that the consumer finalizes a deal on one item but cannot get a good deal (or any deal at all) on another item. Break-

ing the commitments⁴ already made is not always permitted, and even if it is, it usually costs money and leaves the consumer at the point where she started (i.e., with no package). CNs were mainly thought to cope with this problem, that is, to allow the consumer negotiate the package of items with the minimum risk of breaking her commitments and with the maximum chances of getting good deals on all the items.

Moreover, if more than one attribute of an item is negotiable (e.g., price, date), the dependencies among the individual negotiations get more complicated. The outcome of one negotiation is crucial in the other ones. Therefore, CN facilitates dealing with multi-attribute negotiations, possibly of different types.

In [BAV+01, BBK01], important issues related to CNs are addressed:

- **CN Failure.** We talk of CN failure (i.e., exposure) when we need two items A and B, engage in negotiations on both items, and end up winning on A but losing on B. In a single negotiation, there is no guarantee that we will succeed in winning the item. In a CN the risk of failure is even higher because the user negotiates several items⁵. Since the consumer wants the whole package or nothing, she might want to break (if allowed) the commitments she already made in the successful negotiations. Breaking a commitment evidently has a price. Obviously, the possibility of breaking commitments adds more complexity and requires more flexibility to the CN problem.
- **AND-Negotiation and OR-Negotiation.** The package that is the object of a CN is in general made up of many items. If many negotiations are launched for the same item, these negotiations are called OR-Negotiations. Negotiations for different items that make up the package are called AND-Negotiations. Each of these types of negotiation may be run either in parallel or sequentially. However, it is interesting to run OR-Negotiations in parallel in order to save time and also to maximize the chance of a good deal, whereas it is sometimes an obligation to

⁴ Commitment means that one agent (human or software agent) binds itself to a potential contract while waiting for the other agent to either accept or reject its offer [SL95]. If the other party accepts, both parties are bound to the commitment. When accepting, the second party is sure that the contract will be made, but the first party has committed before it is sure.

⁵ *What can go wrong will go wrong*, as Murphy's law goes – if there are more things that can fail, more things will fail.

run AND-Negotiations sequentially since the output of one negotiation may be crucial as input for another one.

- **Intrinsic and Procedural CN Constraints.** Let us consider three different negotiations (N1, N2 and N3) associated with three different items in a CN. Suppose that the attributes for each item are the price, the date and the place. Intrinsic constraints concern the attributes and the dependencies between the items at the attribute level. They may involve just one individual negotiation (e.g., $\text{price1} \leq \text{THRESHOLD}$, date1 in RANGE, $\text{place1} = X$), or more than one individual negotiation (e.g., $\text{price to pay for the package} \leq \text{THRESHOLD}$, $\text{date3} = f(\text{date1}, \text{date2})$). Procedural constraints indicate the control flow between the individual negotiations:
 - Sequential: N2 is launched after N1 is finished.
 - Parallel: N1 and N2 are launched at the same time.
 - Choice: depending on a condition, either N1 is launched or N2 is launched.
 - Wait for: N3 waits for N1 and N2 to finish or waits for either one to finish.
 - Repeat: repeat N1 until a condition is met.

In order to negotiate the different items in a CN, software agents are assigned to individual negotiations. These agents may be instances of a generic negotiation server infrastructure such as GENESIS [BKL+00] that supports a variety of negotiation types. The behavior of software agents is defined via negotiation rules (i.e., protocols), negotiation strategies and coordination strategies:

- **Negotiation rules** [BAV+01] need to be downloaded from the negotiation server so that a specific agent, responsible of the negotiation of a specific item, is correctly instantiated taking into account the type of the negotiation. Some well-known negotiation types are the fixed-price sale, the Dutch auction, the English auction, the bilateral bargaining, and the combinatorial auction.
- **Negotiation strategies** [BAV+01] are used by the agents when generating offers and counteroffers during the course of a negotiation. A differentiation is done

between negotiation strategies applied to one individual negotiation (e.g., “if your bid is always beaten by the same opponent then be less aggressive in your bidding”) and negotiation strategies applied to the CN as a whole (e.g., “if you have little chance of making a deal on an item then don’t commit yourself on the other items of the package”).

- **Coordination strategies** [BAL+02] correspond to the information that the agents need in order to coordinate their actions in a specific CN. Here are two examples: (1) When two agents are participating in separate negotiations with the goal of purchasing just one item, the following rule ensures that the agents make no more than one commitment at the same time: *“If Agent2 is leading or in the process of bidding, then Agent1 should wait.”* (2) When two agents are participating in separate negotiations with the goal of purchasing two complementary items, the following rule minimizes the risk of exposure: *“If Agent2 is trailing, and its chances of making a deal are slim, then Agent1 should wait for further instructions.”*

Clearly, a CN is a *complex process* since it asks for a specific structure of the different negotiations (procedural constraints), it requires the definition of the dependencies between the items at the attribute level (intrinsic constraints), and it involves many agents, each one conducting an individual negotiation on a distant server while cooperating with other agents in solving a common problem: “the consumer wants the whole package or nothing at the best possible price.”

As stated in [BBK01], modeling a CN using workflows gives a visual representation, which is easily understandable by humans, and identifies and formalizes as activities all the necessary items of the CN. This may be helpful in a prospective evolution or modification of the current negotiation items, their sequencing and the dependencies between them. A CN workflow also incites to reason about the variables and the attributes of a CN (such as the prices, dates, etc.). It may for instance specify some forecasting (e.g., “what will be the new reserve-price based on the outcome of the negotiations that are already done”). It facilitates to deal with software agents responsible of the different ne-

gotiations since these agents are assigned to negotiation-activities, and they participate in a CN as actors in the workflow.

4.2.2 Example of Combined Negotiation Packages

Combined negotiations can be used at the B2B, B2C or C2C levels. As an example, a B2C transaction would be when a consumer negotiates a vacation package consisting of a transportation ticket, a hotel room and an excursion ticket. In case one or more items in the package are offered by consumers (e.g., a rare ticket to a concert auctioned on an auction site), a C2C transaction is encountered; and when a travel agency negotiates travel packages on behalf of its clients, we refer to B2B e-commerce. The more items there are to be negotiated and the more providers of such items there are, the more interesting a CN is.

In the following, we describe first a “flight connection” package which can be considered either a B2C example when the trip is arranged by the consumer herself, or a B2B example when the trip is arranged by a travel agency (the common way to arrange trips), and second an “importing” package, which is mainly a B2B example.

4.2.2.1 “Flight Connection” Package

The “flight connection” package may consist of three items: a plane ticket from place₁ (e.g., Montreal) to place₂ (e.g., Paris), another ticket from place₂ to place₃ (e.g., Moscow), and a hotel room for one night in place₂. These three items are clearly interrelated. One obvious constraint is to find a “Paris–Moscow” flight with a suitable departure time; that is, taking into account the arrival time of the “Montreal–Paris” flight. From here we can see the obligation of spending a night in Paris before taking the flight to Moscow. Many other constraints exist such as the date of the trip, the total amount to be spent, the maximum price the consumer is willing to pay for each item, and her preferences for certain air companies.

The three items (two plane tickets and a hotel room) may be negotiated, and this can be done on different negotiation servers (or on the same server, but in separate negotiations). The negotiations practiced on each single server (i.e., each individual negotiation) can be of different types (“type” in this context means “the rules of the negotiation”).

Modeling the package as a CN is profitable since when conducting each negotiation separately, it can happen, for instance, that a deal is made on the “Montreal–Paris” ticket, while an interesting deal on a “Paris–Moscow” ticket is missing out just because the flight “Montreal–Paris” arrives to Paris a few hours later.

Modeling the Flight Connection Package Example

When modeling the Flight Connection CN, the consumer (or business) has to decide, first, how many negotiations she should start for each item. Engaging in more than one negotiation for the same item (i.e., OR-Negotiation) is a way to minimize the risks of failing to make a deal on the item in question.

Suppose that the consumer decides to participate in two separate negotiations for the “Montreal-Paris” ticket and in two separate negotiations for the “Paris-Moscow” ticket (there are many providers of air transportation tickets on the Web, and there is usually a great disparity between the prices) and one single negotiation for the hotel room (the same thing could be said about this item too, but in this example, the consumer may decide to run only one negotiation). The five negotiations that make up the CN are to be conducted separately, and possibly obey to different rules for making bids (offers), for picking a winner, for closing, etc. The consumer chooses to participate, say, in an English auction for the first “Montreal-Paris” ticket on the Air France auction site, and in a Dutch auction for the second “Montreal-Paris” ticket on the Air Canada auction site. One “Paris-Moscow” ticket is to be negotiated in a sealed-bid multi-round auction on the Air France auction site, and the other “Paris-Moscow” ticket in an English auction on the Aeroflot auction site. The hotel room will be negotiated in a bargaining type negotiation on one of the popular commercial auction sites. For a complete description of auction types, refer to [Sur01].

The sequencing in time of the five negotiations is important. Which negotiations should be conducted in parallel, which ones should be conducted in sequence, which ones should be finished (with a successful or unsuccessful deal) before we start the others? Does the consumer need all negotiations for the flight tickets to succeed or does she need just one to succeed? What to do in case one negotiation fails? In this example, the

consumer might decide that the two negotiations for the “Montreal-Paris” ticket will be launched first, and only if one of them succeeds, the other negotiations will be launched. This may be because the consumer knows that the chances of making a good deal on this particular item are rather slim. Note that only one deal should be made on the “Montreal-Paris” ticket even though the two negotiations are launched at the same time (a case of parallel OR-negotiation). In case one “Montreal-Paris” negotiation succeeds, the consumer launches two parallel negotiations for the “Paris-Moscow” ticket. Let us suppose that one “Paris-Moscow” flight with Aeroflot is on the same day as the arrival of the “Montreal-Paris” flight. The other flight, with Air France, is scheduled for the next day, and the consumer would have to spend a night in Paris. To that end, a negotiation for a hotel room in Paris is launched. The negotiations for the “Paris-Moscow” ticket with Air France and the one for the hotel room are started sequentially (a case of sequential AND-negotiation).

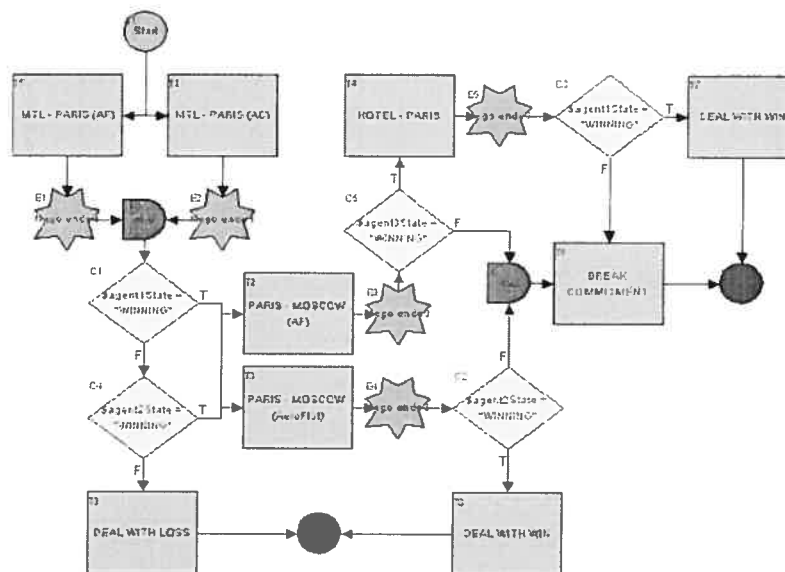


Figure 4.1. Flight Connection Package Workflow Model in WLPI

Figure 4.1 shows a workflow modeling the “flight connection” CN. WLPI were used. There are five main tasks or activities (one for each individual negotiation) represented by rectangles. The ones for the “Montreal-Paris” ticket are launched first, and if one of them succeeds (State = “WINNING”) the two negotiations for the “Paris-

Moscow” ticket, are launched. The negotiation for the hotel room is launched only if the “Paris-Moscow” (AF) ticket negotiation succeeds. There is one “start” state and two “done” states representing the process completion. The star-shaped elements in the figure represent events. The events are to be sent by the tasks to the workflow processor so that the processing continues with the next activity. Events are triggered by XML messages, and in this example, the events are “Nego ended” (i.e., negotiation ended). The diamond-shaped figures are the decisions. They contain conditions that must be evaluated before the succeeding node can be initiated. The conditions evaluate to TRUE or FALSE, and depending on the outcome of the evaluation, the workflow can follow different paths. There are also two And-joins in the workflow. All nodes linked by an And-join must be satisfied before the successor of the join can be activated.

The five activities (i.e., the five negotiations) are assigned to software agents. The tasks labeled “BREAK COMMITMENT”, “DEAL WITH WIN”, and “DEAL WITH LOSS” are assigned to a human agent (the person running the CN).

4.2.2.2 “Importing” Package

Importing goods is a complex procedure in which a buying company is involved in a number of activities/services such as the purchase, the shipment (the term “transportation” is sometimes used thereafter), the insurance, and the forwarding of goods. These activities/services are obviously interrelated. As an example, a special kind of insurance could be preferred while a specific packaging of goods is considered. Many constraints exist as well. Here are some of the constraints that are likely to be involved in the purchase activity: the maximum price the buying company is willing to pay for the goods, the quantity needed, the terms of payment, the delivery date, the packaging of the goods. With regard to the shipping service, which may include inter-modal transportation, a number of scenarios are possible. The supplier can cover the freight shipment and insurance from warehouse of origin to warehouse of destination. Another alternative is to let the buyer cover all charges. In this latter scenario, a constraint might be for instance to find a truck with a suitable arrival (resp. departure) time to port of shipment (resp. from port of destination), taking into account the vessel loading (resp. unloading) time. The buying company could have preferences for specific shipping companies, and may also

specify the maximum amount to be spent for each shipment phase, as well as the total amount for the whole shipping. As for the insurance, the buying company could also have some restrictions regarding the insurance companies, the kind of insurance, the price to pay, etc.

An importing procedure is considered as a sourcing application where multi-stage negotiations such as RFP (Request For Proposal) and RFQ (Request For Quotation) can be applied. Indeed, the buying company may choose to engage in different negotiations for the complementary (i.e., cannot have one without the other) items discussed above (purchase of goods, shipment, insurance, etc.), trying to make the best deal with respect to its interests. We can imagine a CN model as described previously to encompass the activities associated with the negotiation of the different items.

Modeling the Importing Package Example

Figure 4.2 shows a workflow model example for the importing package created using WLPI Studio. Negotiations are defined as activities in the workflow. Software agents are responsible of executing these activities. As the workflow progresses, negotiation-activities evolve through various states: created (creation of the agent), activated (the agent joins the negotiation), executed (the agent negotiates), and marked done (the agent leaves the negotiation and the agent is destroyed).

In this example, the buying company has to take a decision regarding the number of negotiations that should be launched for the purchase of the goods. These tasks could be initiated at the same time (in parallel), but only one deal should be struck. The next step will be to start negotiations for the shipment services. The buying company might choose to begin by negotiating the sea shipment, and then the two surface shipments (from warehouse of origin to port of shipment, and from port of destination to warehouse of destination) because surface transportation is usually more flexible and available than sea transportation. It will hence be easier to schedule the truck arrival (resp. departure) time to port of shipment (resp. from port of destination) with respect to the vessel loading (resp. unloading) time (than to do it in the opposite way). The insurance

and the forwarding negotiations are planned in sequence as the last two items of the model. (For the sequencing, cf. Figure 4.2.)

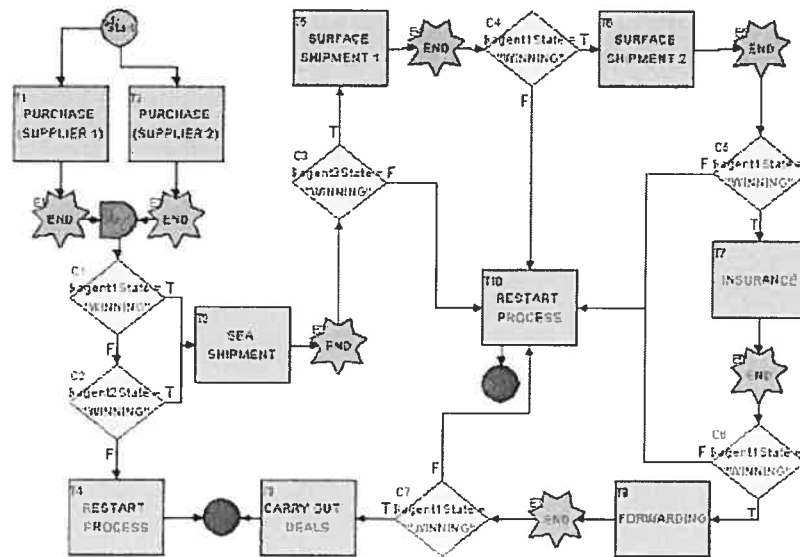


Figure 4.2. Importing Package Workflow Model in WLPI

As stated before, when we fail to make a deal on an item, after concluding deals on other complementary items, we talk of “exposure”. To avoid exposure, the buying company would have to restart the whole process (“Restart Process” task in Figure 4.2) by renegotiating some (or all) of the attributes of the deals already made. For instance, if the buying company fails to find suitable transportation for a given date (fixed in a previous deal), then it could go back and re-discuss the delivery date with the supplier of the goods. In the worst case, this procedure could lead the buying company to breaking its commitments.

4.2.3 The CONSENSUS System

CONSENSUS is based mainly on the following components of a WfMS: the Process Definition Tool used to model the workflow, the Workflow Engine which executes and tracks an instance of the workflow, the Administration and Monitoring Tool used to administer and track the status of the instance, and the Workflow Client Application through which the participants interact with the instances.

The first prototype of CONSENSUS was built on IBM MQ Series Workflow. Then to validate the claim that in CONSENSUS the underlying WfMS may be easily substituted for another WfMS, a new version of CONSENSUS was built on BEA Systems WLPI (cf. Figure 4.3). This version was made up of three units: (1) the WLPI Studio Unit which is used to build the CN workflow and to monitor its execution; (2) the Enactment Unit which is used to launch the CN workflow and to monitor the software agents; and (3) the Coordination Unit which is used to coordinate the work of the software agents. The usage of the system is summarized hereafter, in Sections 4.2.3.1, 4.2.3.2 and 4.2.3.3. More details are given in [BAL+02, BAV+01, BBK01]. Please note that a complete end user documentation comprising UML diagrams (e.g., use case descriptions, sequence charts, etc.) has not been written yet.

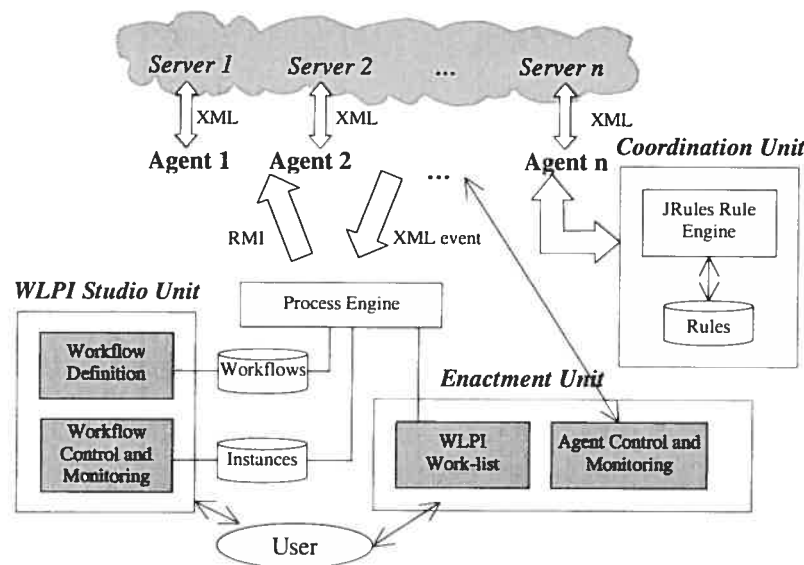


Figure 4.3. CONSENSUS based on BEA Systems WLPI, adapted from [BBK01]

4.2.3.1 WLPI Studio Unit

Figure 4.1 and Figure 4.2 show examples of CN workflows created using the graphical tool of WLPI Studio. One important aspect of modeling a CN is the use of variables that store the CN-specific information required by the workflow at run-time. This informa-

tion is often used to control the logic within the CN. Figure 4.4(a) shows the list of variables of the workflow modeling the “Flight Connection” CN example.

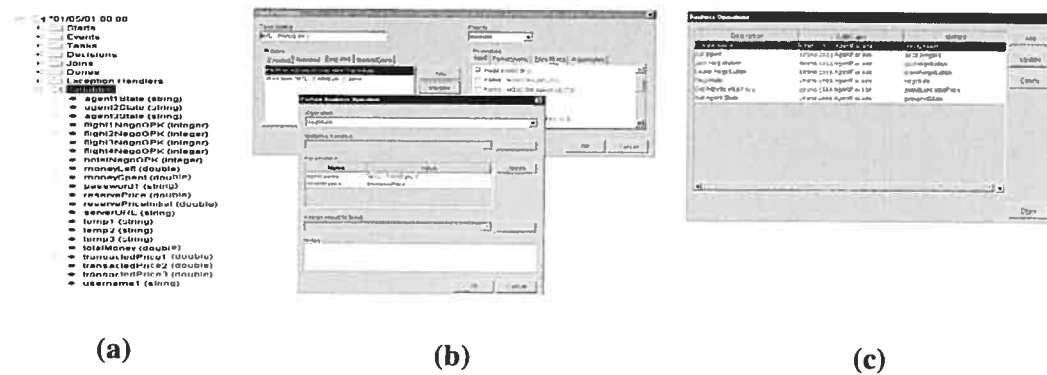


Figure 4.4. WLPI Studio Unit. (a) Workflow Variables, (b) Invoking a Business Operation, (c) List of Business Operations

Business operations are another important concept in the definition of a CN. Defined as a set of beans and methods that implement customized actions, they are called at the “Action” level using “Perform Business Operation”. Figure 4.4(b) shows the “task properties” of the “MTL-PARIS (AC)” task. When the action is executed, the business operation “Negotiate” (a Java method) should be called. The business operation is given workflow variables and constants as parameters. The list of all available business operations for the “Flight Connection” CN example is given in Figure 4.4(c).

4.2.3.2 Enactment Unit

Once a CN model is created and stored in a database, the model is instantiated, and the workflow engine (part of the Enactment Unit) can then start executing the activities in the instance by creating and invoking the software agents responsible for the individual negotiations.

In fact, some applications (*e.g.*, Microsoft Office applications such as Excel and Word) are workflow-enabled and can be invoked directly by the workflow engine, whereas other applications – such as negotiation servers – are not compatible with the standardized workflow interface, and their integration into the business process may be achieved

via a software agent. The latter takes the role of an actor, which is defined in the context of a WfMS, as being a resource that performs a task. It is invoked by a workflow engine, and enables indirect interaction of this engine and the application in question.

Under this perspective, the agents within CONSENSUS are first created, and then the workflow engine invokes them. An indirect interaction between the workflow engine and the negotiation servers is observed. These negotiation servers are not initially compatible with the workflow engine; their integration is only possible via the actor agents.

Note that the agents are invoked by the workflow engine using RMI (Remote Method Invocation), and they communicate with this engine by sending XML events. They participate in negotiations taking place on instances of negotiation servers (e.g., GNP [BKL+00]). The exchanges (e.g., orders, bids, responses) are made using XML documents. The Agent Control and Monitoring tool, also part of the Enactment Unit, is used to watch the progress of the individual negotiations. Figure 4.5 shows a screenshot of this tool during the execution of the “flight connection” CN.

Finally, the user of the system can track and monitor the progress of the CN at run-time, and she can adjust certain intrinsic constraints. Examples include adjusting the total price she is willing to pay, or changing the range of acceptable dates for her flight.

The screenshot shows a window titled "CNSS Agent Control and Monitoring" with a "Main" tab. It contains a table of agents and a "Messages" section with a scrollable log.

agent	nGPK	nState	res\$	ask\$	sent\$	allocated	alloc\$	adj\$	aState
MTL - PARIS (AC)									OUT
PARIS - MOSCOW (AF)	720919	CLOSED	1220 0	430 0	430 0	true	430 0		WINNING
PARIS - MOSCOW (AeroFlot)									OUT
HOTEL - PARIS	720941	OPENED	1590 0	230 0	230 0	true	230 0		LEADING
MTL - PARIS (AF)	720897	CLOSED	800 0	380 0	380 0	true	380 0		WINNING

Messages

```

[CMTool] Connected to CNSS server dev10 lub umontreal ca 7777
[CMTool] Retrieving agent list
[CMTool] Done
[AgentManager] Created agent 'MTL - PARIS (AC)'
[AgentManager] Created agent 'PARIS - MOSCOW (AF)'
[AgentManager] Created agent 'PARIS - MOSCOW (AeroFlot)'
[AgentManager] Created agent 'HOTEL - PARIS'
[AgentManager] Created agent 'MTL - PARIS (AF)'
[PARIS - MOSCOW (AF)] Connected to t3 //labo13 lub umontreal ca.8888
[HOTEL - PARIS] Connected to t3 //labo13 lub umontreal ca.8888
[MTL - PARIS (AC)] Connected to t3 //labo13 lub umontreal ca.8888
[PARIS - MOSCOW (AeroFlot)] Connected to t3 //labo13 lub umontreal ca.8888
[MTL - PARIS (AF)] Connected to t3 //labo13 lub umontreal ca.8888
[MTL - PARIS (AC)] Joined negotiation #720908
[PARIS - MOSCOW (AF)] Joined negotiation #720919
[HOTEL - PARIS] Joined negotiation #720941
[MTL - PARIS (AF)] Joined negotiation #720897
[PARIS - MOSCOW (AeroFlot)] Joined negotiation #720930
[MTL - PARIS (AC)] Witnessed quote Quote[GPK=720916,time=17:16:30,nGPK=720908,state=OPENED,$=200 0]
[PARIS - MOSCOW (AeroFlot)] Witnessed quote Quote[GPK=720936,time=17:15:54,nGPK=720930,state=OPENED,$=200 0]
[HOTEL - PARIS] Witnessed quote Quote[GPK=720949,time=17:17:04,nGPK=720941,state=OPENED,$=200 0]

```

Figure 4.5. Agent Control and Monitoring Tool

4.2.3.3 Coordination Unit

In CONSENSUS, the workflow captures the logic of the CN (i.e., its intrinsic and procedural constraints), whereas the agents capture the logic of the individual negotiations. The agents, by participating in the workflow, share information and cooperate in conducting the CN. They are provided with “individual negotiation” knowledge, as well as with “coordination” knowledge. This knowledge is declarative, and thus it is represented as “if-then” rules which are exploited using an inference engine. In brief, the Coordination Unit has a rule-base, which contains the rules, and a rule engine for exploiting these rules. For more details on this aspect, refer to [BAK01].

4.2.4 Towards a Dynamic Version of CONSENSUS

Workflows are a major enabling technology for CN [BAV+01], and CONSENSUS provides the user a support system to favorably resolve a CN workflow. Supporting dynamic modifications to the CN instance during run-time should however increase the benefits of the CONSENSUS approach. In Chapter 5, we highlight the need for such support and we discuss a solution for a dynamic version of CONSENSUS.

Extensions proposed to CONSENSUS are motivated by events such as the arrival of new offers that may be proposed by the counterpart during a specific e-negotiation, and the willing to avoid a break commitment activity. These offers may necessitate to cancel an already scheduled e-negotiation activity (e.g., if the item to be negotiated is covered by the proposed offer), to move an e-negotiation activity earlier in the process (e.g., if there is a possibility to receive an interesting offer during this e-negotiation that may influence the rest of the scheduled e-negotiation activities), to insert a new e-negotiation activity (e.g., if all scheduled e-negotiations concerning a specific CN item were lost), and so on. A detailed discussion of a scenario asking for a modification of a CN instance is introduced in Section 5.2.1. Moreover, this section identifies other less obvious requirements towards adaptive workflows that stem from the modeling of CNs using the ADEPT WfMS. Those requirements include the dynamic change of decision nodes and the dynamic change of attributes. In brief, the extension proposed to CONSENSUS allows for bringing dynamic modifications to CN instances during run-time: deletion,

move, and insertion of an e-negotiation activity, deletion of an already defined e-negotiation attribute.

4.3 The Multi-Transfer Container Transportation Application

The Multi-Transfer Container Transportation (MTCT) – that could be extended to multi-modal freight transportation – can be defined as the action of moving a container from one terminal to another with the possibility to shift it from one vehicle to another before delivering it to the final destination. The MTCT is considered as one of the sectors in which the fleet management at the operational level is highly dynamic. Other sectors include rescue and emergency services (e.g., ambulance transportation), sanitation, urban transportation, and express mail services. As described by Crainic [Cra02], fleet management covers the whole range of planning and management issues from procurement of power units and vehicles to vehicle dispatch and scheduling of crews and maintenance operations. This type of management can be tackled under various lengths of the planning horizon and levels of details: the strategical, the tactical and the operational level. The latter involves a short planning horizon where the level of details is relatively high. In our work, we focus on the MTCT at the operational level, in which a close follow-up of activities must be achieved to ensure a good customer requests satisfaction.

In the context of the MTCT management, it appears that the processing of a customer request for container transportation can be achieved by a specific sequence of interdependent activities: e.g., attach an empty container to a vehicle, move the empty container to origin location, load the container, move the container to the final destination, unload the container. Moreover, the MTCT requires to create just-in-time the sequence of activities needed to accomplish a request. It also requires a high degree of adaptation of the ongoing activities' sequences to deal with unexpected events (e.g., newly request arrival, delayed vehicles, crew members desistance, technical problems). A solution, based on workflow technology, for the processing of customer requests is investigated.

In the following, we first describe the MTCT application. Then, we give an example of scenario(s) in which the processing of customer requests is required. Finally, the adopted approach for planning the processing of customer requests is presented.

4.3.1 Description of the Application

From a customer request processing perspective, container transportation is constituted of a number of activities of different duration which range from the delivery of empty containers to the origin location where goods are located, to the returning of these containers to depots/terminals. These activities need to be performed in a certain order (“composing activities”), and they are scheduled within a given time window depending on the individual request information, on the resource availability and on the possible paths to follow.

Request Information. A customer request for container transportation is usually well defined; it gathers at least the following information (that we will consider thereafter in this thesis): an origin location where goods are picked-up, a destination where goods are delivered, a pick-up time window and a delivery time window. Other information such as goods characteristics (e.g., item description, packaging type, weight, volume, storage temperature control) may be involved as well. All this information is used – among other information (i.e., resource availability and possible paths to follow) – as input to determine attributes related to the different activities. An empty container is chosen for instance taking into account the volume of the goods, and it is delivered to customer for goods’ loading at a specific time (i.e., pickup time), and at a specific location (i.e., origin location).

Transportation Resources. A set of transportation units which we call (material and human) “resources”, may be composed of a fixed number of containers with fixed wheels, trucks (i.e., vehicles) without loading space, crews (i.e., drivers) and terminals. We suppose that the transportation company offers a full container-load, where one container carries at one time only merchandise related to one client. These resources can be assigned to activities as specific attributes. We call these attributes “input attributes” when referring to material resources and “assign attribute” when referring to human resources.

Activity Templates. A set of activities that we call *activity templates*, are defined. A composition of these activities provides a possible solution to satisfy a customer request. An activity is assigned to a specific driver who becomes responsible for its execution within a specific time and by taking into account information related to the assigned material resources (i.e., input attributes). Table 4.1 shows an example of a set of six activity templates. A possible composition of these activities to satisfy a customer request could be the following sequence: (1)-(3)-(6)-(4)-(3)-(2)-(6)-(1)-(3)-(6)-(5)-(3). Note that a “wait at location” activity is sometimes necessary before going further in the processing of a request.

Table 4.1. Activity Templates Involved in the Processing of a Customer Request for Container Transportation

	(1) Attach container to vehicle	(2) Detach container from vehicle	(3) Move vehicle to location	(4) Load container	(5) Unload container	(6) Wait at location
Input attributes	Container Vehicle Location	Idem ¹	Container Vehicle O_location ² D_location ³	Container Location	Idem	Idem
Assign attribute	Driver	Idem	Idem	Idem	Idem	Idem
Time attributes	MinD/MaxD ⁴ WUT ⁵ EST/LST ⁶	Idem	Idem	Idem	Idem	Idem

¹The same as left, ^{2,3}The origin (resp., destination) location of the activity, which does not necessary correspond to the origin (resp., destination) location of a customer request, ⁴The minimum/maximum duration, ⁵The Warm-Up Time: time when the driver is informed about the activity to carry out, ⁶The earliest/latest starting time.

Paths Scenarios. The composition of activities to satisfy a specific customer request should also be based on a transportation network in which a number of nodes (i.e., locations) and edges (i.e., paths) between these nodes are defined. As a first configuration, we consider a transportation network with a central depot or terminal where resources are located and where a transfer is possible. A transfer is defined as the action of shifting a container from one vehicle to another vehicle. As an example, the sequence “(2)-(6)-(1)” in the composition presented above, represents a transfer.

Taking into account this configuration, a number of path scenarios are possible for the management of customer requests. The simplest scenario would be to consider that the

satisfaction of a customer request consists to ask a couple container/driver (c/d – We consider that each driver is associated with a specific vehicle.) to leave the depot P at a specific time, to pick up the goods at the origin location O specified by the request, to deliver the goods at the final destination D and then to go back to P . In other words, satisfying a request consists of accomplishing the path $P-O-D-P$ (“simple scenario”).

Another scenario would be to ask a couple c/d to leave P at a specific time, to pick up the goods at O and to go back to P with the possibility to make a transfer at P (i.e., to change the driver and the vehicle at P) before delivering the goods at D and then to go back to P (i.e., $P-O-P-D-P$). This represents a “transfer scenario”. It can be motivated by the non-availability of drivers. In this case, we hence need to plan a path $P-O-P$ when a driver is just available to make this portion of the whole path.

In the first two scenarios, c/d should return to P before satisfying a new request. We may however consider that a couple c/d is free to answer a new request as soon as the goods are delivered at a specific destination (i.e., $P-O_1-D_1-O_2-D_2-P$, where O_1/D_1 are related to a specific request and O_2/D_2 are related to another request). We use the term “round scenario”. A combination of the transfer scenario and the round scenario is also possible. Here is an example: $P-O_1-P-D_1-O_2-D_2-P$.

The scenarios presented above take into account a transportation network with a central depot. This transportation network configuration could be extended to a more complex one that gathers a number of distributed depots. Considering this configuration, a “multi-transfer scenario” of the kind $P_1-O-P_2-P_3-\dots-P_n-D$ (where $\{P_1, P_2, \dots, P_n\} \in \mathcal{P}$, \mathcal{P} being the partition of the set of depots) is possible.

Unexpected Events. In a transportation environment, the planning (i.e., the composition of activities) or the re-planning (i.e., the review of the already composed sequence of activities) of a customer request processing is triggered by the occurrence of specific unexpected events. The list of events we are exposing here is not only related to the MTCT application; on the contrary, very similar events may also appear in other sectors such as the express mail and the emergency services.

- *Arrival of new requests.* This is the principal event that can occur. Its satisfaction requires to define a new sequence of activities to be accomplished. In addition, in case of “urgent” customer request (i.e., request that need to be processed in a relatively short time after their arrival), some forecast requests may not materialize, and already planned activities related to requests in a processing phase, may need to be reviewed and adjusted. In fact, a new planning may impact a previous planning (already launched, or waiting to be launched).
- *Delayed vehicles.* If traffic is slower than predicted (e.g., accident, congestion), an adaptation of already planned activities may be required. Indeed, a delay of a particular vehicle can make impossible the execution of the next activities as planned: the latest beginning time of these activities may be exceeded; their assigned crews may be no more available, etc.
- *Crews (e.g., drivers) desistance.* In this case, a re-allocation (or re-assignment) of an activity is desirable. We also refer to the *dynamic* allocation of crews in uncertain environments. Sometimes, crews’ unavailability makes this re-allocation impossible at a specific time. Hence, we may think about modifying the concerned activity by changing for instance its (forecasted) schedule. Of course, this change may require other modifications either within the same sequence of activities or within other sequence(s).
- *Technical problems.* These problems are related to resources such as vehicles and loading machines that are unavailable for a certain period of time. Consequently, sequences of activities should be modified taking into account the re-allocation of available resources, or the delay to fix the problem.

The MTCT application just described can be considered as a Pick-up and Delivery Problem (PDP). A number of papers discuss methods developed for solving this problem [SS95, Mit98]. A distinction is done between PDP with (soft/hard) time windows and PDP without (soft/hard) time windows. The time window constraint complicates the formulation and the resolution of the problem. In general, researchers in the domain consider the “arrival of a new request” as an event triggering the (dynamic) management of resources and the scheduling of a set of routes. Other events such as the ones presented above are of course identified, but they are rarely studied since studying the “arrival of a

new request” is already considered as a complex decision problem where the decision must be taken under considerable time pressure. Algorithms and heuristics are proposed as a means to tackle this dynamic problem and optimize the planned routes between the occurrences of new events. Examples of these algorithms and heuristics include the insertion procedure [WSW+70], neighborhood search heuristics [GGP+98] (e.g., tabu search), and neural networks [PSR92]. Since our main interest in this thesis is not to study the PDP problems and solutions, but rather to focus on the workflow aspects in the particular MTCT application, we will not go into details regarding this topic. Interested readers may refer to [SS95, Mit98] for a survey of the methods used to solve the PDP problems.

4.3.2 Examples of Customer Request Processing Planning

In this section, we illustrate the different steps for satisfying a customer request taking into account the simple path scenario and the transfer path scenario discussed in Section 4.3.1. First, a simple example is exposed, and then an example involving an already planned customer request is proposed.

4.3.2.1 Customer Request Processing Planning – Simple Example

When a request is received, its related information becomes available at the transportation company side. This information, the availability of the resources and the transportation network information are used to generate a solution (if any) for the processing of this request.

Suppose that the transportation network the company is covering is the one shown in Figure 4.6. 20 locations are identified. For this example, we consider a configuration with a central depot (e.g., Drummondville). The remaining 19 locations are used to locate the origin and the destination of the received customer request. The distance (in km) between the different locations can also be expressed in duration (in minutes) such as it is shown in Table 4.2.

Suppose that the transportation company owns a set of containers: C111, C222, C333, etc., a set of vehicles: V101, V202, etc., and that a number of drivers are working for this company: McCain, Watson, etc.

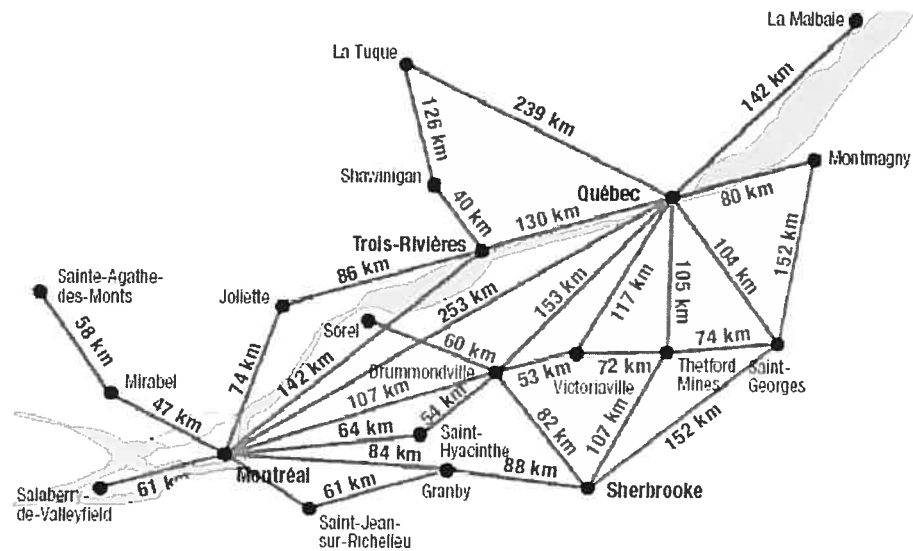


Figure 4.6. Example of a Transportation Network, adapted from [Tra04]

Table 4.2. Duration Between Two Locations (in minutes)

	Montréal	Trois-Rivières	Québec	Drummondville	Sherbrooke	..
Montréal		105	165	75	115	
Trois-Rivières			90	166	209	
Québec				105	209	
Drummondville					60	
Sherbrooke						
...						

Suppose that the following customer request information is received:

Origin location: Québec
Destination location: Montréal
Earliest pickup time: 15/10/2003 08:30
Latest pickup time: 15/10/2003 10:30
Earliest delivery time: 15/10/2003 13:30
Latest delivery time: 15/10/2003 15:00

And that the current reservation of resources is the following, where <st, ft> corresponds to the starting and finishing time of the resource reservation:

C111: {<16/10/2003 09:30, 16/10/2003 13:30>}
C222: {<16/10/2003 09:20, 16/10/2003 16:00>}

C333: {<15/10/2003 09:30, 15/10/2003 14:00>, <16/10/2003 09:20, 16/10/2003 16:00>}
 ...
 McCain/V101: {<15/10/2003 09:30, 15/10/2003 14:00>, <16/10/2003 09:20, 16/10/2003 16:00>}
 Watson/V202: {<16/10/2003 09:20, 16/10/2003 16:00>}
 ...

Based on the above information regarding the customer request, the transportation network, the (non-)availability of the resources, and taking into account the simple path scenario discussed in Section 4.3.1, a solution such as the following one can be found:

Solution found...

Container: C111
Driver: Watson/V202

Depot – starting time: 15/10/2003 08:10
Depot – attach container: <<5 min.>>
Depot – leaving time: 15/10/2003 08:15 <<105 min.>>

Origin – arrival time: 15/10/2003 10:00
Origin – load container: <<30 min.>>
Origin – leaving time: 15/10/2003 10:30 <<165 min.>>

Destination – arrival time: 15/10/2003 13:30
Destination – unload container: <<30 min.>>
Destination – leaving time: 15/10/2003 14:00 <<75 min.>>

Depot – arrival time: 15/10/2003 15:15

****Waiting time before delivery: <<15 min.>>*

A basic workflow model that corresponds to the simple path scenario and that captures a sequence of activities defined between a “start” activity and an “end” activity can be instantiated: (S) start, (A1) attach container to vehicle, (A2) move vehicle to O, (A3) wait at O, (A4) load container, (A5) move vehicle to D, (A6) wait at D, (A7) unload container, (A8) move vehicle to P, (E) end. Since the solution proposed does not specify a waiting time at O, the activity (A3) should then be deleted from the instance. Note that in this case, the activities constitute a simple sequence of actions. Other examples may yield to activities whose control flow is best captured in a state-transition diagram.

The solution found reflects the different attributes (input, assign and time attributes) of the activities, except for the WUT (Warm-Up Time) introduced in Table 4.1, and that will be discussed later in Chapter 6, Section 6.3.3.3. These attributes should be given as input to the different activities of the workflow instance.

4.3.2.2 Customer Request Processing Planning – Re-planning Example

Following the reception of a new request NR, the current reservation of resources is considered so that an available couple c/d is found for the processing of NR. However, it may appear that no solution is possible for NR, even when considering the different path scenarios. This can happen if for instance no driver is available to satisfy NR taking into account the specified pickup and/or delivery time windows (we suppose that $\#(\text{containers}) > \#(\text{drivers})$ holds). This situation may lead to consider the requests for which the processing was already planned and for which the activity “Move vehicle from O to D” is not reached yet. Let R be the set of these requests. A solution for NR may become possible when modifying the solution already proposed for one of the requests of R. Let OR be this request: (1) a new solution for OR is found (e.g., by inserting a transfer at P, and by removing the waiting time at D), and (2) a solution that satisfies NR (e.g., according to the simple scenario) is now possible since the driver previously reserved for OR is now released. Refer to Figure 4.7.

Suppose that the following information corresponds to NR:

Origin location: Sherbrooke
Destination location: Montréal
Earliest pickup time: 15/10/2003 13:30
Latest pickup time: 15/10/2003 14:30
Earliest delivery time: 15/10/2003 14:30
Latest delivery time: 15/10/2003 17:00

And suppose that the current reservation of resources is the following:

C111: {<15/10/2003 08:10, 15/10/2003 15:15>, <16/10/2003 09:30, 16/10/2003 13:30>}
C222: {<15/10/2003 09:30, 15/10/2003 11:40>, <16/10/2003 09:20, 16/10/2003 16:00>}
C333: {<15/10/2003 09:30, 15/10/2003 10:30>, <16/10/2003 09:20, 16/10/2003 16:00>}
 ...

McCain/V101: {<15/10/2003 11:40, 15/10/2003 13:05>, <16/10/2003 09:20, 16/10/2003 16:00>}
 Watson/V202: {<15/10/2003 08:10, 15/10/2003 15:15>, <16/10/2003 09:20, 16/10/2003 16:00>}
 ...

We suppose that the identified OR (the request for which the already planned processing should be modified) is the request considered in section 4.3.2.1; and that the new solution found for the processing of this request is the one specified below. Note that in this solution, strikethroughed elements are the elements that were removed and bolded elements are the ones that were added, when comparing with the old solution (shown in Section 4.3.2.1). The solution for the processing of NR is proposed thereafter.

New solution found for OR...

Container: C111
Driver: Watson/V202

Depot – starting time: 15/10/2003 08:10
Depot – attach container: <<5 min.>>
Depot – leaving time: 15/10/2003 08:15 <<105 min.>>

Origin – arrival time: 15/10/2003 10:00
Origin – load container: <<30 min.>>
Origin – leaving time: 15/10/2003 10:30 <<~~165 min.~~>> <<105 min.>>

Depot – arrival time: 15/10/2003 12:15
Depot – detach container: <<5 min.>>
****Waiting time before attach: <<46 min.>>*
Driver: McCain/V101
Depot – attach container: <<5 min.>>
Depot – leaving time: 15/10/2003 13:11 <<75 min.>>

Destination – arrival time: 15/10/2003 ~~13:30~~ 14:26
Destination – unload container: <<30 min.>>
Destination – leaving time: 15/10/2003 ~~14:00~~ 14:56 <<75 min.>>

Depot – arrival time: 15/10/2003 ~~15:15~~ 16:11

*~~***Waiting time before delivery: <<15 min.>>~~*

Solution found for NR...

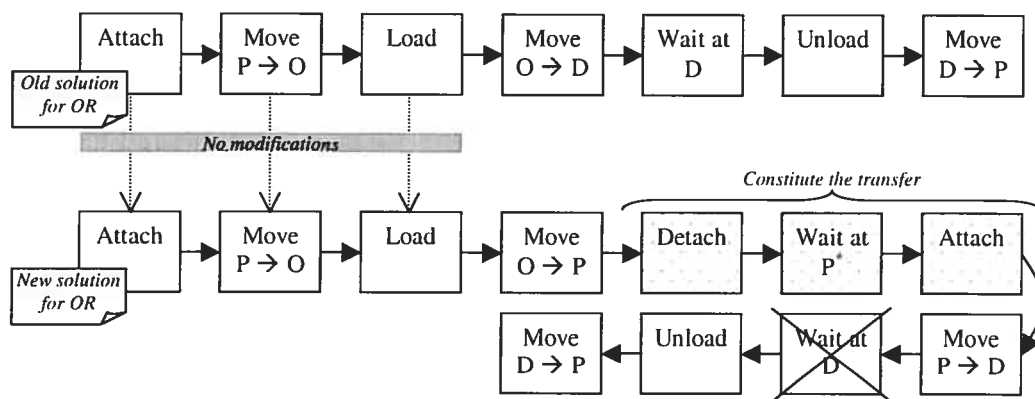
Container: C222
Driver: Watson/V202

Depot – starting time: 15/10/2003 12:25
 Depot – attach container: <<5 min.>>
 Depot – leaving time: 15/10/2003 12:30 <<60 min.>>

Origin – arrival time: 15/10/2003 13:30
 Origin – load container: <<30 min.>>
 Origin – leaving time: 15/10/2003 14:00 <<115 min.>>

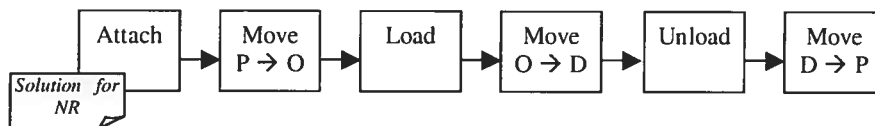
Destination – arrival time: 15/10/2003 15:55
 Destination – unload container: <<30 min.>>
 Destination – leaving time: 15/10/2003 16:25 <<75 min.>>

Depot – arrival time: 15/10/2003 17:40



* Wait until a driver becomes available to continue the processing of the request.

(a)



(b)

Figure 4.7. Re-planning Example. (a) The Proposed Modifications for the Processing of OR, (b) The Proposed Solution for the Processing of NR

4.3.3 Customer Request Processing

In our work, we are interested in managing the processing of customer requests in a multi-transfer container transportation application. Taking into account the observations of previous sections, we exploit *workflow technology* to model and to concurrently manage this processing. The workflow approach reduces in general the need for manual, time-consuming management and organization, and specific features of workflow tech-

nology can result in positive effects for the transportation domain. These features should however include enhanced concepts and functionality.

Indeed, at the operational level, the MTCT application is faced with a continuously changing environment where resource sharing issues are involved. In this context, the need for dynamic creation and adaptation of (optimized) solutions is of utmost importance. Operations research modules (e.g., resource management module, activity scheduling module) should provide the planning and the re-planning of activity sequences. If such modules indicate that changes must be brought to existing solutions, it should be possible to modify corresponding ongoing workflow instances. These modifications at the workflow level are typically of three types: activity postponement, attribute updating and structural modifications. The first modification type allows (1) to react to the lack of availability of resources or (2) to free some planned resources to reallocate them to other (priority) activities. The second modification type allows for reacting to strategic adjustments that tend to improve the efficiency of the global processing. Finally, the third modification type allows for modifying the sequence of a workflow instance by inserting a new activity (e.g., to accommodate a transfer path scenario) or by deleting an existing one (e.g., to remove a “wait at location” activity).

The MTCT application can amply take advantage of workflow technology once its underlying challenging aspects are accommodated. These aspects should cope with the already discussed dynamic management of instances. They should also properly cope with the definition of basic workflow models and with the instantiation of these models. The definition of basic workflow models should rely on well-defined activity templates and it should map out the different path scenarios. The instantiation of a specific model is considered as a complex and a critical operation since it is based on solutions (i.e., planning of activity sequences) provided by operations research modules. Moreover, new instantiations and modifications of workflow instances need to be automated as much as possible so that time-consuming manual interactions are reduced.

We are aware of the fact that workflow technology in the transportation domain is usually applied to manage logistic activities where documents and information are passed from one participant to another according to a set of procedural rules [CCP+98]. How-

ever, the central issue related to workflows in our approach is the focus on supporting *flow of work* and not on supporting flow of “documents” [AM00]. Furthermore, we adopt the idea of emergent workflows described by Jørgensen and Carlsen in [Cra02]: “emergent workflows provide an integrated support for planning, coordination and performance of work”. The workflow definition and enactment are intertwined.

Taking into account what was discussed till now, a workflow-oriented solution applied in the MTCT context should enable the user (i.e., “system administrator”) to efficiently track and monitor the progress of many customer requests in process. Moreover, it should allow crew members (i.e., “drivers”) to identify at the right time their assigned activities and to transmit to the system administrator the state of each activity from its selection to its completion. This will allow, among other things, for determining at any time the state of the different resources.

In chapter 7, we introduce the MTCT system with all its constructs to support the processing of customer requests. As it was motivated all along Section 4.3, the solution provided by the MTCT system is workflow-oriented.

4.4 Summary

In this chapter, we introduced a set of workflow-supported applications that were already discussed in the literature. A motivation for studying complex socio-technical systems was given. The major part of the chapter was devoted for describing the combined negotiation application and the multi-transfer container transportation application: two examples of non-trivial socio-technical applications. CONSENSUS, a workflow-oriented combined negotiation support system, was discussed and a dynamic solution was introduced. This solution will be the subject of our next chapter. The functionality of the MTCT system was also specified. This system will be described in detail in Chapter 7.

The CONSENSUS application and the MTCT application appeared to be well chosen since it allowed us to identify an interesting set of new requirements for enhanced workflow technology:

- The *activity template concept* defined as a standalone activity designed without being part of a workflow model.
- The *template classification* that assigns a specific category to a set of activity templates and workflow models.
- The *warm-up time concept* defined as the necessary time to inform a human actor about an assigned activity.
- The *dynamic insertion of a new activity* at the workflow instance level.
- The *dynamic insertion of a sub-workflow* at the workflow instance level.
- The *dynamic deletion of a scheduled activity* at the workflow instance level.
- The *dynamic deletion of a running activity* by preserving its context.
- The *dynamic move of an activity* at the workflow instance level.
- The *dynamic insertion/deletion of an activity attribute* at the workflow instance level.
- The *dynamic setting/updating of attributes* at the workflow instance level.
- The *dynamic management of work-lists* as a consequence of dynamic modifications.

In Chapter 5, we better develop the requirements stemming from the CONSENSUS application. Then, in Chapter 6, we address each of the identified requirements in the best appropriate manner.

Chapter 5 The Enhanced CONSENSUS System

In this chapter, we reconsider the combined negotiation support system (CONSENSUS) discussed in Section 4.2.3. The WfMS-based CONSENSUS platform was developed to help the user model and enact combined negotiations (CNs). A CN is modeled as a workflow, and at run-time, software agents participate in negotiations as actors in the workflow. In this chapter, we highlight the need for a dynamic version of CONSENSUS. Indeed, this system requires support for dynamic ad-hoc changes induced by unexpected events that can occur during negotiation. IBM MQ Series Workflow and BEA Systems WLPI support this kind of dynamism in a limited way. Consequently, the benefits of the workflow-based CONSENSUS approach to e-negotiations, namely, minimizing the risk of commitment breaking and maximizing the chances of good deals, are slightly reduced. To cope with the required flexibility, we experiment using ADEPT in the context of CNs. ADEPT is considered as a state-of-the-art adaptive WfMS. We show to which extent this system is able to support dynamism as required by e-negotiations, and we outline requirements that should be supported by adaptive WfMSs to fully satisfy the nature of such dynamism. A dynamic version of CONSENSUS based on ADEPT is discussed, and an overall adaptive workflow framework is proposed. This framework extends the WfRM [WfMC95] introduced in Section 2.4.1, for supporting adaptive workflows in the context of a specific application. The “importing package” example introduced in Section 4.2.2.2 will be the running example in our discussion.

The remainder of this chapter is organized as follows: In Section 5.1, we identify a number of dynamic scenarios in the “importing package” example. In Section 5.2, first, we demonstrate that ADEPT is fit to cope *to some extent* with dynamism in the context of CNs; then, we discuss the CONSENSUS version based on ADEPT. In Section 5.3, we provide the overall architecture as an extension to the WfRM for supporting adaptive

workflows. Section 5.4 wraps up the chapter by focusing on the requirements of CONSENSUS towards adaptive workflow technology.

5.1 Dynamic Aspects of the “Importing Package” Example

Although it is widely recognized that WfMSs should provide flexibility, most of today’s systems unfortunately have problems dealing with changes [RRD04a]. However, various contingencies and obstacles that can appear during negotiation may require changes at the workflow instance level.

Taking into account the “importing package” example introduced in Section 4.2.2.2, an obvious dynamic change could come up immediately after negotiating the purchase of the goods. The supplier could offer, for instance, to cover the freight shipment and insurance from the warehouse of origin to the warehouse of destination. The buying company could be interested in this offer, and hence decides to not engage in any of the subsequent negotiations of the CN (i.e., transportation, insurance, forwarding). It should be possible for the buying company to remove all these scheduled negotiations from the workflow instance during run-time.

Obviously, a similar offer could also come from the forwarding agent. In this case, the buying company might find it interesting to engage in the negotiation with the forwarding agent in parallel with transportation, and thus the possibility to move the “forwarding” activity right after the “purchase of goods” activities becomes necessary. In case the negotiation with the forwarding agent succeeds covering the freight shipment and insurance, a next step would be to delete all the negotiation activities related to transportation and insurance.

Among other possibilities, the two dynamic scenarios described above could occur in a real-world importing process. Other dynamic scenarios may appear as well in the context of other CN processes, such as the vacation package and the flight connection package presented in Section 4.2.2. Hence, it would be advantageous for a Combined Nego-

tiation Support System (CNSS) to allow on-the-fly changes while a CN instance is running.

5.2 The CONSENSUS System Based on an Adaptive WfMS

In order to address dynamic aspects in CNs, we experimented using ADEPT. In Section 5.2.1, first, we review the different components of ADEPT that are of interest for the CONSENSUS approach; then, we discuss the modeling of the “importing package” example as well as the possibilities and characteristics of ADEPT with regards to changes. In Section 5.2.2, the integration of ADEPT within CONSENSUS is discussed.

5.2.1 Dynamic Modifications Using ADEPT

As already introduced in Chapter 3, ADEPT offers support for ad-hoc dynamic changes. The ADEPT Workflow-Editor is a build-time client application for modeling activities and workflows. It corresponds to the Process Definition Tool of the WfRM. As with WLPI, the workflow model is stored in a database. The provided ADEPT Client monitors the execution of a workflow instance. It corresponds to a Workflow Client Application when referring to the WfRM. The user can intervene, via the ADEPT Client, by inserting or deleting an activity to the instance already created and launched. The activity to insert should exist in one of the instances already created, including the ones related to a different workflow model. It is not allowed to define/model a new activity during runtime. Of course, a certain number of constraints must be satisfied before proceeding to the modification steps, i.e., correctness verification (cf. Section 3.2.2.4).

We used ADEPT to model and run CN processes in order to address the dynamism issue in CNs. Two main criteria were applied to retain this system among other adaptive WfMS prototypes (cf. Section 3.3). Indeed, the first and foremost criterion is its compliance with the WfRM, whereas the second criterion concerns the availability of its API.

Figure 5.1 shows the “importing package” example as provided by the ADEPT Client. This example is based on the second scenario described in Section 5.1. Activities in Figure 5.1(a) correspond to the different negotiations of the “importing package” as shown

in Figure 4.2. Two “empty” nodes are used for the And-split and the And-join of the “purchase (supplier 1 *and* 2)” activities (nodes S1 and S2). Inserting an activity to the current instance requires synchronization with activities that must be completed before and after the inserted one. In our example, the “forwarding” activity (node F) should be activated after the two “purchase (supplier 1 *and* 2)” activities, and obviously before the “carry out deals” activity (node C). The edge from node S1 (resp. S2) to node F, and the one from node F to node C in Figure 5.1(b) show the synchronization. Figure 5.1(c) depicts the case where the negotiation with the forwarding agent succeeds. All the remaining negotiations related to transportation (nodes T1, T2, and T3) and insurance (node I) are deleted. The “carry out deals” activity is then launched straightaway. Note that it was possible to delete node T1 although it has already been activated. The two activities “forwarding” and “sea shipment” are activated in parallel; however, the “forwarding” activity had to be completed first.

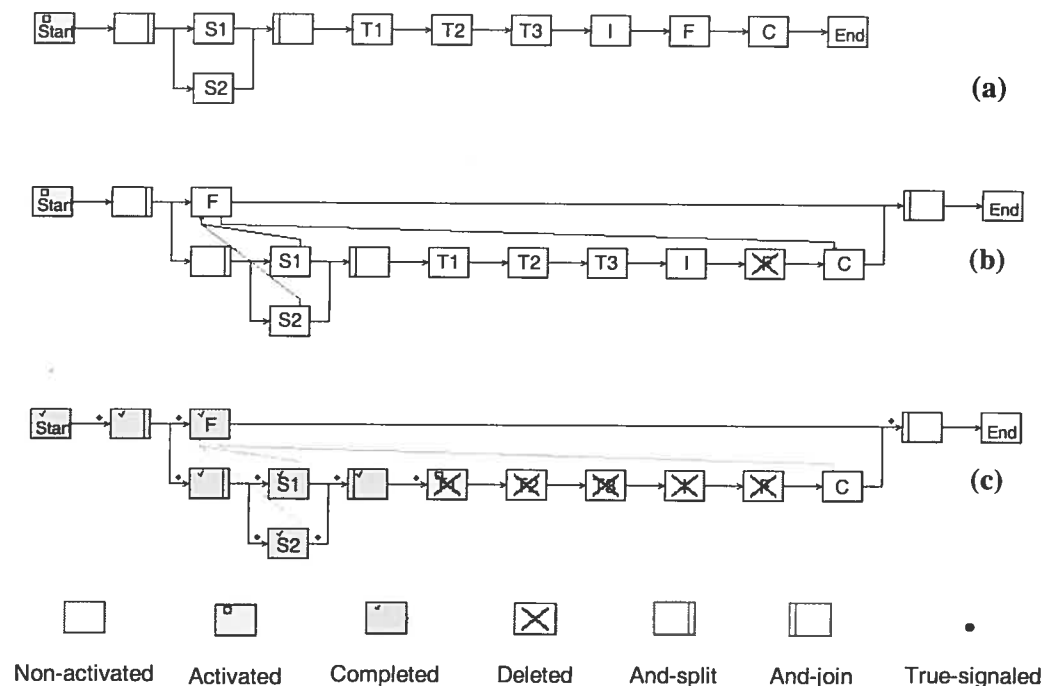


Figure 5.1. Importing Package during Run-time in ADEPT – Modeled without Decision Branches. Instance State (a) After Creation, (b) After Moving Task F, (c) After Deleting Tasks: T1, T2, T3, and I

From our experience with ADEPT as a standalone WfMS, we realized that nodes modeling decisions make CN processes less flexible to deal with changes during run-time. Indeed, adjusting a moved activity with its corresponding decision branch is impossible in ADEPT. The insert operation and the delete operation do not cover “XOr-Split”/“XOrJoin” nodes. Consequently, instead of modeling CNs as it is shown in Figure 5.2, we chose to model them without decision branches at all (cf. Figure 5.1(a)), letting the user decide manually whether to go for the next negotiation in the sequence, to delete specific negotiation(s), or to insert new one(s). Obviously, the user should take into account the results of the previously completed negotiations (e.g., deal or not). The previous argumentation suggests that in order to offer a more flexible model, we need to define less automatic activities, avoiding for instance decision branches.

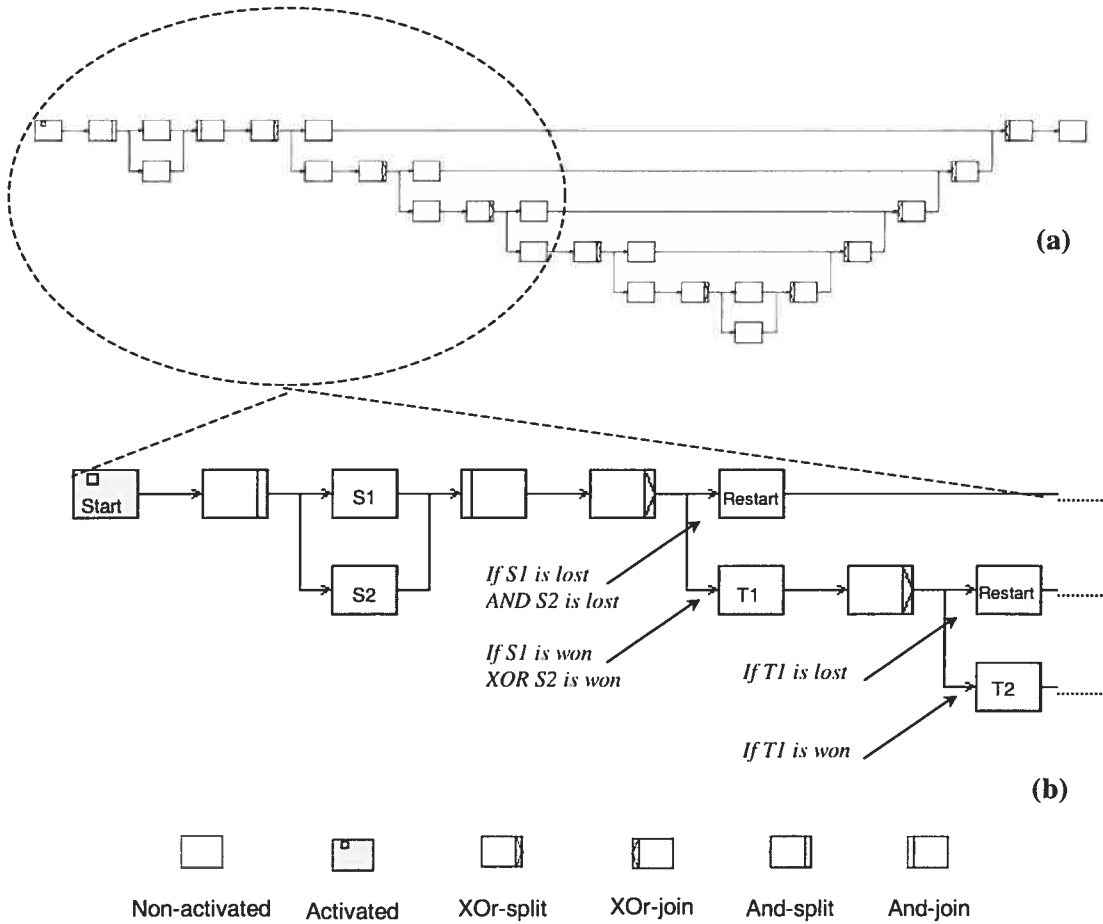


Figure 5.2. “Importing Package” in ADEPT – Modeled with Decision Branches. (a) The Whole Picture, (b) Detailed Part of the Process

In the case of dependent attributes between activities, e.g., an item needs as input the result of a predecessor item, ADEPT does not allow to delete the producing task. This is perfectly coherent. However, since it is not allowed to modify attributes – mainly to delete the attributes that were pre-assigned to the consuming activity – this makes, once again, our model less flexible regarding deletion. Dependent attributes may easily appear in CNs, and a possible solution could be to permit changes of attributes.

Finally, the move operation is not provided by the ADEPT prototype we are using. We had to replace it by a delete followed by an insert.

5.2.2 ADEPT in CONSENSUS

The availability of the ADEPT API makes it possible to implement client applications and work-list handlers for specific domains. Indeed, a client application for the medical domain was implemented and provided within the released version of ADEPT [RT02]. In the context of CONSENSUS, we have implemented a client application that supports the launching of automatic activities. The latter refer to negotiation activities that invoke application-related methods, e.g., methods to create and to destroy an agent, to join a negotiation, to leave a negotiation, to negotiate, to get an agent state and to get an adjudicated price. This feature is not supported by the provided ADEPT Client. We also noticed that the ADEPT API itself does not provide any method that allows the implementation of automatic activities. In contrast, the WLPI API provides such methods which we call from our application (Figure 5.3).

Whenever a negotiation activity is reached, our ADEPT Client Application detects this new activity state (i.e., *Activated*), and it calls successively methods `Automatic_Call_createAgent(...)`, `Automatic_Call_joinNegotiation(...)`, and `Automatic_Call_negotiate(...)`. Once the agent is created, the `Automatic_Call_getAgentState(...)` is called continuously until the agent state becomes `OUT`, `WINNING`, or `LOSING`. Once the agent joins the negotiation, the state of the corresponding negotiation activity becomes *Selected*. The activity state turns to *Running* as soon as the agent begins negotiating, and the activity state remains *Running* as long as the agent state is different from `OUT`, `WINNING`, or `LOSING`. Then, it turns to *Completed*.

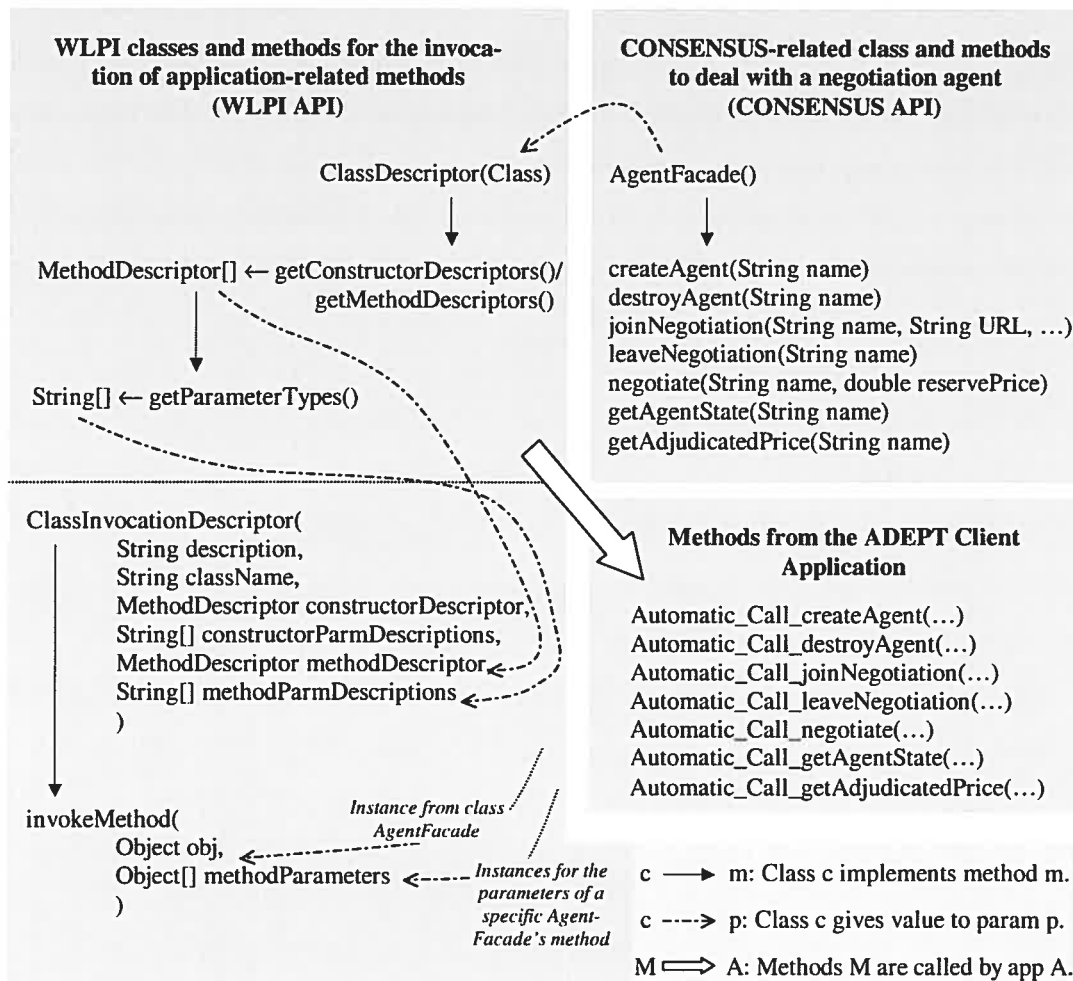


Figure 5.3. WLPI Methods Called by the ADEPT Client Application for the Implementation of Negotiation Activities

Once we integrated ADEPT within CONSENSUS, we made an interesting observation concerning the automatic activities and the opportunities for user intervention. Indeed, as specified in Section 4.2.3, CONSENSUS comprises an *Agent Control and Monitoring Tool* (Figure 4.5) from which the user can monitor the work of the agents responsible of individual negotiations. While the user is interacting with this tool, the workflow instance could not go further in the execution. This gives time for the user to think about a possible adjustment, and to bring appropriate changes to the instance. In the current version of CONSENSUS built on ADEPT, the user should interact with two control and monitoring tools, as defined by the architecture of CONSENSUS [BAV+01]: the work-

flow control and monitoring tool, and the agent control and monitoring tool. The architecture of CONSENSUS was designed before integrating the flexibility feature, and hence does not take care of this extension. Taking into account this new feature, usability can be improved by integrating the workflow control and monitoring tool with the agent control and monitoring tool, so that the user will not have to switch from one window to the other to intervene at the workflow instance level and at the agent level.

5.3 Adaptive Workflow Framework

The architecture of CONSENSUS [BAV+01] (cf. Section 4.2.3, Figure 4.3) should be reviewed in order to support adaptive workflows. For this, we have extended the WfRM by proposing a new overall architectural framework for adaptive workflows (cf. Figure 5.4). This framework allows for designing concrete workflow-oriented system architectures in the context of specific applications.

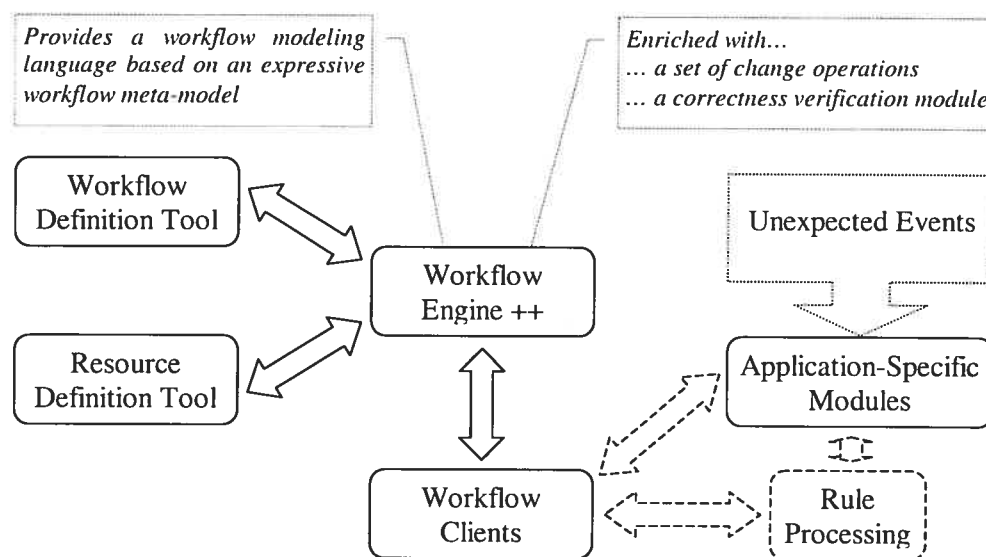
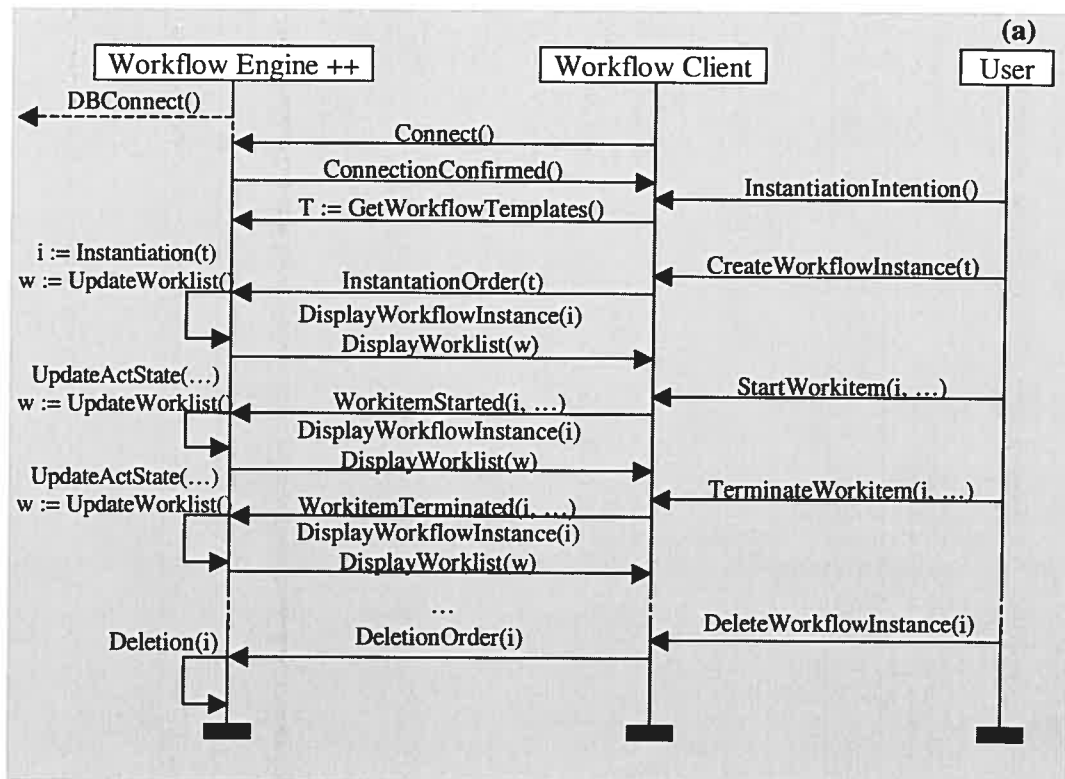


Figure 5.4. Adaptive Workflow Framework

The “Workflow Engine ++” corresponds to the core of the framework. The workflow modeling language provided by the engine should be based on a workflow meta-model that is expressive enough to allow practically relevant changes. The engine needs also to be enriched with a set of useful change operations, and it asks for a correctness verification module (cf. Section 3.1). The “Workflow Definition Tool” and the “Workflow Cli-

ents” are two modules already defined within the Workflow Reference Model (WfRM). The “Resource Definition Tool” is useful to define resources required for the definition of activities, e.g., negotiation agents in the CONSENSUS application and drivers as well as material resources in the MTCT application. “Application-Specific Modules” communicate unexpected events to the “Workflow Engine ++”. The decision regarding the changes that must be applied on the set of workflow instances is either taken by the user of the system (i.e., application domain expert), or derived automatically using a decision module. In the first case, the user specifies the changes via a workflow client. In the second case, the “Rule Processing” module may remedy the lack, within the workflow meta-model, of constructs for *automatic* workflow changes (e.g., events, triggers, rules).

We illustrate in Figure 5.5 the sequence of messages that are exchanged between the “Workflow Engine ++”, a “Workflow Client” module such as a workflow control and monitoring tool, and the user of the adaptive workflow system. We consider the case when a normal execution with no adaptation is required (Figure 5.5(a)), and the case when an insertion (Figure 5.5(b)) or a deletion (Figure 5.5(c)) of an activity is required.



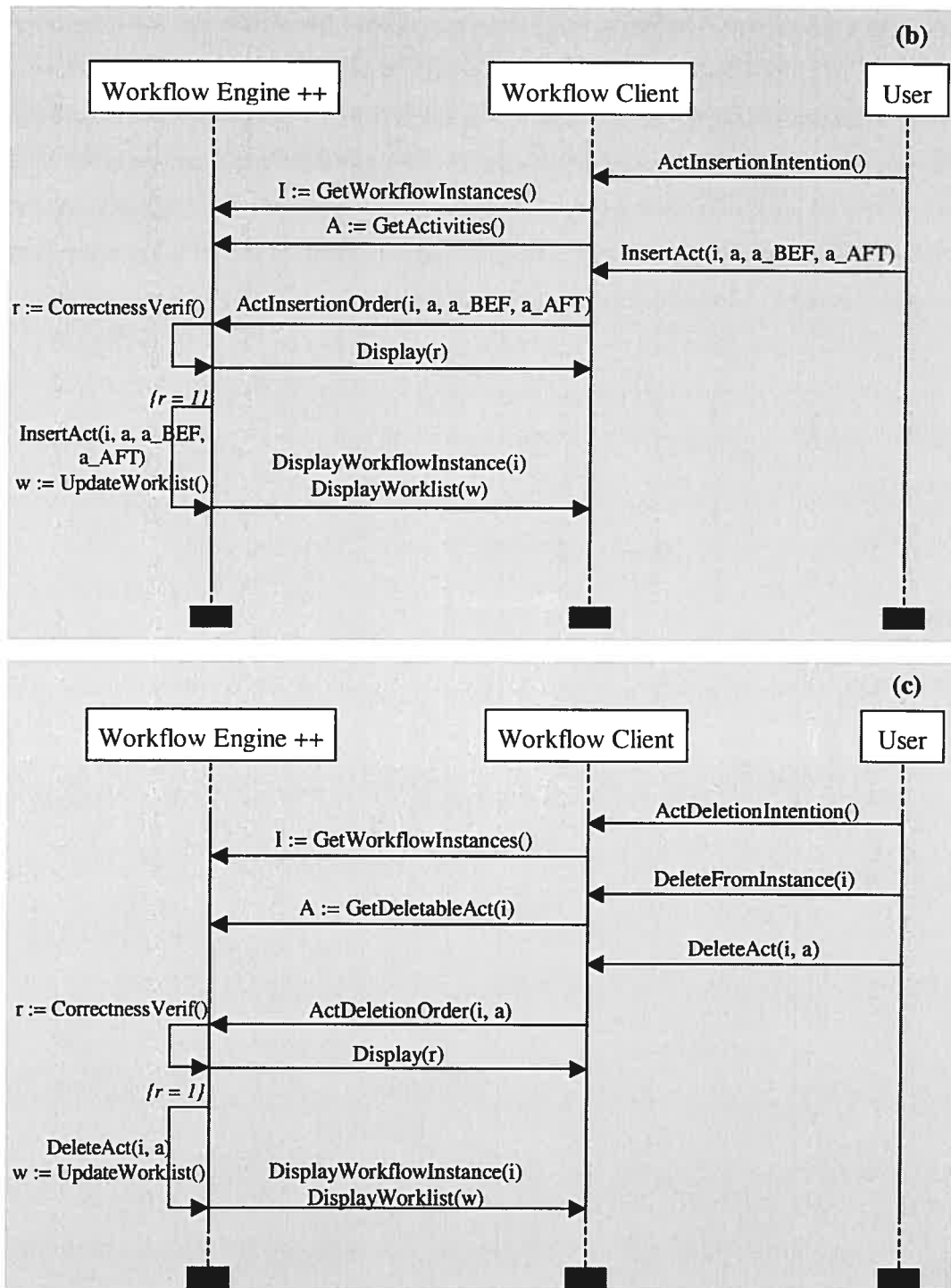


Figure 5.5. Sequence of Messages Exchanged (a) during a Normal Execution of a Workflow Instance, (b) when an Activity Insertion is Required, and (c) when an Activity Deletion is Required

Dealing with unexpected events in the domain of workflows can be compared with error handling in the domain of transactions. We will discuss this issue in what follows.

5.3.1 Adaptive Workflows and Transaction Management

The requirements resulting from dealing with unexpected events are by far more challenging than those faced by standard transaction management (error handling). Standard transaction models define their correctness criteria in terms of the transaction ACID properties [Elm92]:

- **Atomicity:** The transaction is a single unit of processing. Either all of its activities are executed or no activity is executed.
- **Consistency:** The activities are executed only when they result in a consistent state.
- **Isolation:** The activities are executed without the interference of activities of other concurrently executed transactions.
- **Durability:** All results of a committed transaction are persistent, regardless of subsequent system failures.

A workflow can be seen as a possibly long-running transaction, and the ACID properties have to be relaxed in conjunction with these long-running transactions. This is of utmost importance to improve the performance of a system implementing a transactional execution, but also to let more failures recovery. One first step towards relaxing the ACID properties is the definition of nested transactions. Nested transactions [Mos82] allow finer grained recovery and provide more flexibility in terms of transaction execution. Another notion quite similar to nested transactions is the definition of sagas [GS87]. A saga refers to a long-running transaction that can be broken up into a collection of sub-transactions that can be interleaved with other transactions. When compared to nested transactions, sagas only permit two levels of nesting: the top level (saga) and simple transactions, and at the outer level full atomicity is not provided (i.e., sagas may view the partial results of other sagas). A saga relaxes the requirement that a long-running transaction need to be executed as an atomic action. Of course, a compensation mechanism needs to be implemented in order to guarantee that a saga would commit all its sub-transactions or it would roll back any committed transaction. This relaxes the durability

property. The concept of sphere has been defined such that compensation can be applied not only on one activity but also on a group of activities (called sphere) [Ley95]. A detailed discussion of transactions applied in the domain of workflows is given by Worah and Sheth in [WS97].

Further issues in the analysis of correctness properties in adaptive workflows can be discussed in conjunction with transaction management. As an example, deadlocks, which constitute an important problem in transaction management, can appear as a result of modifications in workflows. In transaction management, the blocking of transactions by a two-phase locking can give rise to deadlock, i.e., two or more transactions are simultaneously waiting for each other to release a lock before they can proceed. In workflow management, modifications such as skipping or deleting an activity may result in a deadlock since the successor activities would wait for the termination of the skipped or deleted activity, e.g., in order to provide needed data. Moreover, modifications resulting in undesired cycles may cause deadlocks. In order to ensure the correctness (i.e., soundness) of a workflow after a modification is made, correctness checks need to be carried out. As an example, in the ADEPT approach, modification operations have formal pre- and post-conditions which ensure *by construction* that the resulting process schema does not contain deadlocks. We already discussed, in Section 3.2.2.4, this issue of correctness verification in the context of reviewed adaptive workflows projects. In Section 6.4.2.1.2, we introduce a general correctness criterion ensuring the safe interruption of a running activity.

5.4 Summary and Discussion

CN is a novel negotiation type [BAV+01] that is required, for example, in the context of supply chains and e-procurement. CONSENSUS was probably the first workflow-based system to support CNs. Flexibility has widely been recognized as an important feature of WfMSs in general, but in the context of CNs, the inability to cope with flexibility puts limits to the benefits of the CONSENSUS approach. Indeed, CN requires flexibility to accommodate the various contingencies and obstacles that can appear during negotiation. For example, if a supplier or a shipping company makes a new offer that might be

of interest for a buying company, the buyer will review negotiation activities already planned within the workflow model and may want to rearrange them (e.g., to dynamically delete, replace, or move activities). In this context, the ADEPT change and verification facilities have proven as coherent with the flexibility requirements in CNs. However, there are several requirements identified within the CONSENSUS project that have not yet been fully supported by ADEPT:

- Dynamic change of *decision nodes*, so that the automated execution provided by workflows does not play against the flexibility. In fact, decision nodes play an important role in the computational representation for automated execution.
- Dynamic change of *attributes*, so that the structural change operations provided are not needlessly forbidden. As an example, allowing the deletion of an attribute increases the chances to pass through the verifications that exist behind the activity deletion operation. Indeed, it confronts the dependent attributes problem.
- Dynamic *move operation*. We were applying successively the delete operation and the insert operation to compensate the absence of the move operation. However, the move operation should relax the verifications related to the activity deletion operation. As an example, the verification of dependent attributes should not generate correctness problems when the new position of a moved activity A is still preceding activities taking input from A.

These requirements, gathered from studying CONSENSUS under the “flexibility perspective” and from considering a state-of-the-art adaptive WfMS, help us not only to provide interesting input for the enhancement of the “flexibility” feature of ADEPT, but also to clarify and refine the needs for adaptive workflows in general. Indeed, the identified requirements allow us for deriving an adaptive workflow framework in which we stressed the need for an appropriate set of change operations, for a correctness verification module, and for a workflow meta-model that is expressive enough to allow practically relevant changes.

Chapter 6 Workflow Management Requirements

The CONSENSUS and the MTCT applications studied in the previous chapters serve us to investigate the needs for a set of enhanced concepts and functionality for WfMSs. In CONSENSUS, process activities represent e-negotiations, and software agents are responsible for their execution. Automatic activities are hence involved. There is an interest for manual intervention during run-time (*i.e.*, human involvement in the loop). In the MTCT system, process activities represent transportation activities, and human participants such as drivers are responsible for accomplishing them. Manual activities are hence involved. There is an interest for automatic modifications during run-time (*i.e.*, reactive system).

It appears that characteristics inherent to such complex applications are still not adequately supported by current workflow technology. These characteristics are translated into a list of workflow constructs and real-time features. In this chapter, we report on this list. As an essential basis, we use established ideas such as (1) the basic workflow concepts and structures that exist behind workflow modeling (*e.g.*, activity, activity attributes, control/data flow and structural constructs in activity-based workflow modeling methodologies), (2) the concepts behind workflow enactment (*e.g.*, workflow/activity state, work-list) and (3) concepts related to the organizational configuration (*e.g.*, organizational model, organizational role).

In the following, we present in Section 6.1 the workflow technology enhancement process we devised. Section 6.2 exposes the list of enhanced workflow concepts and functionality. Sections 6.3 and 6.4 discuss these concepts and functionality, respectively, and expose investigated solutions for each of them.

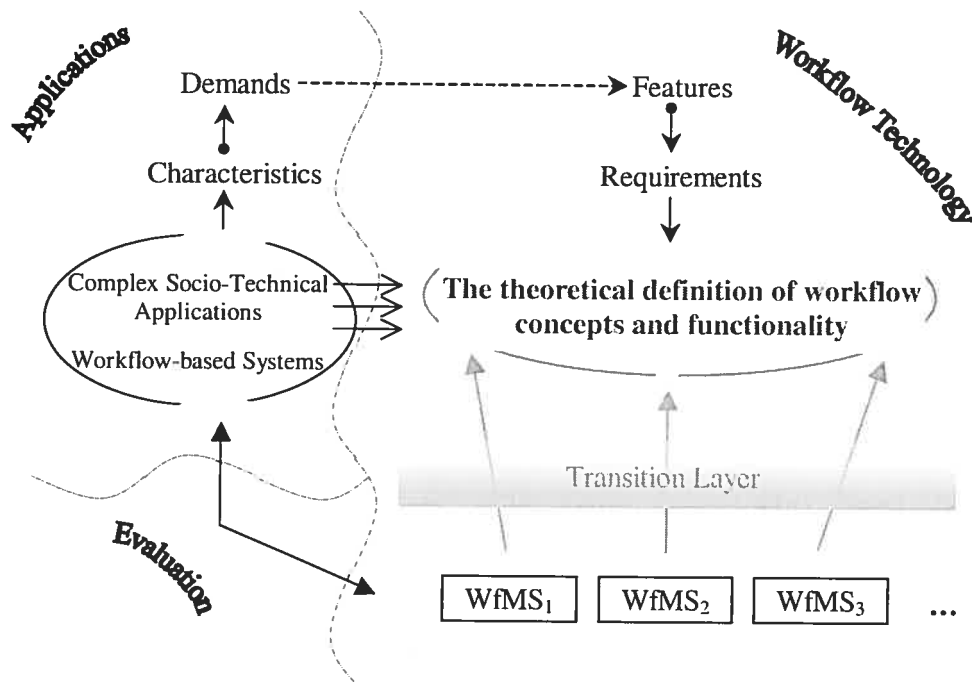
6.1 Workflow Technology Enhancement

Figure 6.1 shows the workflow technology enhancement process. Characteristics of complex socio-technical applications are translated into requirements towards workflow technology. This translation is indirectly accomplished passing through application demands and workflow technology features. The set of requirements identified can be divided into subsets, depending on the functionality already provided by the WfMSs that we are considering. The purpose of a subset is to complement existing WfMS features.

The *theoretical definition of workflow concepts and functionality layer* gathers the theoretical definition of the identified workflow technology requirements. A *transition layer* exists between the *theoretical definition layer* and WfMSs. The transition layer defines the implementation of the requirements subsets. In the best case, the WfMS provides a direct solution based on its offered features. Otherwise, a workaround solution can be implemented. In both cases, we rely on what is offered by the considered system and hence the *transition layer* is present to some degree. In the worst case, no solution is provided at all; an implementation needs to be directly integrated within the WfMS. Such an implementation may previously require some research work. In this case, the *transition layer* is absent.

Once this integration of new features is made, as a final stage of the workflow technology enhancement process, we may demonstrate/evaluate (1) the enhanced WfMS(s) (*e.g.*, ADEPT itself), and (2) the workflow-based systems we developed (*e.g.*, MTCT system, CONSENSUS system, ADEPT specific applications systems).

We recognize the fact that building a comprehensive list of requirements for WfMSs is an evolutionary task. This means that the system functionality is improved through the continuous assessment and revision of representative applications. In our case, the iterative investigation of complex applications stemming from typical, yet representative areas, the e-negotiations and the transportation domains, helped us to identify a set of enhanced workflow concepts and functionality. We believe that this list should be used to help improve what today's WfMSs offer in term of concepts and functionality.



- A WfMS_n offers a set of features to practically implement the theoretically defined workflow concepts and functionality.
- ... are translated into...
- ... imply...
- ... generate...
- The evaluation is made via the enhanced WfMSs and via the workflow-based systems.
- Applications give input to workflow technology.

Figure 6.1. Workflow Technology Enhancement

6.2 Enhanced Workflow Concepts and Functionality

Taking into account our experience with three specific WfMSs (IBM MQ Series Workflow, BEA's WLPI, and ADEPT) as well as our review of the literature related to current WfMSs, we specify for each concept/functionality identified, if its lack is a general problem \mathcal{G} (*i.e.*, it is not provided by most WfMSs) or if it is a WfMS-specific problem (\mathcal{A} for ADEPT-specific problem, \mathcal{M} for MQ Series-specific problem, and \mathcal{W} for WLPI-specific problem). We also specify whether it was possible or not to find a solution (\mathcal{DS}

for direct solution or WS for workaround solution) based on the features offered by ADEPT. Among the three WfMSs considered, we focus on ADEPT because it is the only one that already offers some functionality for ad-hoc changes. In the case where no solution was possible in ADEPT, we discuss either a possible implementation of the solution (I), or theoretical ideas (TI) for a possible support of the concept/functionality.

Here is the list of requirements for workflow technology.

Enhanced workflow concepts (cf. Section 6.3):

- The activity template concept [\mathcal{M} , \mathcal{W} , \mathcal{DS} , 6.3.1]
- The template classification [\mathcal{A} , WS , 6.3.2]
- The activity temporal aspects [\mathcal{G} , \mathcal{DS} , 6.3.3]
 - The activity starting/finishing time [\mathcal{A} , WS , 6.3.3.1]
 - The activity duration [\mathcal{DS} , 6.3.3.2]
 - The activity warm-up time concept [\mathcal{G} , WS , 6.3.3.3]

Enhanced functionality applied at the workflow instance level (cf. Section 6.4):

- The dynamic insertion of an activity [\mathcal{G} , \mathcal{DS} , 6.4.1]
 - The dynamic insertion of a new activity instance [\mathcal{A} , WS , 6.4.1.1]
 - The dynamic insertion of a block of activities [\mathcal{G} , WS , 6.4.1.2]
- The dynamic deletion of an activity [\mathcal{G} , \mathcal{DS} , 6.4.2]
 - The interruption of an activity execution while preserving its context [\mathcal{G} , TI , 6.4.2.1]
- The dynamic move of an activity [\mathcal{G} , WS , 6.4.3]
- The dynamic modification of activity attributes [\mathcal{G} , I , 6.4.4]
 - The dynamic insertion/setting/updating of input attributes [\mathcal{G} , I , 6.4.4.1]
 - The dynamic deletion of input/output attributes [\mathcal{G} , I , 6.4.4.2]
 - The dynamic (re-)assignment of activities to a participant [\mathcal{A} , I , 6.4.4.3]
 - The dynamic setting/updating of time attributes [\mathcal{G} , I , 6.4.4.4]

- The dynamic management of work-lists [*G*, *I*, 6.4.5]
- The automatic/manual modification of workflow instances [*G*, *I*, 6.4.6]

This list shows only one requirement (the interruption of an activity execution while preserving its context) for which theoretical ideas (*TI*) are elaborated. In the following, this requirement is studied in depth. Detailed formal definitions are given. Based on these definitions, a general correctness criterion ensuring what we refer to as the *safe* interruption of a running activity is specified.

Other concepts and functionality from this list, namely the ones marked with *DS*, *WS*, and *I*, are discussed in less formal detail in what follows.

6.3 Enhanced Workflow Concepts

In this section, we discuss concepts identified in Section 6.2. We also report on direct or workaround solutions we investigated to support each of these concepts.

6.3.1 The Activity Template Concept

In order to introduce a standard way for defining activities, it is useful to devise a set of activity templates related to the studied application. Activity templates are standalone activities that can be designed without being part of any workflow definition. They are defined for prospective use during the scheduling of the different activities in a workflow model or in a workflow instance. Each activity template consists of a task with three types of attributes:

- *Input/output attributes*, which specify the information needed to accomplish a task (input attributes) or the information produced by the task (output attributes). This captures the semantic aspects of the task.
- *Assignment attributes*, which specify the actor(s)/role(s) responsible of accomplishing (or allowed to accomplish) this task. This is mainly used by the system to let the task appear in the appropriate work-list in case of a human actor, or to call the appropriate application/program/software module in case of a software actor.

- *Time attributes*, which specify the (min/max) duration of the task, its (earliest/latest) starting time, and its related warm-up time.

Activity templates should be accessible from the execution phase mainly to allow the dynamic insertion of a new activity instance based on their definition. Refer to Section 6.4.1.1 for more details regarding this issue.

In current WfMSs, the activity template concept is not defined. Moreover, even if it was defined, we cannot take advantage of this concept since no dynamic insertion is allowed during run-time in most WfMSs.

In ADEPT, this concept is defined. However, the prototype of ADEPT does not allow the dynamic insertion of an activity based on activity templates.

6.3.2 The Template Classification

Since a WfMS is usually used in the context of different applications, we found it necessary to define the “template classification”. This concept assigns a specific “category” (*i.e.*, application domain) to a set of activity templates and workflow models. It fosters the more focused selection of a specific activity template or (sub-) workflow: interest for the modeling and for the execution phase. As an example, activities such as: “attach container to vehicle”, “move container to origin location”, “load container”, etc. belong to the transportation domain and can rarely be useful for the design/adaptation of workflow models/instances in other domains. The classification concept also facilitates the extraction of resources in systems such as the MTCT system. It facilitates the implementation of the Resource Extraction Client (cf. Section 7.2).

Commercial WfMSs such as IBM MQ Series Workflow and BEA’s WLPI, usually propose a tree structure for workflows. This structure allows to define categories that gather workflow models. Example of categories:

- “Banking” gathering workflow models such as “Credit Request” and “Savings”.
- “Sales and Underwriting” gathering “Life Insurance”, “Medical Insurance”, etc.

A WfMS's user has usually authorizations for specific categories. A filter with "category" as criterion is used to retrieve, when necessary, workflow templates that belong to a specific category.

The classification of *activity templates* according to the application they belong to appears promising as well. Usually, in current WfMSs, the first level in a tree structure defines the categories; the second level defines the workflow models and the activities are gathered within the third level. However, we would also like to view activity templates not related to a specific workflow model, at the same level as the workflow models.

The workaround solution we adopted in ADEPT is the following. When defining the workflow templates and the activity templates related to a specific application, we save them with a specific prefix. *E.g.*, all activity templates related to our Multi-Transfer Container Transportation application were saved with the "MTCT" prefix. This facilitates filtering thereafter.

6.3.3 The Activity Temporal Aspects

Activity time attributes such as the duration and the starting/finishing time of an activity are discussed in the literature. The ADEPT project treats these two aspects in detail [DRK00]. In Sections 6.3.3.1 and 6.3.3.2, we discuss the activity starting/finishing time and the activity duration, respectively. A differentiation should however be made between (1) the *planned starting time* of an activity, (2) the *activation time* of an activity (*i.e.*, when the activity is due, taking into account the control flow), and (3) its *assignment time* to a work-list. Usually, within current WfMSs an activity is assigned to a work-list as soon as it is due within the flow. However, workflow participants should not be surprised by activities, and they should know in advance about the next activity to carry out. Hence, the assignment time of an activity to a work-list should depend on the planned starting time of the activity and on the necessary warm-up time. Eder *et al.* tackle a similar problem by working on future personal schedules [EPG+03]. Their work is motivated by the need to provide early information about future tasks (*i.e.*, forecasting of tasks). Their approach is based on probabilistic time management. The warm-up time concept will be presented in Section 6.3.3.3.

6.3.3.1 The Activity Starting/Finishing Time

We distinguish between absolute dates (*i.e.*, fixed calendar dates) and dependant dates between activities. Absolute dates are referred to as *external dependencies* [RXZ04]. An external dependency is caused by parameters external to the system (*e.g.*, time): an activity “a” can enter a specific state “s” only if a certain condition “c” is satisfied where the parameters in “c” are external to the workflow. An example of an external dependency is that activity “a” of workflow “W” can start at “9:00 am GMT”. Dependant dates are referred to as *time dependencies between activities* [DRK00]. Time edges are introduced in [DRK00] to connect two activities and define a minimal or maximal time distance between them. Time relationships could be: completion/start, start/start, completion/completion, and start/completion. An example of dependant date is: activity “a” must be completed two days before activity “b” starts.

In IBM MQ Series Workflow, the starting/finishing time of an activity can be defined within the activity by specifying respectively its “START” condition and its “EXIT” condition.

In BEA’s WLPI, a distinction is made between synchronous actions and asynchronous actions. One of the asynchronous actions is the “set task due date” used to specify the activity starting time. “Time event” is another asynchronous action that can be used to specify the finishing time.

The definition of dependent dates between activities in IBM MQ Series Workflow and in BEA’s WLPI is not straightforward; by contrast, this constitutes one of the strengths of ADEPT.

However, the ADEPT prototype does not allow the specification of an absolute date for the activities’ starting/finishing time. The workaround solution we found is as follows:

We use the “time edge” concept and we define a minimum and a maximum time distance between the “start” activity (S) and each of the activities (A). The earliest and the latest starting time of A (EST_A/LST_A) are specified taking into account the real starting time of S. Once the execution of S is completed, its real starting time ST_S is then known.

The minimum time distance and the maximum time distance of the “time edge” between S and A are respectively equal to $EST_A - ST_S$ and $LST_A - ST_S$.

6.3.3.2 The Activity Duration

This is not a problem in ADEPT. The duration is directly specified within the activity. In MQ Series and WLPI, the duration of the activity can be defined within conditions (“EXIT” condition in MQ Series and “decision” in WLPI) using a date function format in a specific expression. Example: *when* the expression that compares “the current date/time” with the sum of “the activity real starting time” and “the activity duration” is evaluated to TRUE, *then* continue the execution with the next activity taking into account the control flow.

6.3.3.3 The Activity Warm-Up Time Concept – Integration of Preparation Activities

The introduction of [EPG+03] motivates convincingly the need for providing early information about upcoming activities:

In the execution of workflows, workflow participants are typically “surprised” by the activities they should perform, surprised in the sense that they find these activities in their to-do-lists when these activities are ready, i.e. all preceding activities are finished. Information about upcoming activities would be much earlier available in the workflow system. For an example, when the first activity of a sequence is ready, the succeeding activities will be ready soon. Current workflow systems do not make use of this information and do not forward this information to the participants depriving them of the possibility of planning their work ahead. [...]

In current WfMSs, activities are assigned to work-lists in-order regarding the control flow. We would like to find a technique for allowing activities to be dispatched out-of-order when (1) the difference between the starting time of the activity and the time required to get prepared to this activity (*i.e.*, the warm-up time WUT) corresponds to the current time, but also when (2) the activity has a high probability of being reached by the control flow. The question that may arise is the following: how to measure/predict this probability? In [EPG+03], probabilities are taken for granted. However, in real-world applications, probabilities can rarely be fixed in advance.

The idea behind the “out-of-order” handling of activities stems from the dynamic scheduling (*i.e.*, looking-ahead, pre-fetching) approach sometimes based on prediction, adopted by processor technologies. In fact, the dynamic scheduling allows instructions to execute out-of-order regarding the instructions order within a program, when there are sufficient resources and no data dependencies (*e.g.*, structural hazards, WAW, WAR).

A way to support the warm-up time concept is to introduce it at the workflow modelling level by integrating “preparation activities”. Preparation activities are defined for work-list management purpose (*i.e.*, notification purpose) but also for resource management purpose. Indeed, A specific activity may require some preparation work done in advance. This preparation may ask for some resource reservation.

When early information about a future activity “Act” needs to be provided to the (human) participant responsible of the execution of “Act”, a preparation activity “Act_{prep}” related to “Act” is scheduled within the workflow. During run-time, “Act_{prep}” is an automatic activity that appears at the right time (*e.g.*, at time: “EST_{Act} - WUT_{Act}”) in the work-list of the participant responsible of “Act”. “Act” could be executed as soon as EST_{Act} is reached: “Act_{prep}” leaves the work-list, and “Act” appears instead.

We include preparation activities within the formal definition of WSM-Nets [RRD04c], already introduced in Section 2.2.3.

Definition 6.1 (Extended WSM-Net – Preparation Activities) A tuple $S = (N, N_{prep}, NT, CtrlE, pa)$ is called an extended WSM-Net if the following holds:

- N is a set of activities
- $NT: N \mapsto \{StartFlow, EndFlow, Activity, PreparationActivity, AndSplit, AndJoin, XorSplit, XorJoin, StartLoop, EndLoop\}$
 NT assigns to each node of the extended WSM-Net a respective node type.
- $N_{prep} \subset N$ is the set of preparation activities.
 $\forall n \in N_{prep}, NT(n) = PreparationActivity$
- $CtrlE \subset N \times N$ is a precedence relation
- $pa: (N \setminus N_{prep}) \mapsto N_{prep} \cup \{UNDEFINED\}$
 pa assigns to each activity of $(N \setminus N_{prep})$ either a specific preparation activity from N_{prep} or $UNDEFINED$.
 $\forall b \in (N \setminus N_{prep}) \mid pa(b) = b_{prep} \in N_{prep}, b_{prep} \in Pred^*(S, b)$ holds.
 $Pred^*(S, b)$ denotes all direct and indirect predecessors of activity b .

Taking into account Definition 6.1, a workflow such as the one shown in Figure 6.2(c) can be modelled.

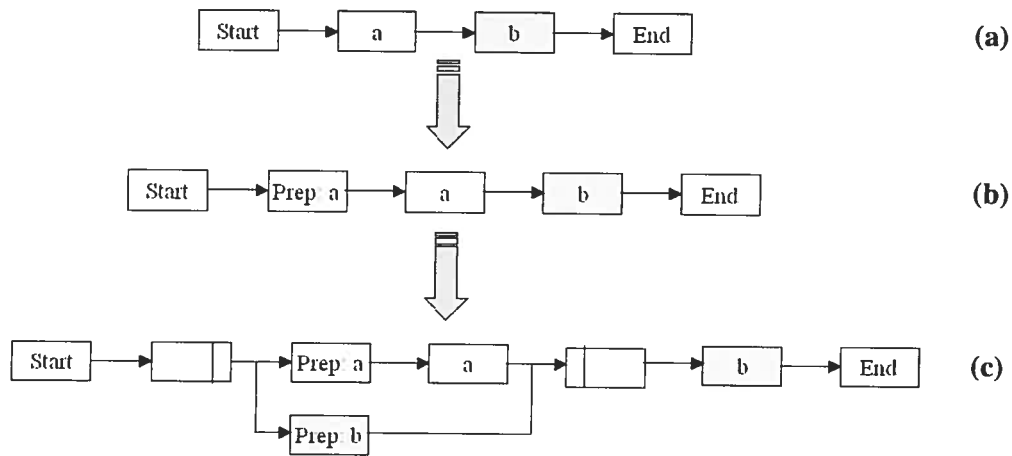


Figure 6.2. Integrating “Preparation Activities” to a Workflow. (a) A Workflow with Two Activities (“a” and “b”) Defined in Sequence, (b) Integrating “Prep: a”, (c) Integrating “Prep: b”

Given an initial schema S with activities defined in sequence as the workflow in Figures 6.2(a), if a “preparation activity” is to be defined for an activity “a”, which is the first activity to be executed in the workflow, it will be inserted in sequence with “a”, previous to “a” (Figure 6.2(b)). In general, a preparation activity “Prep: b” defined for an activity “b” is to be inserted in parallel with all the activities that precede “b” (Figure 6.2(c)).

However, other modelling structures more complicated than the simple “sequence” of activities, *e.g.*, concurrency or parallel branching, synchronization, selection or conditional branching, iteration, may be involved in a schema S . A way that helps studying each of these structures with the purpose of integrating “preparation activities” is to proceed as follows:

A schema S involving a specific structure is designed. An instance I_S on S is created, and the insertion of a preparation activity b_{prep} is made by specifying its “before nodes” and its “after nodes”. The “after nodes” of b_{prep} consist of b and of all its successors. Let AN the set that gathers these “after nodes”. The set of “before nodes” (BN) is defined as $N \setminus AN$ (the complement of AN in N).

We used ADEPT to insert “preparation activities” during run-time, and to discover where these activities need to be scheduled within a workflow taking into account a specific structure. We retain the following remarks:

- (1) “Preparation activities” are inserted the same way in a “parallel” and in a “conditional” branching; but in the latter, the work-list requires more sophisticated management. Indeed, activities in conditional branches are announced. Once a decision is taken regarding the branch to be followed, preparation activities corresponding to the activities that will not be executed should be removed from the work-list(s). An alert stating the “non-execution of these activities anymore” may also be generated.
- (2) When a loop is defined within a workflow, the activities defined in the loop are announced just once as “repetitive activities” and this is made before accessing the loop.

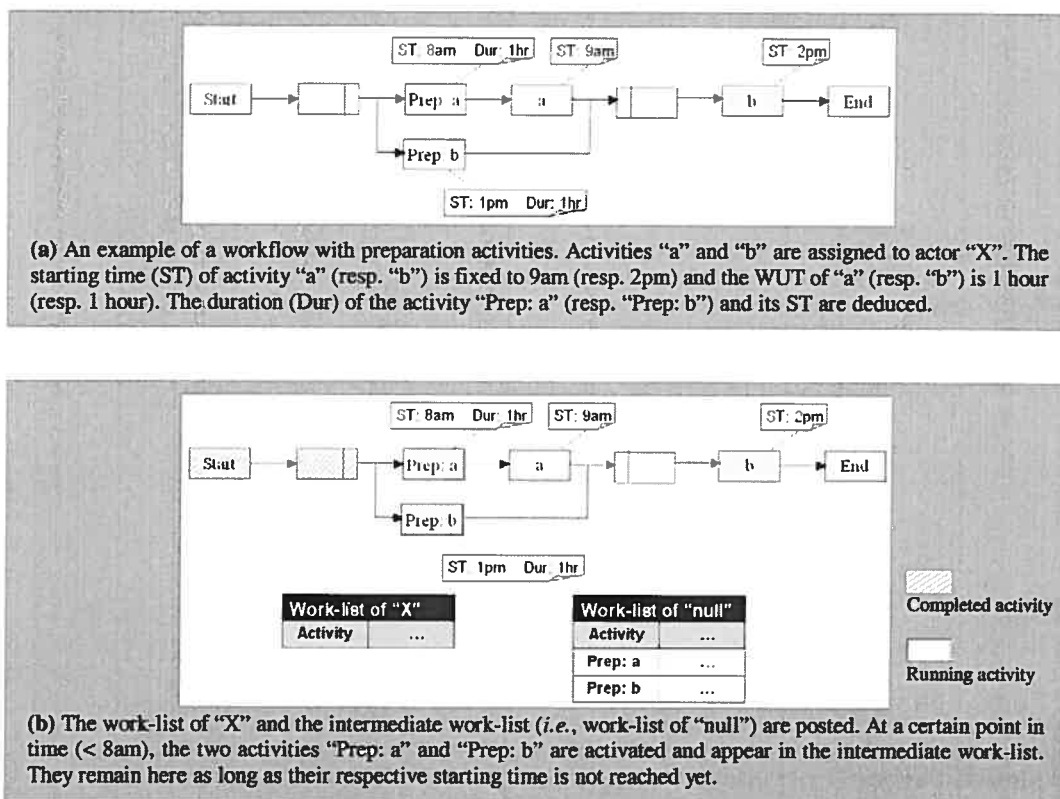
We observe two obvious disadvantages with the integration of “preparation activities”:

- (1) Workflow participants are notified, via their work-list, about all the activities of the workflow as soon as the “start” node is completed. What we really need is a just-in-time notification.
- (2) If each activity in the initial workflow needs to be associated with a preparation activity, the number of (non-empty) activities in the workflow doubles. Moreover, we will be dealing with at least as many parallel branches as the number of activities requiring a preparation activity. The workflow becomes complicated to understand. A dynamic modification of a workflow instance becomes complicated to manage since an activity has a special relation with its preparation activity. Finally, we may also think about the loss of the workflow general view since the preparation activities are not a genuine part of the business process.

In the following, we present and discuss how each of these two disadvantages can be dealt with.

6.3.3.3.1 Dealing with the First Disadvantage of the “Preparation Activities” Approach: Introduction of an Intermediate Work-list with a Listener Process

The just-in-time notification is possible when we manage the work-list in a way that its update depends not only on the control flow but also on the “starting time” of the preparation activities. This can be resolved by a suitable implementation. Once an activity is due taking into account the control flow, it is assigned to an intermediate work-list. A listener process on this work-list should be implemented to detect when the “starting time” of a work-item is reached – so that this work-item appears in the appropriate work-list. The mechanism of an intermediate work-list with a listener process is explained in Figure 6.3.



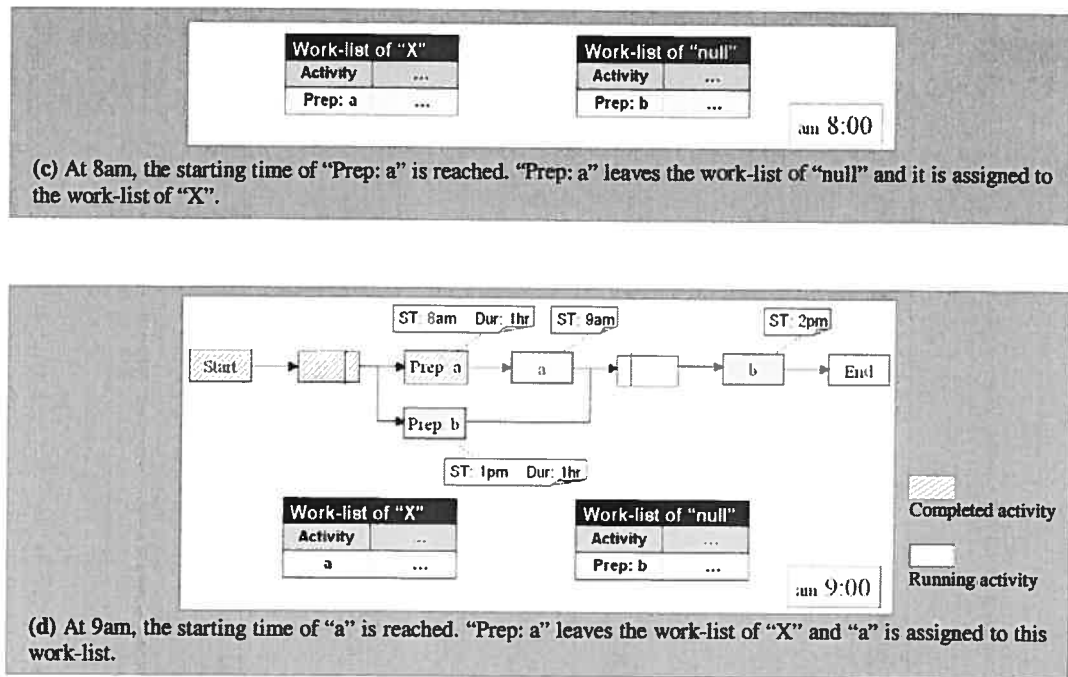


Figure 6.3. The Mechanism of an Intermediate Work-list with a Listener Process

6.3.3.3.2 Dealing with the Second Disadvantage of the "Preparation Activities" Approach: Defining Preparation Activities in the Background (First Solution)

Given a schema S (cf. Figure 6.4), let S' the schema that corresponds to S and that incorporates warm-up times for activities that need to be forecast in advance. A label specifying the WUT can be associated with each of these activities. The structure of S' will still show the genuine business process, *i.e.*, the same structure as S . In brief, the "warm-up time concept" should be defined as a construct related to an activity in the same way other similar constructs are usually defined (*e.g.*, EST/LST, EFT/LFT).

We associate S' , considered as a high-level schema, with a low-level schema S'_{II} . S'_{II} will integrate the preparation activities in parallel branches; preparation activities will however be kept out of sight of the WfMS's user controlling and monitoring the workflow (*i.e.*, they are in the background).

A transformation function needs to be defined: $S \rightarrow S'_{II} \rightarrow S'$

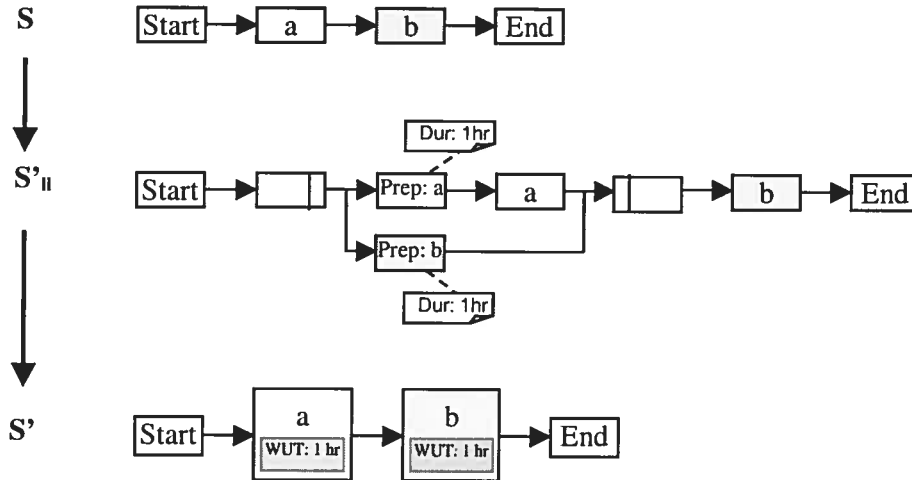


Figure 6.4. Sending “Preparation Activities” to the Background

6.3.3.3 Dealing with the Second Disadvantage of the “Preparation Activities” Approach: A Layered Workflow Architecture (Second Solution)

Instead of incorporating the “preparation activities” directly within the workflow, we may think about gathering all the preparation activities related to a specific workflow within a separated (sub-)workflow. We may devise a “layered workflow architecture” (Figure 6.5). The challenge here is how to define links between an activity and its associated preparation activity so that a modification brought to an activity is reflected to its associated preparation activity.

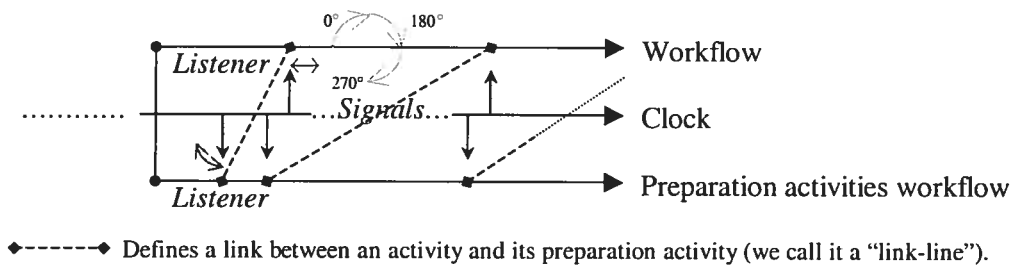
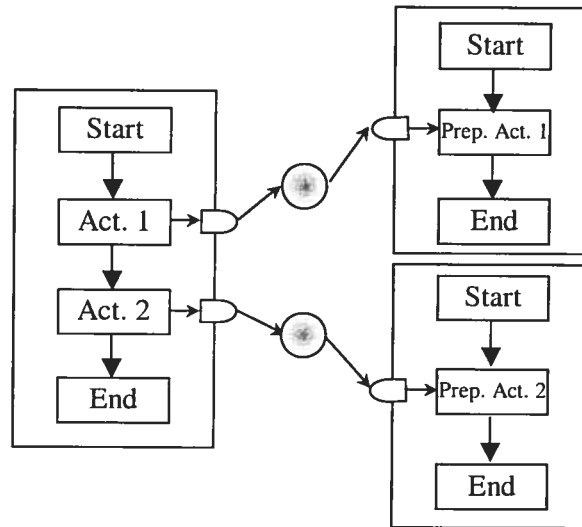


Figure 6.5. Explanation of the Layered Workflow Architecture for the Support of the WUT Concept

For a workflow activity “Act”, we specify a “starting time” (*e.g.*, an absolute date) and a WUT (*e.g.*, in minutes). For a preparation activity Act_{prep} , we specify a “starting time” that corresponds to “ $EST_{Act} - WUT_{Act}$ ”. A modification brought to the “starting time” of Act involves a shift of the link-line to the right or to the left (\leftrightarrow). A modification brought to the WUT of Act involves a rotation of the link-line (\curvearrowright). The rotation angle lies between 0° and 90° , and the angle of the link-line should always lie between 270° and 360° . A link-line angle of 270° means the WUT is equal to 0 minutes. This can apply to automatic activities where no human actors are involved. Nevertheless, the link-line angle can never be 360° . The “ 360° ” corresponds to a horizontal line, meaning that the WUT is equal to ∞ . As a consequence, the “ 90° ” rotation angle is also impossible. The really used intervals for the link-line angle and for the rotation angle are respectively a sub-interval of $[270^\circ, 360^\circ]$ and a sub-interval of $[0^\circ, 90^\circ]$.

When a modification is brought to the ST or to the WUT of an activity, a message should be sent from this activity (the sender) to its related preparation activity (the receiver). In [ABE+00], the authors introduce *performatives* used in the context of proclats. The latter are defined as lightweight workflow processes equipped with communication channels. Performatives are used to specify communication and collaboration between proclats. They have attributes such as “channel” (the medium used to exchange the performative), “sender” (the identifier of the proclat creating the performer), “receiver” (the identifier of the proclat receiving the performer), “action” (the type of the performative), and “content” (the actual information that is being exchanged). In our context, the workflow and its corresponding preparation activities workflows can be seen as proclats (Figure 6.6). Performatives are defined to allow a one-way communication between “activities” and their related “preparation activities”. The action attribute in our case can be viewed as a “notification”: an activity notifies the corresponding preparation activity about a modification brought to its ST or to its WUT. The content attribute specifies the new ST or the new WUT.



★ A notification action

Figure 6.6. The “Proclat” Idea for the Support of the WUT Concept

6.3.3.3.4 Extension of the Warm-Up Time Concept – An Overview

The WUT concept can be extended to a notification system where alerts can be triggered *asynchronously* to handle an event or exception. Chiu *et al.* for instance study the urgency requirements for alert routing in a healthcare application, employing mobile technologies, and healthcare partner process integrations [CKW+04]. In their approach, they propose to separate user alerts from user sessions with the WfMS. Online users are alerted through ICQ (I seek you) with the task summary and reply URL as the message content. If the user is not on-line, or does not reply within a pre-defined period, the WfMS sends the alert by email. At the same time, another alert may be sent via SMS (Short Message) to the user’s mobile phone. Whatever the alert channel has been, the user needs not connect to WfMS on the same device, or even on the same platform. For example, after receiving a SMS alert, the user may use her handset to connect to the WfMS via WAP, or she may reply with an SMS message. Alternatively, the user may find a PC (Personal Computer) with Internet connection or use her PDA (Personal Digital Assistant) to connect to the WfMS.

6.4 Enhanced Workflow Functionality Applied at the Workflow Instance Level

In this section, we discuss functionality identified in Section 6.2. We report on direct or workaround solutions to cover the dynamic insertion, deletion, and move of an activity. This will comprise theoretical work ensuring the *safe* interruption of an activity execution. Then, the implementation within the system of the dynamic modification of activity attributes is discussed. As a consequence to the dynamic modification of attributes, the adaptation of work-lists is considered. Each of the functionalities identified can be applied either manually or automatically. This will be finally discussed.

6.4.1 The Dynamic Insertion of an Activity

This insertion should be based on previously defined activity templates and sub-workflows. During insertion, temporal constraints should be respected and input attributes of the inserted activity should be linked to newly generated data elements. This is discussed in [RD98]. The dynamic insertion of an activity could be extended to the dynamic insertion of a sub-workflow. As an example from the MTCT application, the sequence of the two activities “detach container from vehicle” and “attach container to vehicle” should be inserted each time a container needs to be transferred from one vehicle to another.

Commercial WfMSs, such as WLPI, allow re-executing an activity already completed. However, this does not correspond to the dynamic insertion of an activity since no structural modification is possible.

This functionality is defined in ADEPT, however the ADEPT prototype exposes the main problem discussed in Section 6.4.1.1. The dynamic insertion of a sub-workflow (Section 6.4.1.2) is not possible in ADEPT.

6.4.1.1 The Dynamic Insertion of a New Activity Instance

It is not clear in the literature of ADEPT if the activity to be inserted has to be chosen from the set of activities of workflow instances, or rather from the set of activity tem-

plates. The ADEPT prototype allows only the first option. Since this is possible, it should not be a problem to support in a workaround manner the second option:

When a specific activity template is chosen to be inserted within a workflow instance, this activity can be automatically defined within a workflow model, between a “begin” node and an “end” node; and an instance of this workflow can be created. The problem of inserting an activity based on a previously defined activity template amounts to the same thing as the problem of inserting an activity instance.

The WAW (write after write) problem should not exist. That’s why the parameters of two activity instances defined from the same activity template within a specific workflow, should be linked to distinct data elements. In the MTCT system, two activities A1 and A2 defined from the same activity template are usually distinct within a specific workflow. As an example, the activity “Move vehicle *v* from location *a* to location *b*” may be present twice in the same workflow. The three variables *v*, *a*, and *b* are linked to three data elements. Data elements in A1 should be different from data elements in A2.

A refined issue is to allow the insertion of a new activity by defining it from scratch. This is known by the “on-the-fly” editing [Sie96] where new activities can be defined at run-time. This means that the user is supposed to deal with a graphical workflow editor during run-time. A similar idea is evoked in [KBB98]. The authors discuss partial execution that supports creating and executing workflows and workflow fragments “on-the-fly” as they are needed – or as the information becomes available –, rather than requiring the entire workflow to be specified ahead of time.

6.4.1.2 The Dynamic Insertion of a Block of Activities

There is no solution defined within current WfMSs. The workaround solution we developed can be captured by the five steps described below. This solution is based on the fact that the dynamic insertion of an activity is possible. Refer to Figure 6.7.

Step 1: Take the first activity in the given sub-workflow and insert it between the BEFORE nodes and one of the AFTER nodes AN1. Based on a *symmetrical control structure* as used in [RD98], Figure 6.8 shows the effective structure of the workflow after

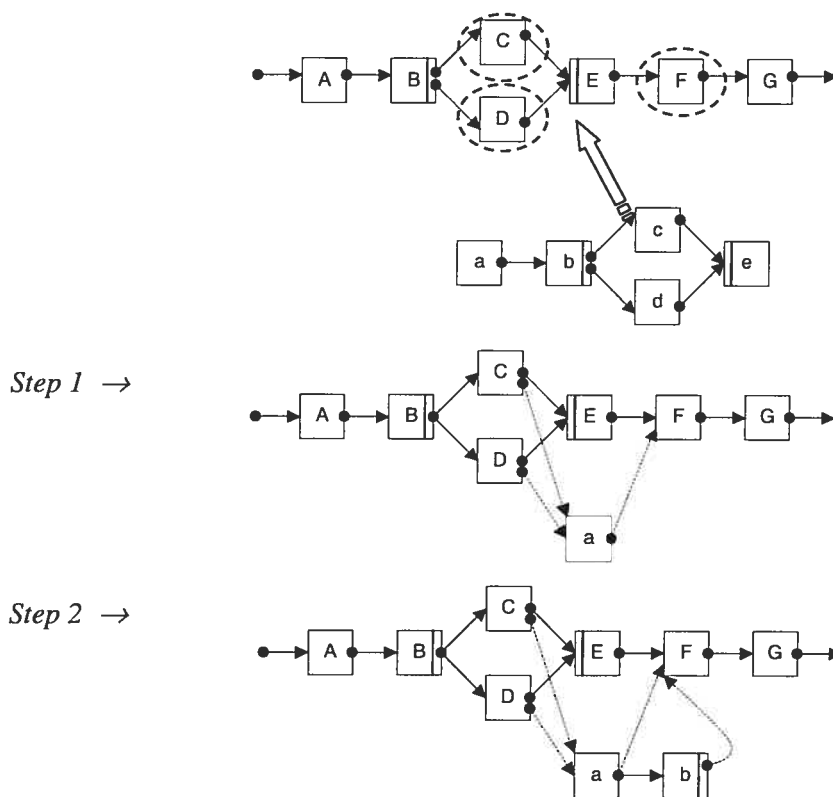
inserting the first activity. Empty nodes and sync-edges are introduced as a consequence to the insertion. Reduction rules are applied.

Step 2: Take one next activity instance in the given sub-workflow with no predecessors that are not inserted yet (except for loops there must always be one existing) and insert it between its already inserted predecessors and AN1. Figure 6.9 shows the effective structure of the workflow after inserting the second activity.

Step 3: Continue like this until you reach the last activity in the provided sub-workflow.

Step 4: Insert this last activity between its already inserted predecessors and all AFTER nodes.

Step 5: Delete all edges between the AFTER node AN1 and all newly inserted nodes except for the last one.



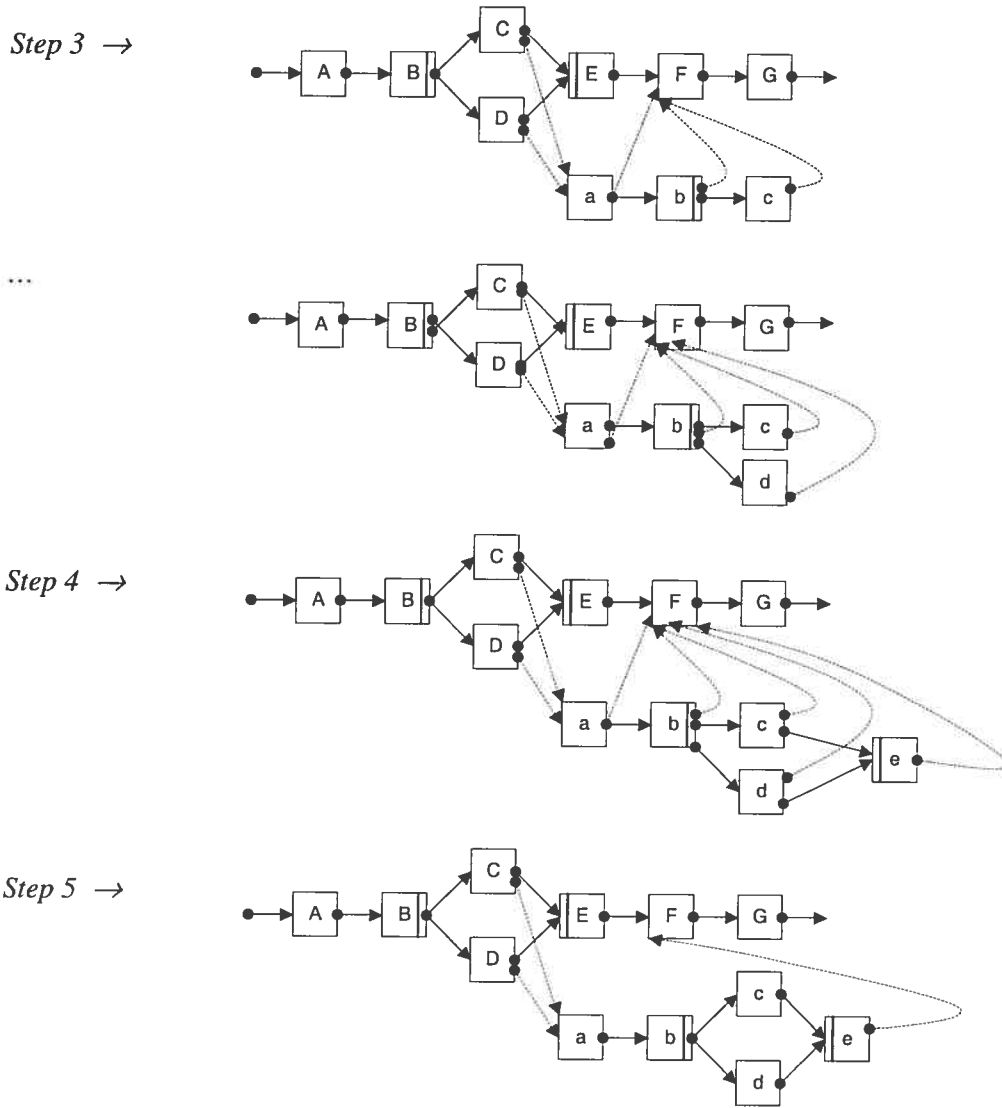


Figure 6.7. Steps for the Dynamic Insertion of a Sub-Workflow

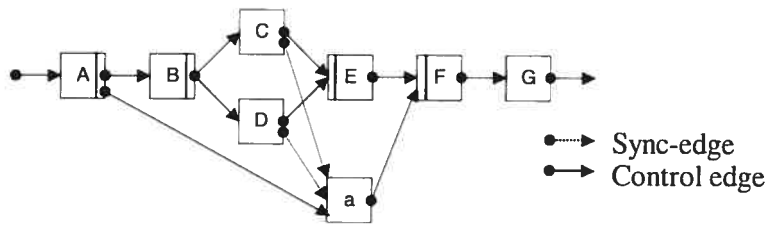


Figure 6.8. Valid Structure of the Workflow Resulting from Step 1

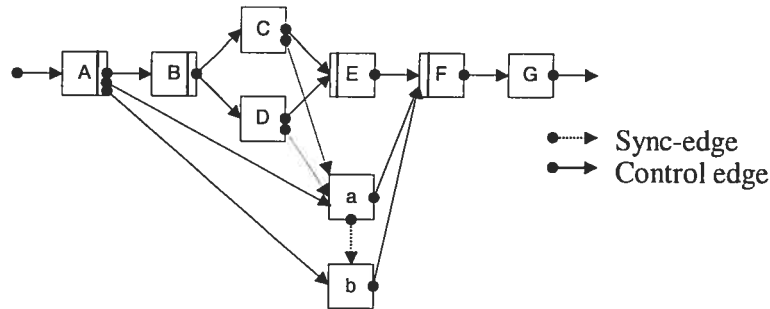


Figure 6.9. Valid Structure of the Workflow Resulting from Step 2

Suppose that the sub-workflow shown in Figure 6.10 is to be inserted between {C, D} and {F} (cf. Figure 6.7). Is the “End loop” node considered a “predecessor” to the “empty” node, to “b” and to “c”? The first time “b” and “c” are executed it is not. When the loop is executed more than once, the “End loop” node becomes a “predecessor” to the “empty” node, to “b” and to “c”.

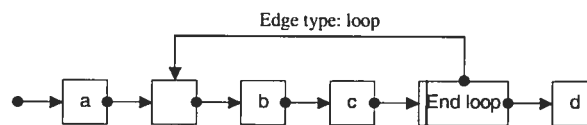


Figure 6.10. Example of a Sub-Workflow Including a Loop

The workaround solution just exposed (inserting the activities of a sub-workflow *one by one*) has at least four problems:

- (1) If not automated, this solution presents too much (manual) manipulations/interactions with the system, which is an error-prone solution.
- (2) The activities insertion order is very important. *E.g.*, an activity input parameter should be previously provided as an output parameter of a predecessor activity.
- (3) This solution has been applied to sub-workflows with sequential activities only; the insertion of complex modelling structures is problematic. Indeed, in the example of figure 6.7, the insertion of an And-split and an And-join, as well as of activities between these two specific nodes is not obvious.

- (4) Step 5 (deletion of un-useful edges) is complicated to accomplish because of the introduction of sync-edges and empty nodes. This step can however be skipped.

An appropriate solution for the insertion of a sub-workflow is to consider the latter as a whole with input and output data. This was not addressed yet in the literature, but it could fit well in the approach proposed in [RD98] for the insertion of a single activity. Indeed, in this approach, the correctness of the flow of data is verified to ensure a syntactically correct schema.

6.4.2 The Dynamic Deletion of an Activity

ADEPT provides this functionality. In other WfMSs, the activity instance can be “force finished” (MQ Series Workflow) or it can be marked as “done” without execution (WLPI). Marking an activity instance as done without executing it can remedy in certain cases to the dynamic deletion of this activity. This is true when for instance we do not expect output attributes from the activity instance marked as done.

In the following, we discuss a sophisticated issue in conjunction with the dynamic deletion of an activity instance, that is, the safe interruption of an activity in a running state.

6.4.2.1 The Interruption of an Activity Execution While Preserving its Context

An activity interruption triggered by the appearance of unexpected events in environment such as the MTCT cannot be avoided. As an example, technical problems of vehicles, traffic jams or forced rerouting may appear at any time while vehicle V is on the road moving goods between origin location O and destination location D . This usually leads to the interruption of the “move V from O to D ” activity. In such a situation, an adaptation of an already planned flow of activities for the satisfaction of a customer request is required; this adaptation should take into account the current context of the interrupted activity. The new transportation solution may propose to send a new vehicle V' to the current position of V or to change the already planned route leading to D . In both cases, the current position of V should be known such that an appropriate new solution can be proposed.

Preserving the context of an interrupted activity consists of saving data, which is produced by or associated with this activity. This must be done at the right time, *e.g.*, as soon as the data become available or relevant. At this point, it is important to have a closer look at the granularity of work unit descriptions. Usually, a business process activity can be further subdivided into *atomic steps* corresponding to basic working units or to data provision services. Basic working units are either directly coded within application programs or can be worked on manually by people. Distinguishing between activities and atomic steps is useful for the following reasons: Atomic steps are not managed within work-lists like activities are. This contributes to better system performance since the cost for managing and updating work-lists decrease. Furthermore, this approach offers more flexibility to users (if desired) since they can choose the order in which they want to work on atomic steps. The distinction between activities and atomic steps finally leads to the following basic considerations.

We distinguish between a *continuous* and a *discrete data update by activities*. The “move V from O to D” activity is an example of an activity continuously updating the “V current position” data element by a GPS system. An example of an activity discretely updating data is even more obvious in process-oriented applications. We may think about the activity “fill in a form” with many sections, each one asking for information (*i.e.*, data) related to a specific topic. The information becomes relevant, and therefore may be kept in the system, only after the completion of a specific section. Filling in a section could be seen as working on a particular *atomic step*.

We highlight the fact that a process activity may apply both updating kinds: it may discretely update a particular data element d_1 and continuously update another data element d_2 . Moreover, data elements may be discretely updated by a specific activity n_1 and be continuously updated by another activity n_2 . As an example, an activity “monitor patient” in a simplified medical treatment process such as the one introduced in Chapter 2 and depicted again in Figure 6.12, may ask to measure twice a day the “patient temperature” and to continuously control the “patient heart electric signals”. On the other hand, the “patient temperature” may be permanently controlled in case of high fever within

activity “monitor patient” while it may be measured twice a day after operation within activity “aftercare”.

Data continuously or discretely updated by activities may be only relevant for the specifically studied application (*e.g.*, the vehicle “current position” for the container transportation process depicted in Figure 6.13) or they may be relevant for process execution as well. In the latter case, these data are consumed by process activities and therefore have to be supplied by preceding activities. At the occurrence of exceptional situations, it may appear that mandatory process relevant data will not be available at the time an activity is invoked. Depending on the application context and the kind of data, it may be possible to provide the missing data by *data provision services*, which are to be executed before the task associated with the respective activity is handled.

We distinguish between *exclusive application data* and *process relevant data*. Note that an exclusive application data may become process relevant when a failure occurs. In the transportation application, an example of process relevant data would be the “container temperature” (continuously) measured during a “move V from O to D” activity and relevant for a “Report to customer” activity within the same process. Reporting on the container temperature would inform the customer whether the transported goods (*e.g.*, foods) were or were not continuously preserved under the appropriate temperature. The “V current position” is an example of exclusive application data since it is relevant for the application, in particular for the optimization module of the application (*cf.* Chapter 7), but not for the business process management system. If, however, a road traffic problem occurs, the “current position” of V may become relevant for the process as well; *i.e.*, the origin location O’ of a newly proposed activity “move V from O’ to D” changing the already planned route leading to D, would correspond to “current position” of V (*i.e.*, O’ := “V current position”).

Figure 6.11 shows a data classification scheme in the context of business processes. This classification puts the frequency of updating activity data and the relevance of these data into relation. Within these two dimensions of data, we respectively differentiate between:

- a continuously and a discretely updated data, and
- an exclusive application data and a process relevant data

Taking into account this classification, and knowing that exceptions stemming from the application environment cannot be avoided and generally appear during activity performance, it would be a challenge not to lose available data already produced by the activity that will be inevitably interrupted or deleted. In order to formally specify the correctness criterion for interrupting running activities while preserving their context, formal definitions of requisite foundations for this specification are indispensable.

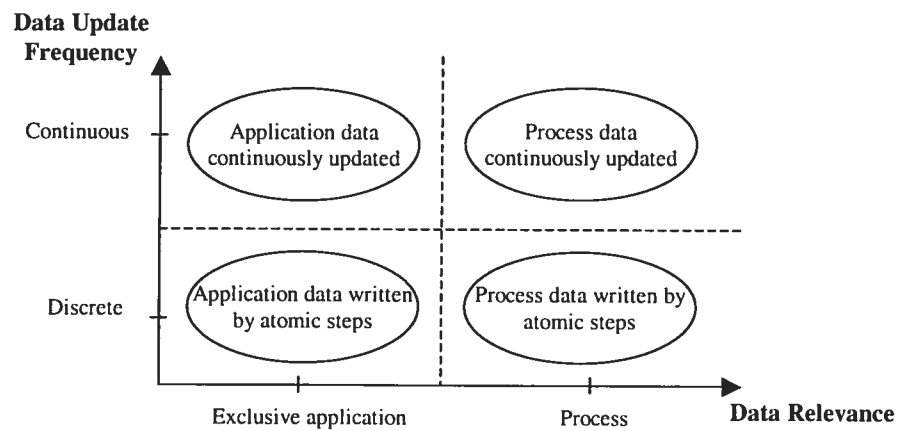


Figure 6.11. Data Classification Scheme

In the following, we first define such foundations (Section 6.4.2.1.1). Then, we introduce a general correctness criterion ensuring a safe interruption of a running activity (Section 6.4.2.1.2). Finally, the adopted approach is discussed in Section 6.4.2.1.3.

6.4.2.1.1 Formal Framework

In order to precisely define the different kinds of data and update frequencies, we use the established formalism of Well-Structured Marking-Nets (WSM-Nets) [RRD04c] (cf. Section 2.2.3) and extend it for our purposes. As motivated previously, an activity can be subdivided into a set of atomic steps. The lower two lanes in Figure 6.12 show the atomic steps assigned to the process activities as well as the data flow between these steps. For example, the atomic steps “measure weight”, “measure temperature”, and “wash patient” are assigned to activity “prepare patient”. “Provide weight” is an exam-

ple of a data provision service assigned to activity “operate” as atomic step. If an exceptional situation (e.g., failure at the “measure weight” atomic step level) occurs, this data provision service will be invoked in order to supply input data element “weight” of the activity “operate” (and particularly of its atomic step “anesthetize”). We define a partial order relation on the set of atomic steps (incl. data provision services) assigned to a certain activity. The precedence relation depicts a micro control flow between elements of this set. Note that, by contrast, a macro control flow is defined between activities. We set up this relation by assigning numeric labels to atomic steps, e.g., an atomic step with numeric label “1” is considered as a predecessor of all atomic steps with numeric label “2” or greater. By default, all atomic steps have number “1”, i.e., they can be worked on in parallel. In this case, the actor which works on the respective activity is considered as being the expert in choosing the best order. Data provision services have number “0” since they must be executed before all atomic steps assigned to the same activity, in order to properly supply these atomic steps with the required input data.

So far the formal definition of WSM-Nets has not considered splitting activities into atomic steps. Therefore we extend the definition by including this additional level of granularity. In the following, S describes a process schema.

Definition 6.2 (Extended WSM-Net – Atomic Steps) A tuple $S = (N, D, NT, CtrlE, DataE, ST, P, Asn, Aso, DataE_{extended})$ is called an extended WSM-Net if the following holds:

- N is a set of activities and D is a set of process data elements
- $NT: N \mapsto \{StartFlow, EndFlow, Activity, AndSplit, AndJoin, XorSplit, XorJoin, StartLoop, EndLoop\}$
To each activity, NT assigns a respective node type.
- $CtrlE \subseteq N \times N$ is a precedence relation setting out the order between activities.
- $DataE \subseteq N \times D \times NAccessMode$ is a set of data links between activities and data elements (with $NAccessMode = \{read, write, continuous-read, continuous-write\}$)
- ST is the total set of atomic steps defined for all activities of the process (with $P \subseteq ST$ describing the set of data provision services)
- $Asn: ST \mapsto N$ assigns to each atomic step a respective activity.
- $Aso: ST \mapsto \mathbb{N}$ assigns to each atomic step a number indicating in which order the atomic steps of a certain activity are to be executed. By default: If $s \in P$, $Aso(s) = 0$ holds; otherwise, $Aso(s) = 1$.
- $DataE_{extended} \subseteq ST \times D \times STAccessMode$ is a set of data links between atomic steps and data elements (with $STAccessMode = \{read, write\}$)

As can be seen in the example from Figure 6.12, there are atomic steps which produce data (e.g., “measure weight”) and others which do not write any data element (e.g., “wash patient”). In order to express this fact, we logically extend the set $DataE$ to set $DataE_{extended}$ which comprises all read/write data links between atomic steps and data elements. In particular, an intra-activity data dependency may be defined such that intermediate results of an activity execution can be passed between subsequent atomic steps st_1 and st_2 with $Asn(st_1) = Asn(st_2)$; i.e., $\exists (st_1, d, write), (st_2, d, read) \in DataE_{extended}$. As an example (Figure 6.12), consider the intra-activity data flow from “anesthetize” to “operate” via data element “sensory perception degree”. In fact, the atomic step “operate” needs this data element to decide when to begin surgery.

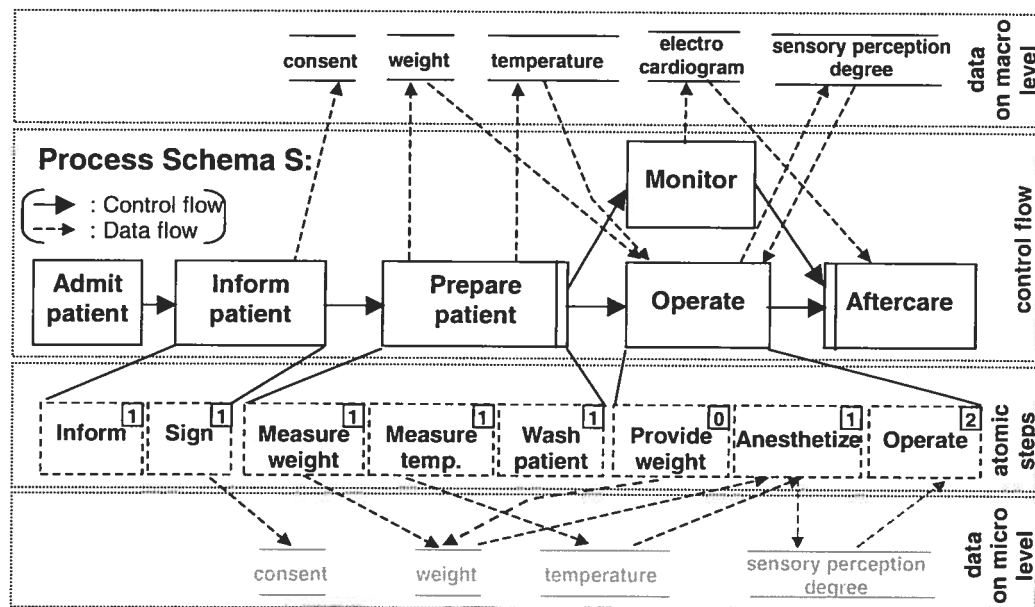


Figure 6.12. Medical Treatment Process (Atomic Steps)

Intuitively, since we have extended WSM-Nets by adding atomic steps we also have to extend the definition of process instance markings by assigning markings to atomic steps.

Definition 6.3 (Process Instance on Extended WSM-Net) A process instance I on an extended WSM-Net S is defined by a tuple $(S, M_{extended}^S, Val^S)$ where:

- $S = (N, D, NT, CtrlE, \dots)$ denotes the extended WSM-Net I was derived from
- $M_{extended}^S = (NS^S, STS^S)$ describes activity and atomic step markings of I :
 $NS^S: N \mapsto \{NotActivated, Activated, Running, Completed, Skipped\}$
 $STS^S: ST \mapsto \{NotActivated, Activated, Running, Completed, Skipped\}$
- Val^S describes a function on D . It reflects for each data element $d \in D$ either its current value or the value *UNDEFINED* (if d has not been written yet).

Markings of activities and atomic steps are correlated. When an activity becomes activated, related atomic steps (with lowest number) become activated as well. The atomic steps will then be carried out according to the defined micro control flow. As soon as one of them is executed, both the state of this atomic step and of its corresponding activity change to *Running*. An activity is marked as *Completed* after completion of all corresponding atomic steps. Finally, if an activity is skipped during process execution, all related atomic steps will be skipped as well.

As already motivated, it is important to distinguish between data elements that are only relevant in the context of applications and data elements that are relevant for process progress as well. We can see whether a data element is relevant for the process if there is an activity reading this data element.

Definition 6.4 (Data Relevance) Let S be an extended WSM-Net, let $w \in \{write, continuous-write\}$ and $r \in \{read, continuous-read\}$. Then we denote $d \in D$ as

- an exclusive application data element if
 $\exists (n, d, w) \in DataE \Rightarrow \neg \exists (m, d, r) \in DataE$
- a process relevant data element if
 $\exists (n, d, w) \in DataE \Rightarrow \exists m \in Succ^*(S, n) \cup \{n\}: (m, d, r) \in DataE$
 $Succ^*(S, n)$ denotes all direct and indirect successors of activity n .

The Data Relevance dimension captures both data elements that are produced by the process, but are only consumed by the application, and data elements that are produced and consumed by the process. In the medical treatment process (cf. Figure 6.12), data elements “weight” and “temperature” taken during the “prepare patient” activity are examples of process relevant data elements. They are of utmost importance for carrying out the subsequent “operate” activity (*i.e.*, to calculate the quantity of anesthesia that has to be administered to the patient). The “electro cardiogram” data element tracked during

the “monitor” activity is another example of process relevant data. It is used by the “aftercare” activity. By contrast, “consent” is an exclusively application data element. As explained in Section 6.4.2.1, when a failure occurs, an exclusive application data element may become relevant for the process as well. A patient who already consented upon a surgery accepts the risks, and the “consent” data element may thus be used in subsequent activities dealing with respective problems. Turning now to the container transportation process, “current position” is an exclusive application data element whereas “container temperature” is a process relevant data element (cf. Figure 6.13).

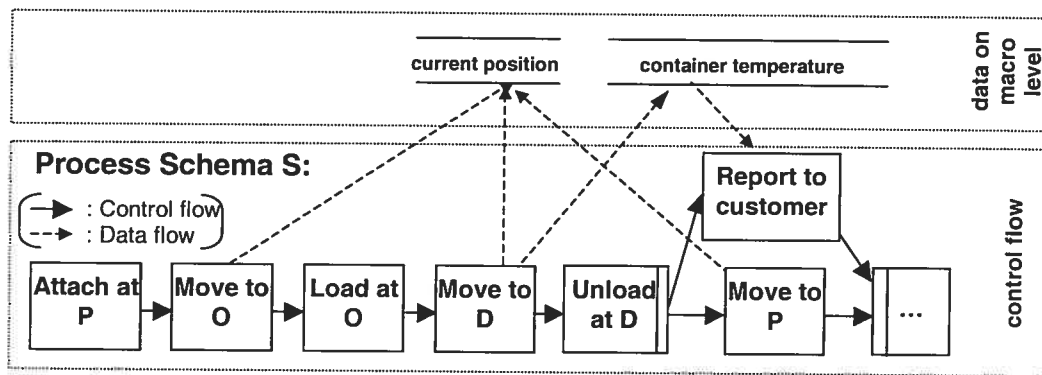


Figure 6.13. Container Transportation Process

We now define the notion of data update frequency. Based on this notion we will be able to define a criterion for safely interrupting running activities while preserving their context. Intuitively, for a discrete data update by atomic steps there are certain periods of time between the single updates, whereas for continuous data updates the *time slices* between the single updates converge to 0.

For defining the *time slices* between data updates, we need the function:

$$stp: ST \mapsto \mathbb{R} \cup \{UNDEFINED\}$$

This function maps each atomic step of *ST* either to a specific point in time or to *UNDEFINED*. In detail:

$$stp(st) := \begin{cases} t_{st} & \text{if } \exists (st, d, write) \in DataE_{extended} \\ UNDEFINED & \text{otherwise} \end{cases}$$

$$\text{whereby } t_{st} := \begin{cases} \text{completion time of } st \\ \infty & \text{by default} \end{cases}$$

Note that the infinite default value we assign to t_{st} is updated as soon as st is completed. Hence, the real completion time of st is assigned to t_{st} .

Definition 6.5 (Data Update Frequency) Let S be an extended WSM-Net, let $w \in \{\text{write}, \text{continuous-write}\} \subset NAccessMode$, and let $d \in D$, $n \in N$ with $(n, d, w) \in DataE$. Let further ST_n^d be the set of atomic steps associated with activity n and writing data element d ; i.e., $ST_n^d := \{st \mid asn(st) = n, \exists (st, d, write) \in DataE_{extended}\}$.

Then we denote (d, n) as:

- a discrete data update of d by n if $\exists (n, d, write) \in DataE$
In terms of atomic steps: $\forall st \in ST_n^d : stp(st) = t_{st} \neq UNDEFINED$
- a continuous data update of d by n if $\exists (n, d, \text{continuous-write}) \in DataE$
In terms of atomic steps: $ST_n^d = \emptyset$

In case an activity n continuously updates a data element d , no atomic steps writing d are dissociated, i.e., there are no atomic steps associated with n that write d ; e.g., take the absence of atomic steps writing the “current position”, the “container temperature”, and the “electro cardiogram” in Figures 6.12 and 6.13. These data elements are examples of data continuously updated respectively by a GPS system, a thermometer, and a cardiograph instrument.

On the other hand, the set of atomic steps discretely writing a data element may be limited to only one atomic step. The “consent”, the “weight”, and the “temperature” are written once respectively by the “sign”, the “measure weight” and the “measure temperature” atomic steps (cf. Fig. 6.13).

Figure 6.14 summarizes the classification of the data involved in the medical treatment process and in the container transportation process, taking into account the general data classification scheme presented in Figure 6.11.

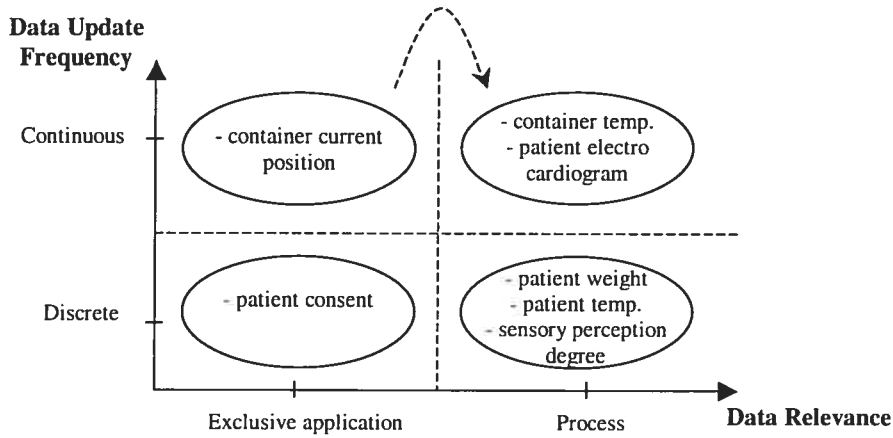


Figure 6.14. Data Classification in the Medical Treatment and Container Transportation Processes

6.4.2.1.2 Correctness Criterion

In order to correctly deal with exceptional situations, it is crucial to know those points in time when running activities can be *safely* interrupted. A running activity is safely interrupted means that the context of that activity is kept such that all input data of subsequent activities are correctly supplied. This context preservation will allow for finding possible solutions for exceptional situations. We denote these points in time as *safe points* of the respective activities.

The challenging question is how to determine the safe point of an activity. In order to adequately answer this question, our distinction between continuous and discrete data update is helpful. As the following definitions show, it is possible to precisely determine the particular *safe interrupt points* for discrete and continuous data updates, *i.e.*, those points in time when the respective data are updated such that subsequent activities reading these data are correctly supplied.

Definition 6.6 (Safe Interrupt Point for a Discrete Data Update) Let (d, n) ($n \in N$, $d \in D$) be a discrete data update of d by n , and let ST_n^d be the set of atomic steps associated with n and writing d . Let further $B := \{stp(st), st \in ST_n^d \mid \neg \exists p \in P: Asn(p) = n \text{ and } (p, d, write) \in DataE_{extended}\}$. Then the safe interrupt point t_{safe}^d of (d, n) corresponds to the maximum point in time any atomic step writes d (on condition that d cannot be provided by a data provision service). Formally:

$$t_{safe}^d := \begin{cases} \max(B) : B \neq \emptyset \\ UNDEFINED : otherwise \end{cases}$$

Informally, the safe interrupt point for a discrete data update by atomic steps is that maximum point in time when the last write access to the respective data element has taken place.

Definition 6.7 (Safe Interrupt Point for a Continuous Data Update) Let (d, n) ($n \in N$, $d \in D$) be a continuous data update of d by n with a start updating time t_1 and a finish updating time t_k . The safe interrupt point t_{safe}^d of (d, n) ($t_1 < t_{safe}^d < t_k$) corresponds to the time when d becomes relevant for subsequent activities. This time is fixed by the user. If no safe interrupt point is fixed by the user $t_{safe}^d := UNDEFINED$ holds.

Intuitively, for continuous data updates there is no “natural” safe interrupt point. Therefore, we offer the possibility to define a safe interrupt point by the user. An example usage for such a user-defined safe interrupt point would be the “waiting time” in order to get the right container temperature after attaching it to the vehicle that shall power the refrigeration system within the container.

In order to determine the safe point of an activity, we have to consider that there might be several safe interrupt points. One example is the activity “prepare patient” which has two safe interrupt points belonging to data elements “weight” and “temperature” (Figure 6.12).

Definition 6.8 (Activity Safe Point) Let $\{d_1, \dots, d_k\}$ be the set of data elements (continuously) written by activity $n \in N$ (i.e., $\exists (n, d_i, w) \in DataE$, $i = 1, \dots, k$, $w \in \{write, continuous-write\}$). Let further $t_{safe}^{d_1}, \dots, t_{safe}^{d_k}$ be the related safe interrupt points.

Then we denote $t_{safe} = \max\{t_{safe}^{d_1}, \dots, t_{safe}^{d_k}\}$ as the safe point of n (if $t_{safe}^{d_i} = UNDEFINED \forall i = 1, \dots, k$, t_{safe} is set to $UNDEFINED$ as well). Thereby, t_{safe} corresponds to the time when n can be safely interrupted keeping its context. An activity n can be safely interrupted if all input data of subsequent activities of n are provided.

Using the notion of activity safe point, we can state a criterion based on which it is possible to decide whether a running activity can be safely interrupted or not.

Criterion 6.1 (Interrupting a Running Activity by Keeping its Context) Let S be an extended WSM-Net, let I be an instance on S , and let $w \in \{\text{write, continuous-write}\} \subset \text{NAccessMode}$. A node $n \in N$ with $NS^S(n) = \text{Running}$ and safe point t_{safe} can be safely interrupted at $t_{\text{interrupt}}$ if one of the following conditions holds:

- $\neg \exists (n, d, w) \in \text{DataE}$
- $t_{\text{safe}} \leq t_{\text{interruption}}$ OR $t_{\text{safe}} = \text{UNDEFINED}$
- $\forall (n, d, w) \in \text{DataE}, t_{\text{interrupt}} < t_{\text{safe}}^d : d$ is an exclusive application data element

A running activity can be safely interrupted from a process perspective if it either writes no data or if it solely writes exclusive application data. If a running activity writes process relevant data it can be safely interrupted if it has an undefined safe point or its safe point has been already transgressed. Finally, if exclusive application data become process relevant (e.g., if an exception handling process makes use of the full context of the interrupted activity), the last condition of Criterion 6.1 may not be applicable.

In order to illustrate the defined correctness criterion, we consider the container transportation process. Based on process schema S provided in Figure 6.13, instance I_S in Figure 6.15 has been started. Taking into account a defined transportation network, each of the activities' locations in I_S is captured by a coordinate (x, y) . E.g., the origin and the destination locations in activity "move vehicle V from Montréal to Québec" would respectively correspond to the coordinates $(1.5, 3.5)$ and $(13, 8)$ within the transportation network. Suppose that a road traffic problem occurs at time $t_{\text{interrupt}} = t_{\text{begin}}^n + 75\text{minutes}$ (elapsed time since departure) while V is on the road between Montréal and Québec. At this time, suppose that the GPS system is indicating $(7, 5.5)$ for the current position of V . To avoid the traffic problem, an optimization module may propose a new transportation solution that consists of changing the already planned route leading to Québec. The new route includes a detour via another location, that is Trois-Rivières located at position $(7, 7)$. However, this new solution is only possible if V is close enough to Trois-Rivières, which means that the current position of V is beyond $(6, 5)$. This corresponds to $t_{\text{safe}}^{\text{"CurrentPosition"}} = t_{\text{begin}}^n + 60\text{minutes}$. In addition, suppose that the right container temperature is reached 15minutes after finishing loading the container and hence after the departure from the origin location, i.e., $t_{\text{safe}}^{\text{"ContainerTemperature"}} = t_{\text{begin}}^n + 15\text{minutes}$. Taking into account Definition 6.8, the safe point of activity "move vehicle V from Montréal to Qué-

bec” corresponds to $\max\{t_{safe}^{CurrentPosition}, t_{safe}^{ContainerTemperature}\} < t_{interrupt}$. Hence, this activity can be safely interrupted. The exclusive application data element “current position” was used to generate the new solution shown in Figure 6.15. Following the road traffic problem, this data element becomes process relevant as well: it is given as input to the inserted activity “move vehicle V from current location to Trois-Rivières”. Note that in this specific example, the “container temperature” data element is not relevant for the definition of the safe point, and hence it could be fixed to *UNDEFINED*.

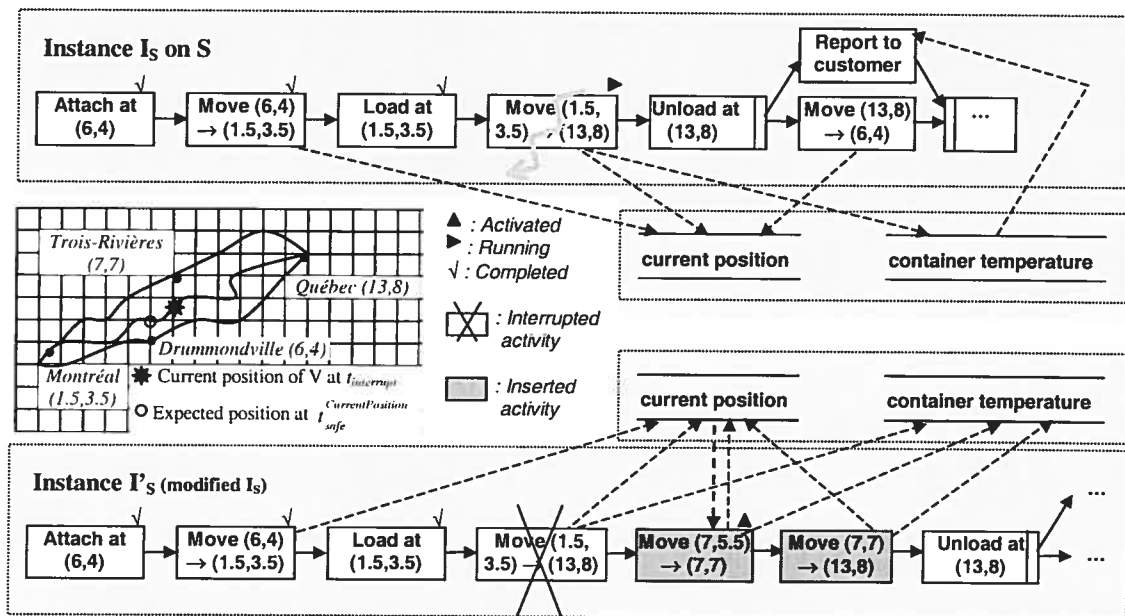


Figure 6.15. Container Transportation Scenario

6.4.2.1.3 Discussion

The solution proposed to ensure a safe interruption of a running activity adopts a “divide and conquer” approach: An activity is divided into atomic steps so that the interruption of this activity becomes possible by preserving its context.

In [MS02, SSO01] “pockets of flexibility” are defined. So called “containers” comprise different activities and constraints posed on these activities (*e.g.*, activity B always before activity C). These containers can be inserted into certain regions within the process. If process execution reaches such a container the assigned user can choose the order of

working on the offered activities by obeying the imposed constraints. This idea can be compared to our approach of subdividing activities into atomic steps and posing an order relation on them if necessary. However, both approaches use a different level of granularity and focus on different aims. The approach presented by [SSO01] provides more flexibility regarding process modeling whereas our approach uses atomic steps for being able to preserve the data context in case of unexpected events during run-time.

The two kinds of data addressed by the Data Relevance dimension of our data classification scheme have already been discussed within the literature [AH02, WfMC99b]. In [WfMC99b], a differentiation is made between application data and process relevant data. It is argued that application data may become process relevant if they are used by the workflow system to determine a state change. In this paper, we adopt the same definitions and interpretations as provided in [WfMC99b]; furthermore, we judiciously highlight the fact that exclusive application data may become process relevant when a failure occurs. In [AH02], a bigger variety of process data is featured: analysis data, operational management data, historical data, etc. It is stated that application data cannot be directly accessed by a workflow system but only indirectly through instance attributes and applications themselves. Hence, only the way of accessing application data from a WfMS is discussed.

The infinite completion time assigned as a default value to an atomic step st may be more precisely predicted using, for instance, the forward/backward calculation technique based on the duration of activities as proposed in [EPG+03, EP02]. This would allow estimating an activity safe point (t_{safe}) as a specific point in time (instead of infinite) even before reaching this point.

Another interesting application of the presented results arises in the context of process schema evolution [RRD04a], *i.e.*, process schema changes and their propagation to running process instances. One important challenge in this context is to find correctness criteria in order to ensure correct process instance migration after a process schema change. According to the compliance criterion [CCP+98, RRD04a] it is forbidden to skip already running activities, *i.e.*, the respective process instances are considered as being non-compliant. However, if we transfer the concepts of safe interruption of activities to the

safe deletion of activities the number of process instances compliant with the changed process schema can be increased.

6.4.3 The Dynamic Move of an Activity

This functionality is not provided by current WfMSs. A workaround solution consists of inserting at a new position the activity we want to move, and then to delete this activity from its current position. Deleting then inserting the activity may not be possible because of data flow conflicts (*e.g.*, detection of missing input data). In the context of a move operation, a relaxation of the consistency verifications applied for the delete operation should be done.

6.4.4 The Dynamic Modification of Activity Attributes

Functionality discussed in Sections 6.4.1 to 6.4.3 brings *structural modifications* to workflow instances. *Attribute modifications* are another kind of modification applied at the activity attribute level: the insertion, deletion, setting, and updating of activity attributes. We mainly distinguish between input/output attributes, assignment attributes, and time attributes (*cf.* Section 6.3.1). In the following, we address modifications applied in the context of each of these three types of attributes.

6.4.4.1 The Dynamic Insertion/Setting/Updating of Input Attributes

It could happen that the definition of a workflow model is not complete [Sad99]. In this case, workflow activities could be identified, but only elementary descriptions would be given. We use the terms “partial” or “just in time” execution [KBB98] where the definition of the workflow is not completed until the information becomes available (before it is required). This information may be used to set or update the value of an already defined activity attribute or to define and insert a new attribute for a specific activity not reached yet. In a combined negotiation importing process for instance, when a company begins negotiating with the supplier(s), it is still not necessary to know details regarding the “Insurance” negotiation activity. These details may come later and they may trigger the insertion of new activity attributes such as the “kind of insurance”.

In current WfMSs, the workflow modeller chooses where the values for activity instance attributes are to be obtained. Though most WfMSs run-time API provides a function to “dynamically” set activity attributes during run-time, an attribute value can still only be obtained from a so called “input container”⁶ (1) set with a default value, (2) provided when an activity instance is started or (3) linked to an output container.

This restriction regarding “when” an input container is set should be relaxed. In all cases, since the function allowing to set activity attributes is provided by most APIs, the functionality allowing activity attributes to be set/updated “*anytime*” at run-time can be implemented in a specific workflow client related to a specific application.

An advanced issue is however to allow the insertion of a new attribute to an activity in a running state. We refer to the dynamism at the activity level rather than at the workflow level. The deletion of an activity in a running state by preserving its context (Section 6.4.2.1) may remedy the insertion of a new attribute to an activity in a running state. Indeed, the running activity to which a new attribute is to be inserted is deleted by preserving the work already done. A new activity comprising the remaining work from the deleted activity is inserted. The new attribute will be defined within this activity before its insertion.

A motivation behind the insertion of a new attribute to a running activity stems from the e-negotiation domain. Indeed, sometimes the negotiation rules change during the negotiation process. This is known as a “multi-stage” negotiation. MOAI LiveExchange [Moai04] and Inspire [Int04] are two examples of auction/bargaining systems that support multi-stage negotiations [NBB+03]. In MOAI LiveExchange, trade terms may be settled many times during negotiation and in Inspire, an agent participating in a negotiation may decide to negotiate a new issue that will be added to the already existing issues. In both systems, a new term/condition may appear in the (e-)contract.

In CONSENSUS, to support multi-stage negotiations, mainly to secure contracts in B2B negotiations, it should be possible to dynamically insert new attributes to e-negotiation

⁶ A container is defined as a data element.

activities while these activities are already in a running state. Indeed, the process of drawing up a contract consists of negotiating a number of terms and conditions. The (two) parties should agree upon these terms and conditions by negotiating them. In this context, a new stage of negotiation is entered when a new term/condition is “dynamically” introduced for negotiation. Here is an example of a scenario that can occur in a goods importing process. A company A sells cars and a company B is considering purchasing. A number of issues (*e.g.*, price, warranty) are negotiated between A and B. While negotiating, a new issue (*e.g.*, cars shipment) may appear of interest for the buying company to negotiate.

6.4.4.2 The Dynamic Deletion of Input/Output Attributes

The need for the “dynamic deletion of an activity attribute” functionality was motivated from studying the “Combined Negotiation” case. Indeed, dependent attributes may easily appear in CNs. When modelling the CN workflow, a “carry out deals” task for instance usually takes as input attributes such as the “final price” of each item that is planned to be negotiated within the workflow. Suppose that during run-time, a specific negotiation task N for a specific item I has to be deleted. Since N produces the “final price” attribute that is consumed by the “carry out deals” task, N cannot be deleted.

A first possible solution is to remove the consuming task (*e.g.*, the “carry out deals” task), to insert a new one with the appropriate attributes (*i.e.*, without the input attribute: “final price” of I), and finally to delete N. Another straightforward solution is to delete the input attribute from the consuming task(s). This removes the “parameter unsupplied” problem that appears when trying to delete N during the pre-deletion step, *i.e.*, the data dependency verification step, and N can then be deleted.

In the example we have just presented, we experimented the need to delete a specific activity input attribute and this was necessary for an activity deletion purpose. The semantic verification apart, the *input* attribute deletion operation does not require consistency verifications. At the semantic level, we must ensure for instance that the input attributes are not used by a specific code/program, etc.

However, the deletion operation of an activity *output* attribute should be handled with caution. It can be comparable to an activity deletion; the same data dependency verification should be applied. Note that in our case studies, we did not experiment the need to delete an output attribute. Examples can be found in case studies stemming from the medical domain. *E.g.*, a specific activity may no more provide a specified output attribute. The latter should be removed.

6.4.4.3 The Dynamic (Re-)Assignment of Activities to a Participant

This should be done to a *valid* workflow participant. [KBB98] evokes the late binding of resources. Verifications regarding inconsistent actor dependencies should be made. These verifications are already an issue at the design level and should be considered during run-time. The re-assignment during run-time is allowed by most WfMSs. Usually, a “task reassign permission” should be set on, and the reassignment is possible once the activity appears in the work-list of a specific user. A relaxation of this functionality allowing the activity reassignment anytime before its execution is an issue. Most APIs provide the appropriate function. It is only a matter of properly implementing this functionality in a specific workflow client.

A sophisticated issue is related to the modification of the organizational model during run-time. This requires an on-the-fly verification of the workflow assignments.

6.4.4.4 The Dynamic Setting/Updating of Time Attributes

The setting of time attributes during the design phase should already call for verifications regarding time inconsistencies. As examples, the maximum time distance between two activities should not be exceeded; the minimum time distance should not have harmful effects on subsequent activities. These verifications should be considered again during run-time.

In current WfMSs, the activity duration can be set/updated during run-time. However, the updating of the activity starting/finishing time is still an issue to be considered. Indeed, the starting time and the finishing time are defined via conditions. Since conditions, once they are specified during build-time, cannot be modified anymore, modifying time during run-time becomes very complicated.

The dynamic setting/updating of time attributes has also an effect on the resource management. A resource that has already been reserved for a specific activity may become available for another activity waiting for it.

6.4.5 The Dynamic Management of Work-lists

The reassignment or the deletion of an activity already assigned to a specific work-list should be complemented by correctly managing the underlying work-lists. Following a reassignment, the work-item that corresponds to the reassigned activity should be removed from its original work-list and it should appear in the appropriate work-list taking into account the new assignment (if not null). The work-item that corresponds to a deleted activity should be removed from its work-list. The updating of an activity input attribute or time attribute should be complemented by a correct updating of the information provided by the work-lists.

The dynamic management of work-lists should be done when implementing a specific workflow client related to a specific application. However, the WfMS API should already provide the functions allowing the implementation of this management.

6.4.6 The Automatic/Manual Modification of Workflow Instances

Dynamic modifications can be manually applied to workflow instances (human in the loop). The functionality discussed in Sections 6.4.1 to 6.4.4 should be provided from the workflow client, *e.g.*, a workflow monitoring and control tool, so that specific users can be granted the permission to manually bring modifications on workflow instances. On the other hand, some applications may require automatic modifications. The MTCT application is such an example. In fact, the MTCT system is a reactive system that reacts to specific optimization model solutions by bringing the appropriate modifications to the pool of workflow instances.

The automation level of modifications at run-time characterized by the “automatic” and the “manual” modification is considered as a property for a specific application. This property can be fixed within the template classification (Section 6.3.2).

We think that a WfMS should facilitate the integration of a tool, *e.g.*, a rule-processor, for the application of automatic modifications.

6.5 Conclusion

In this chapter, we have (1) motivated the need to review some of the already existing workflow concepts and functionality, and (2) discussed original workflow concepts and functionality to better support complex application characteristics.

Taking into account what current WfMSs provide, specifically ADEPT, a workaround solution was necessary to deal with most of the identified requirements. Even when a direct solution was apparently available, that solution had to be reviewed, leading most of the time to a workaround solution.

At the implementation level, we distinguish between (1) what the API of a specific WfMS allows to implement, and (2) what a workflow client provided by that WfMS offers as functionality. We have asked ourselves the following question: “the challenge today is it to provide a complete API and to allow the access to all the features from the provided workflow client?” When a workflow-based system is to be implemented for a specific application, a customized workflow monitoring and control tool is most of the time implemented. From this perspective, a complete API is indeed sufficient.

Only one of the identified functionalities has been formally and deeply studied: the interruption of an activity execution while preserving its context. Similarly, we intend in future work to further investigate some of the workaround solutions discussed. Theoretical foundations shall be elaborated mainly for the warm-up time concept, and for the functionality allowing one to dynamically insert a block of activities and to dynamically move an activity.

Chapter 7 The MTCT System

In the context of the MTCT application studied in Section 4.3, the processing of customer requests for container transportation is achieved by specific sequences of interdependent activities. These sequences need to be created just-in-time, and furthermore, they need to be adapted to deal with unexpected events that may occur. The creation and the adaptation of activity sequences should be based on an optimized resource management and activity scheduling. Moreover, a number of special workflow concepts and functionality are required to correctly manage activity sequences.

Taking into account the adaptive workflow framework introduced in Section 5.3, in this chapter, we devise a workflow-oriented system architecture for the processing of customer requests for container transportation:

- Optimization models are involved to take care of the resource management and of activity scheduling.
- Specific workflow concepts and functionality are used to deal with activity sequence creation and adaptation.
- Finally, the proposed architecture includes a rule processing part to reduce the time-consuming manual interaction with the system.

In the following, we first describe the transportation system framework that we developed (Section 7.1). Then, the architecture of the MTCT system is presented (Section 7.2). This architecture is based on workflow technology, optimization technology, and rule engines. Section 7.3 gives examples regarding the planning and the modification of the processing of a customer request that illustrates the use and the characteristics of the developed architecture. Section 7.4 reports on the implementation of the MTCT system. Section 7.5 concludes the chapter by exposing a set of useful new workflow concepts

and functionality derived from studying the MTCT application, and from developing the MTCT system.

7.1 The Transportation System Framework

We introduce an original transportation system framework adapted to the MTCT application [BBK+02b]. This framework is conceptually divided into two main layers: a *workflow layer* and a *coordination layer*. Refer to Figure 7.1.

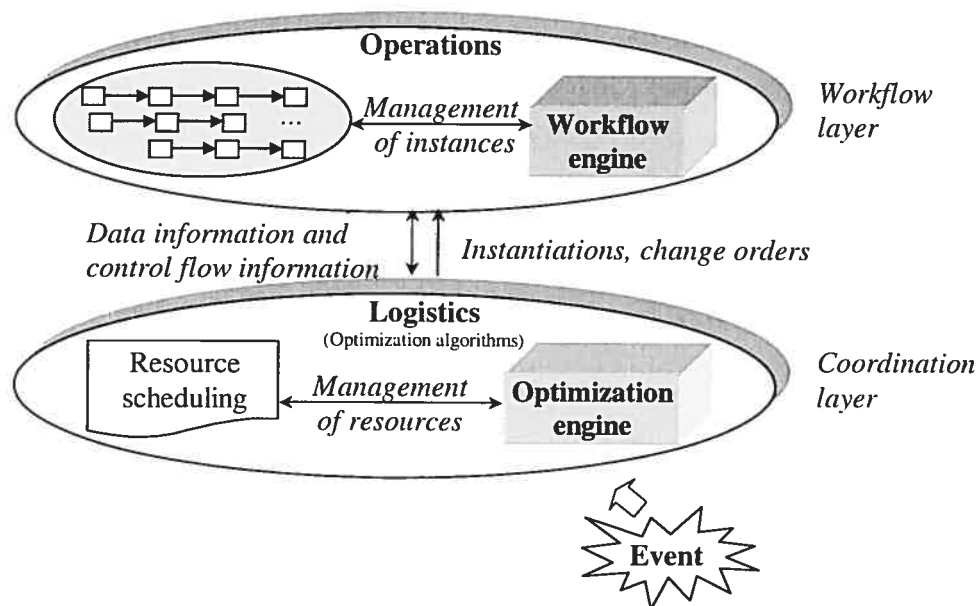


Figure 7.1. Transportation System Framework

The workflow layer essentially gathers a set of concurrently running workflow instances, each of them being associated with a specific customer request. Knowing that a workflow instance is composed of a sequence of activities, and that the state of these activities is known at any time, it is hence possible to determine the set of used resources such as vehicles, containers, and drivers. Since we are dealing with activities to be achieved by humans, the dispatching of the appropriate crews at the appropriate time plays an important role. We take advantage of the work-list concept to ensure this task. Crews have their personal work-list to quickly identify their assigned activities. It should also be possible for crews to transmit feedbacks to the coordination layer about the state of their

ongoing activities (*e.g.*, normal termination, abnormal termination due to technical problems).

The coordination layer is responsible for a certain number of tasks that ensure the efficient allocation of resources. It is responsible for receiving the new requests, for asking the workflow layer to instantiate new workflow instances, and for reacting accordingly to unexpected events by sending modification orders to the workflow layer. In brief, the coordination layer gathers a set of optimization algorithms that are used for the management of resources and for the scheduling of activities.

Following the occurrence and reception of unexpected events, the coordination layer is able to notify the workflow layer that the pool of workflow instances needs to be modified. These notifications are of four types: instantiation notifications, suspension and postpone notifications, attribute updating notifications, and structural modification notifications. Note that unexpected events do not act directly on the pool of workflow instances. Instead, they trigger resource management algorithms, and results generated by these algorithms are translated into appropriate changes of the pool. These results lead to various actions such as the (re-)scheduling of activities, and the (re-)allocation of resources. Refer to Figure 7.2.

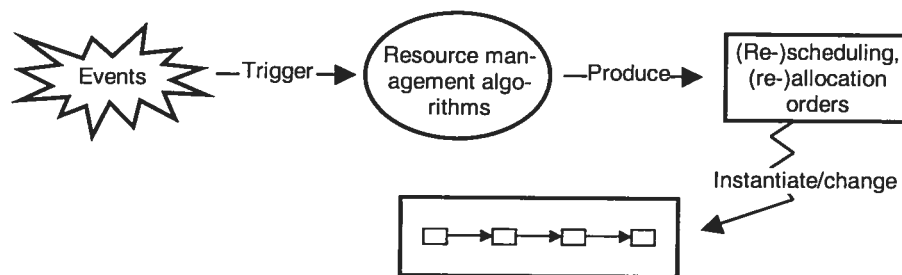


Figure 7.2. Different Steps from the Detection of an Event till the Instantiation/Change of Workflow Instances

The instantiation is usually followed by the setting of activities' attributes. In fact, the arrival of a new customer request instantiates a basic workflow model, and attributes related to the activities are determined based on the results provided by the triggered resource management algorithms. The information related to a customer request, *e.g.*,

pickup/delivery time/location, is given as input to these algorithms. It is possible that the resulting instance is not fully predefined; typically this occurs when some activities' attributes are not set from the beginning because of their unavailability. We propose to just-in-time set these attributes as soon as the needed values become available. It has to be ensured that the necessary values are available when they are required.

The transportation system framework presented in this section calls for an (automatic) interaction between the two defined layers. The development of a system based on this framework may be considered as an enterprise application integration (EAI) problem. The three main issues that should be addressed are the workflow management, the resource management, and the interaction management. The first two kinds of management are associated with the workflow layer and the coordination layer, respectively. The interaction management takes care of the exchanges between both layers. An automatic interaction may for instance be based on a rule processing approach.

7.2 Architecture of the MTCT System

Taking into account the framework presented in Section 7.1, we propose in this section a workflow-oriented system architecture applied for the MTCT application (Figure 7.3). This system – that we call the MTCT system – enables the user, *i.e.*, system administrator, to efficiently track and monitor the progress of multiple customer requests being processed. Moreover, the system allows crew members, *i.e.*, drivers, to identify at the right time their assigned activities and to transmit to the system administrator the state of each activity from its selection to its completion.

In the following, we first describe the different components of the MTCT system (Section 7.2.1). Then, an overview of its underlying management mechanisms is given (Section 7.2.2). Finally, a possible extension of the system is depicted (Section 7.2.3).

7.2.1 System Components

Two phases are distinguished in this system: the build-time phase and the run-time phase. The build-time phase is executed less frequently than the run-time phase. The end

of the build-time phase defines the starting point for a successful run-time phase execution. The latter constitutes the daily working environment.

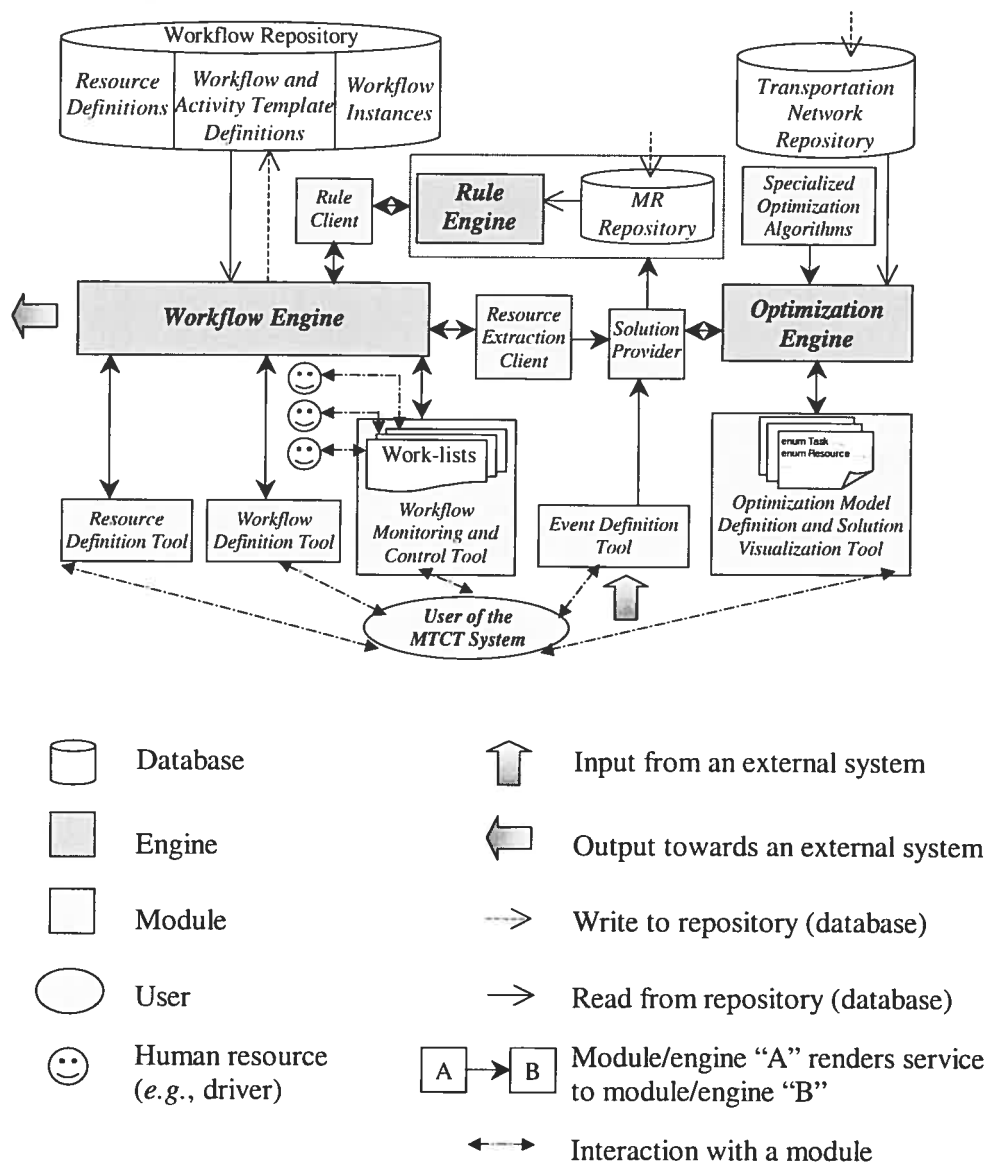


Figure 7.3. Architecture of the MTCT System

7.2.1.1 Build-time Components

During build-time, a set of activity templates is defined using the *Workflow Definition Tool*. The latter is also used to design basic workflow models that capture the sequenc-

ing of the most likely required activities for the processing of a customer request. Activity templates and basic workflow models are stored in the Workflow Repository as Workflow and Activity Template Definitions.

The *Resource Definition Tool* allows the definition of resources that make possible the accomplishment of the activities. The resources are stored in the Workflow Repository as Resource Definitions. The planned (fixed) availability of the human resources is defined via workflows using the Workflow Definition Tool. This will be detailed in Section 7.2.2.2.

Optimization models (OMs) are described with the *Optimization Model Definition Tool*. These models are used to (re-)plan the processing of customer requests. Refer to Section 7.2.2.2 for details.

Modification rules (MRs) are usually defined using a *rule editor* (not shown in Figure 7.3 for simplicity purpose). They go into the MR Repository. Modification rules and rule engines are discussed in Section 7.2.2.3.

The *Transportation Network Repository* holds information about particular locations or depots of the transportation network as well as the durations to move between two locations. This information, once it is specified, is rarely modified.

7.2.1.2 Run-time Components

At run-time, when a new event appears, the system administrator of the MTCT system uses the *Event Definition Tool* to define this event, *e.g.*, a new request arrival, as well as its related data. This triggers the selection of a specific OM. The *Solution Provider* module takes care of this selection. As long as no solution is found, a number of OMs may be solved. Specialized Optimization Algorithms are called by the *Optimization Engine* to solve a selected OM. Three data sources are used to initialize the OM:

- (1) The Event Definition Tool provides event information.
- (2) The *Resource Extraction Client* provides data related to the current reservation or unavailability of resources reflected by the state of our workflow instances.

- (3) The Transportation Network Repository already as defined for the build-time phase.

When an optimized solution is generated, it is interpreted and translated into a set of modifications that are applied on the pool of currently running instances. The *Rule Client* is responsible of automatically communicating these modifications to the Workflow Engine. Modifying the pool of workflow instances consists in the creation of a new workflow instance, or in the structural or attribute modification of an existing workflow instance. The interpretation of solution implications on this pool is the task of the *Rule Engine* and the MR Repository. The system administrator can also make manual modifications. Indeed, the optimized solution can be displayed to the system administrator via the *Solution Visualization Tool*, so that she can take decisions regarding the modification(s) to bring to the pool of instances. Manual modifications are communicated to the Workflow Engine via the *Workflow Monitoring and Control Tool*.

The *Workflow Engine* is responsible of applying modifications on the pool of workflow instances. It also executes the instances by enforcing the sequencing of the activities and by dispatching work at the appropriate time to the appropriate human resource. Work-lists, which are part of the Workflow Monitoring and Control Tool, are used to show which activity needs to be carried out. Each human resource has her personal work-list to quickly identify her assigned activities.

7.2.2 Underlying Management Mechanisms

The MTCT system architecture is based on workflow technology, optimization technology and rule engines. Workflow management and resource management constitute the essential part of the system. They can be complemented by a rule management part, so that the modifications brought to the pool of workflow instances are automated.

7.2.2.1 Workflow Management

The architecture of the MTCT system is based on WfMS modules. These modules provide advanced functionality that go far beyond the basic workflow definition, workflow execution, and workflow monitoring. Indeed, they allow for:

- Defining transportation resources (drivers, vehicles, containers, etc.) and their respective initial location and availability.
- Defining specific templates for transportation activities: “attach container to vehicle”, “move vehicle to location”, “load container”, etc.
- Tracking the state of the current workflow instances. This reflects the current reservation of the different resources.
- (Automatically) Setting specific time attributes at run-time: the minimum and maximum duration of an activity, its warm-up time, and its earliest and latest starting time.
- (Automatically) Adjusting activity attributes at run-time:
 - Rushing or postponing the execution time of a specific activity.
 - Changing the driver responsible of an activity.
 - Changing the location(s), the vehicle or the container assigned to an activity.
- (Automatically) Bringing structural modifications to workflow instances:
 - Adding a transfer to an already planned customer request processing.
 - Removing or interrupting a specific activity.
- Once the execution of a workflow instance is completed, (automatically) recording this instance as historical data (*i.e.*, audit). Workflows are hence seen as providing a way to represent a blueprint of activities so that analysis becomes possible for the detection and for the prevention of bottlenecks at the operational level.

7.2.2.2 Resource Management

The resource management is another main part of the MTCT system. Two aspects related to resource management are discussed: the static aspect and the dynamic aspect. The static aspect refers to the way the resources are captured within the system: their representation, and the management of their availability. The dynamic aspect refers to the optimized resource scheduling and allocation.

7.2.2.2.1 Static Resource Management

The diagram of Figure 7.4 describes the entities that are used for capturing the resource structure and the relations between them. A resource type (*e.g.*, vehicle) gathers a set of resources (*e.g.*, V101, V202). Unlike material resources, human resources (*i.e.*, drivers) are not continuously available but only within their own shift. The planned unavailability (*i.e.*, the complementary of the availability or shift) of the different drivers over a period of time is captured by a workflow with parallel branches. Each branch of the workflow corresponds to a specific driver and each activity of the branch defines a period of unavailability for this driver. Refer to Figure 7.5.

Resources can be assigned to activity instances. The tables corresponding to the dashed part of the entity-relation diagram (Figure 7.4) are frequently updated. At a specific time, the reservation of the different resources is deduced from the set of activity instances where the state is different from “completed”, “deleted” or “skipped” (cf. Section 2.1).

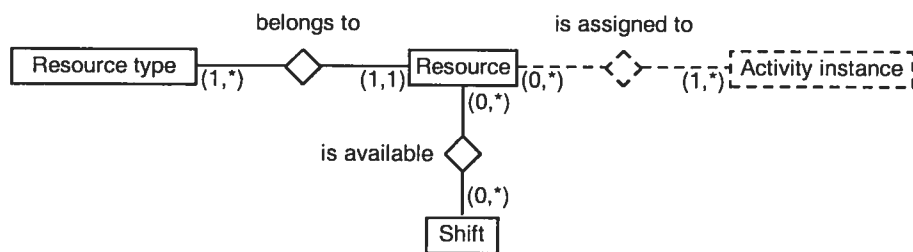


Figure 7.4. Entity-Relation Diagram for the Resource Management in the MTCT System

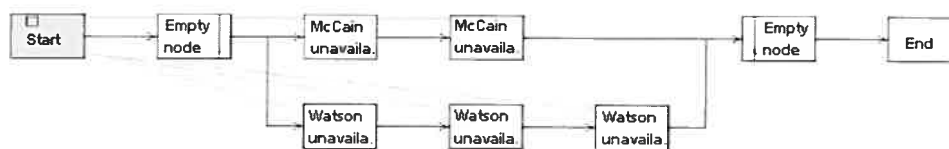


Figure 7.5. Example in ADEPT of a Planned Unavailability Workflow Instance for the Two Drivers McCain and Watson

7.2.2.2.2 Dynamic Resource Management

The need for an optimized management of resources when (re-)planning activities in the container transportation domain is well recognized [TCD93] and can be answered by

defining specific optimization models. These can be defined as a data-independent abstraction of an optimization problem in which the aim is to find the best of all possible solutions. Formally, the goal is to find a solution in the feasible region, *i.e.*, the set of all possible solutions, which has the minimum or maximum value of the objective function [Alg00]. In our context, we use OMs to plan the processing of customer requests and to re-plan this processing when necessary. These OMs should assign resources to activities while satisfying the constraints of a customer request and by taking into account the transportation network information. Our resource allocation problem is modeled as a constraint satisfaction problem that we resolve using constraint programming [Tsa93].

When modeling our problem, we leveraged the work reported in [Tri98, WHF+95]. Suitable strategies to answer a customer request according to the different path scenarios presented in Section 4.3.1 were developed. An example of a strategy consists of minimizing the duration of a request processing, *i.e.*, minimizing the reservation duration of a set of resources. Taking into account this strategy, the following defines a model that picks an available resource and schedules the different activities to answer a customer request according to the “simple scenario”, *i.e.*, the path P-O-D-P:

Given

- R : a set of resources of a specific type
- $S = \{ \langle r_i, st_{ij}, ft_{ij} \rangle \mid i=1, \dots, m; j=1, \dots, n; (st_{ij}, ft_{ij}) \text{ is a reservation block (start- ing/finishing time) for } r_i \in R \}$, at a specific point in time S reflects the current reservation of $r_i \in R$.
- *Customer request information*: origin location O , destination location D , pick-up time window (put_{min}, put_{max}) , and delivery time window (dt_{min}, dt_{max})
- *Transportation network information*: $duration(\text{Move}(P-O))$, $duration(\text{Move}(O-D))$, $duration(\text{Move}(D-P))$ where P corresponds to the depot
- *Durations of specific operations*: $duration(\text{Load})$, $duration(\text{Unload})$

Objective function

$$Z = duration(\text{Waiting_time}(O)) + duration(\text{Waiting_time}(D)) + c$$

$$c \text{ is a constant: } c = duration(\text{Move}(P-O)) + duration(\text{Load}) + duration(\text{Move}(O-D)) + duration(\text{Unload}) + duration(\text{Move}(D-P))$$

Z corresponds to the duration of the request processing.

Optimization

Minimize Z

Subject to the following constraints (where t^* corresponds to the leaving time at P):

$$(C1) \ t^* + duration(\text{Move}(P-O)) + duration(\text{Waiting_time}(O)) \geq put_{min}$$

$$\begin{aligned}
(C2) \quad & t^* + \text{duration}(\text{Move}(P-O)) + \text{duration}(\text{Waiting_time}(O)) + \\
& \text{duration}(\text{Load}) \leq \text{put}_{max} \\
(C3) \quad & t^* + \text{duration}(\text{Move}(P-O)) + \text{duration}(\text{Waiting_time}(O)) + \\
& \text{duration}(\text{Load}) + \text{duration}(\text{Move}(O-D)) + \\
& \text{duration}(\text{Waiting_time}(D)) \geq \text{dt}_{min} \\
(C4) \quad & t^* + \text{duration}(\text{Move}(P-O)) + \text{duration}(\text{Waiting_time}(O)) + \\
& \text{duration}(\text{Load}) + \text{duration}(\text{Move}(O-D)) + \\
& \text{duration}(\text{Waiting_time}(D)) + \text{duration}(\text{Unload}) \leq \text{dt}_{max} \\
(C5) \quad & \forall \langle r, st_j, ft_j \rangle \in S \text{ where } r \in R, t^* > ft_j \vee t^* + Z < st_j
\end{aligned}$$

When selecting a specific OM, the Solution Provider module provides to the Optimization Engine the necessary data to solve this model (*i.e.*, the “given statements”). Once a solution is returned from the Optimization Engine to the Solution Provider, the latter passes it over to the Rule Processing part of the system.

7.2.2.3 Rule Management

In the architecture of the MTCT system, we use rule engines to represent and exploit modification rules. A rule such as: “*If a new request arrives, and if a solution is found when a specific optimization model is solved, and if a specific basic workflow model has already been defined, and if a workflow instance manager exists, then a new workflow instance related to the newly arrived request is instantiated from the basic workflow model*” can be nicely coded as a declarative statement [MB00]. The rules can be coded as standalone atomic units, separate from and independent of the rest of the application logic. This makes the rules easier to develop and maintain. In the following, we describe the design and the implementation of modification rules.

7.2.2.3.1 Designing Modification Rules

At the design level, we use UML state diagrams to model modification rules and to specify the different states as well as the transitions between these states. One state diagram gathers rules logically interrelated. As an example, there exists a set of rules that should be applied so that a created workflow instance becomes ready to execute. The “Event[Condition]/Action” paradigm is applied to represent modification rules; it allows to shift from one state to another state. An example of such state diagram is depicted in Figure 7.6. For simplicity purpose, an English-like syntax has been used to define “Event[Condition]/Action” statements. This example has been developed in a standalone

fashion, not considering inputs from optimization models previously described. It presents the different states in which a workflow instance is involved from the adjustment of its structure till the assignment of its attributes. This state diagram provides a possibility to modify a workflow instance just after its creation taking into account the state of the different resources:

- (1) The creation of a workflow instance is triggered by the arrival of a customer request; the condition “request accepted” should hold.
- (2) The location of the different resources is verified, and the structure of the workflow instance is adjusted consequently. It is verified for instance if an empty container, a vehicle, and a driver are already available at the request’s origin location at pick-up time. In this case, the two activities “attach container to vehicle” and “move vehicle to origin location” are deleted.
- (3) The possibility to transfer a container from one location to another before delivering it to the final destination is verified, and the initial P-O-D-P path is completed with intermediate locations. Basically, “move vehicle to location” activities are inserted between the activity “load container at origin” and the activity “move vehicle to destination”. The number of such inserted activities depends on the number of transfers. In this example, a maximum of three transfers is modeled. The insertion of more than one “move vehicle to location” activity corresponds to the insertion of a sub-workflow.
- (4) The change of vehicle on an intermediate location is verified, and a “detach container from vehicle” activity and an “attach container to vehicle” activity are inserted between two “move vehicle to location” activities. This is done by inserting a “detach-attach” sub-workflow. In parallel, activity’s attributes are assigned as soon as they become available.
- (5) The end state is reached once all activities’ attributes are assigned.

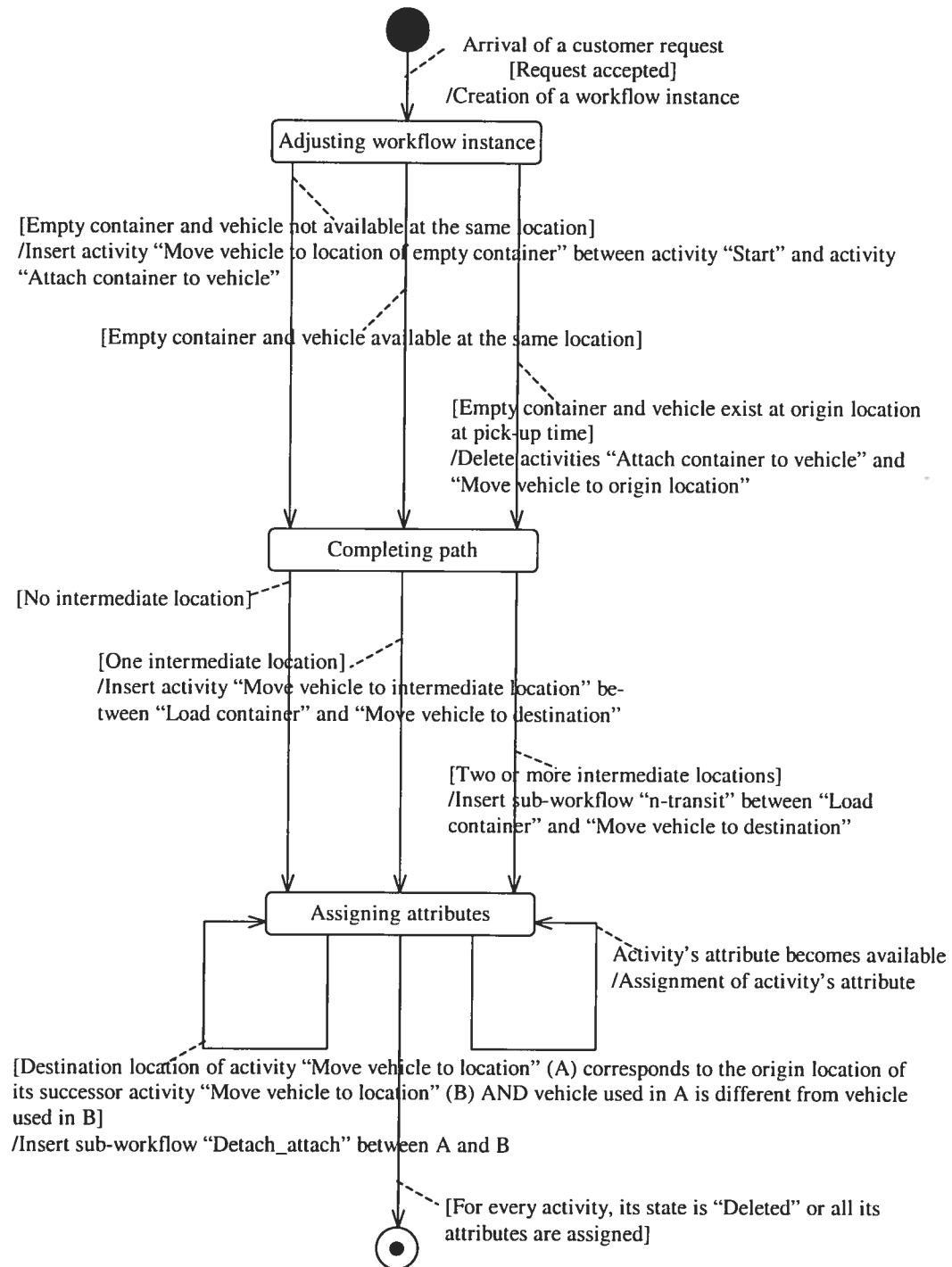


Figure 7.6. Workflow Instance Creation and Adaptation Following a Request Arrival – State Diagram

7.2.2.3.2 Implementing Modification Rules

At the implementation level, we use rule engines to write our modification rules. In fact, a rule engine usually includes a special language for writing rules. It is defined as a software component designed to evaluate and execute rules [MB00]. Rule engines have already been applied for dynamic modification of workflows [MR99]. This approach intends (1) to detect semantic exceptions, (2) to derive which instances and control flow areas are affected, and (3) to automatically adjust the affected areas. In the MTCT system, we only experimented with the automatic workflow instantiation and the automatic attributes setting; however, the automatic structural modification of instances can take advantage of the approach proposed in [MR99].

7.2.3 Interface of the MTCT System to External Systems

The MTCT system provides an input point from an external system through the Event Definition Tool, and an output point towards an external system through the Workflow Engine. A simulation system, such as ARENA [KSS02], is an example of an external system that can be used to evaluate the performance of the MTCT system.

On the one hand, the different manual activities of concurrently running workflow instances can be simulated. Particularly, resources are represented using specific icons (Figure 7.7), and the different states of the activities are simulated. As an example, icons that correspond to resources assigned to activities in a “running” state are animated. Activities’ states are communicated to the simulation system via the Workflow Engine.

On the other hand, once an activity is completed at the simulation side, this information is communicated to the MTCT system via the Event Definition Tool. The activity’s state can now turn to “completed”. Unexpected events can also be simulated and communicated to the MTCT system. For instance, a traffic problem can be simulated as discontinuing a vehicle in movement.

We identified two main challenges in integrating a simulation system with the MTCT system. First, the simulation system needs to be synchronized with the MTCT system. In systems such as ARENA, an extension called ARENA Real Time provides this feature of synchronization. Second, the simulation in the same simulation environment of many

activities executed in parallel is a must. This is possible for instance if many flowcharts at the simulation side can be launched in parallel.

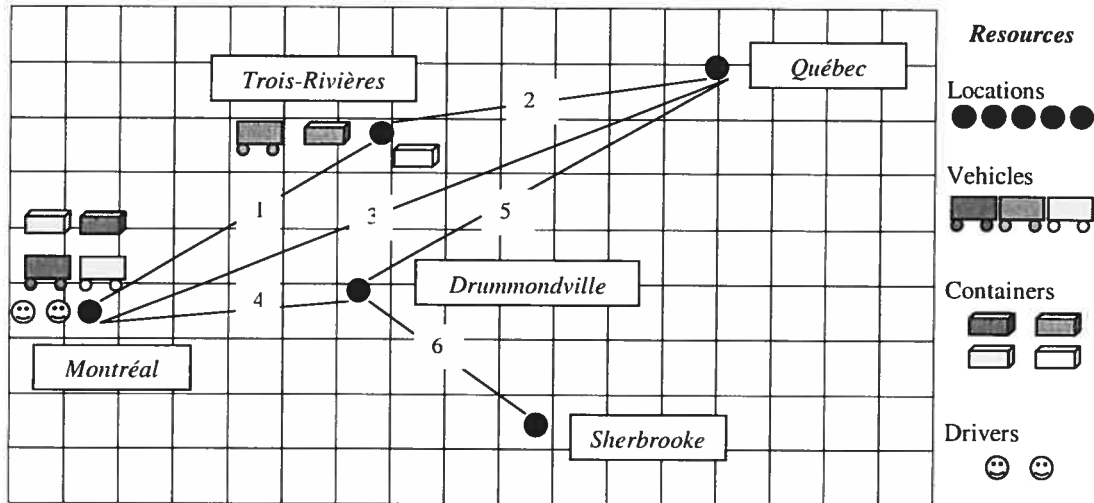


Figure 7.7. A Transportation Network Representation: Resources Represented as Icons in a Simulation Environment

7.3 Planning and Modifying the Processing of Customer Requests - Examples

We illustrate here the different steps for satisfying a customer request within the MTCT system. The example already discussed in Section 4.3.2.1 is considered again here. When a request is received, the system administrator uses a “request information” form (Figure 7.8) provided by the Event Definition Tool to specify the related information. This information, the availability of the resources and the transportation network information are used to generate a solution.

If a solution is found (as in our case), the system administrator uses the Workflow Monitoring and Control Tool to instantiate a basic workflow model and to adjust this model taking into account the solution shown in Figure 7.8, *i.e.*, to delete the “wait at O” activity since the solution does not show a waiting time at O. If no solution is found, the customer request is rejected.

Request Information

Origin (pickup location)

Destination (delivery location)

Earliest Pickup Time Earliest Delivery Time

Latest Pickup Time Latest Delivery Time

Solution found...

Container: C111
Driver: Watson / V202

Attach container (Parking) at time: Wed Oct 15 08:10:00 EDT 2003 <-5>
Parking - leaving time: Wed Oct 15 08:15:00 EDT 2003 <-105>

Origin - arrival time: Wed Oct 15 10:00:00 EDT 2003
Load container (Origin): <-30>
Origin - leaving time: Wed Oct 15 10:30:00 EDT 2003 <-165>

Destination - arrival time: Wed Oct 15 13:30:00 EDT 2003
Unload container (Destination): <-30>
Destination - leaving time: Wed Oct 15 14:00:00 EDT 2003 <-75>

Parking - arrival time: Wed Oct 15 15:15:00 EDT 2003
***Waiting time before delivery: 15 minutes

Duration in minute

Figure 7.8. “Request Information” Form

Two types of edges are used in our workflow model: the control edges and the time edges. The used WfMS prototype ADEPT does not allow the specification of a fix calendar date for the activities’ starting time. We use instead the “time edge” concept and define a minimum and a maximum distance between the “start” activity (S) and each of the other activities (A). The earliest and the latest starting time of (A) are specified taking into account the real starting time of (S). Once the execution of (S) is completed, its real starting time ST_S becomes known. The minimum distance and the maximum distance of the “time edge” between (S) and (A) are respectively equal to $EST_A - ST_S$ and $LST_A - ST_S$.

The system administrator launches (S) to specify the five following output attributes (cf. Figure 7.8): the customer request origin location (Quebec), the customer request destination location (Montreal), the central depot of our transportation network, and the container and vehicle IDs shown in the solution (C111 and V202). These attributes are given as input to the different activities of the workflow instance. The other elements of

the solution (*e.g.*, driver, starting time/duration of the activities) are used to set the assignment attribute and the time attributes for each activity.

The set of steps just accomplished by the system administrator: workflow instantiation, activity deletion, execution of (S) and attributes setting, can be automated so that time-consuming manual interactions with the system are reduced. For that reason, we need modification rules such as the one shown in Figure 7.9 in the ILOG JRules [JRules04] notation. Rules have a “WHEN part” which specifies the conditions that must be met in order for the “THEN part” to be executed. The rule in Figure 7.9 applies to a workflow instantiation. Four class instances are involved in this rule: RequestInformation and OptimisationModel are classes from our implemented application; ProcessTemplate and ProcessInstanceManager are classes provided by the ADEPT API.

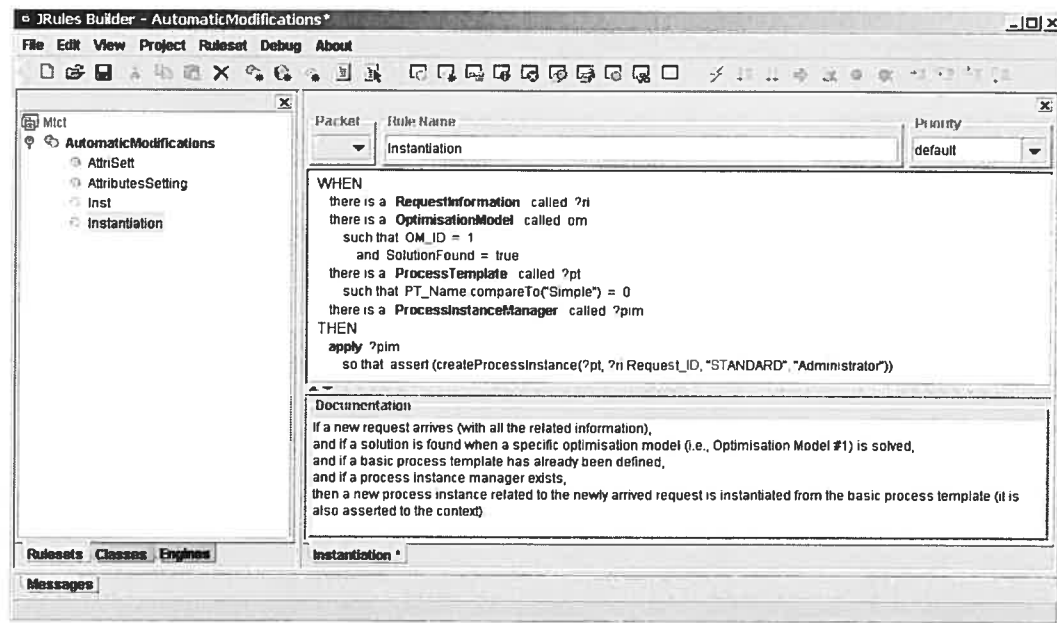


Figure 7.9. A Modification Rule of the Pool of Workflow Instances

Another interesting example of a modification rule addresses the handling of a road traffic problem. Suppose that during the execution of a “move vehicle to destination” activity, a road traffic problem occurs. To cope with this problem, a new transportation solution that consists of changing the already planned route leading to the destination location may be proposed. The new solution includes a detour via another location. How-

ever, this solution can be considered only if the activity has been safely interrupted. This example has already been discussed in the context of the studied “activity execution interruption” functionality (cf. Section 6.4.2.1.2, Figure 6.15). The following rule captures the just exposed reasoning:

<i>If</i>	<i>a road traffic problem occurs (with all the related information)</i>
<i>and if</i>	<i>the “move vehicle” activity has been safely interrupted</i>
<i>and if</i>	<i>a solution is found when a specific optimization model (i.e., <code>change_route optimization model</code>) is solved</i>
<i>then</i>	<i>two “move vehicle” activities are inserted to the workflow instance related to the request in question</i>

7.4 Implementation of the MTCT System

The implementation of the MTCT system includes a WfMS (ADEPT with an API extension), an optimization system, and a rule processing system.

We use the ADEPT prototype to cover the workflow management part of the system. Besides the selection criteria already stated in Section 5.2.1, ADEPT has been adopted because it supports in a certain way the “activity template” concept, some temporal aspects, except the WUT (introduced in Chapter 4, Table 4.1, and discussed in Chapter 6, Section 6.3.3.3), and two structural changes: the insertion and the deletion of an activity.

A *Mediator* component that extends the existing ADEPT API was implemented. This component provides functions for the dynamic setting/updating of input attributes, assignment attributes and time attributes, and for the dynamic management of work-lists. Figure 7.10 shows the WfRM-compliant ADEPT structure with the added Mediator component.

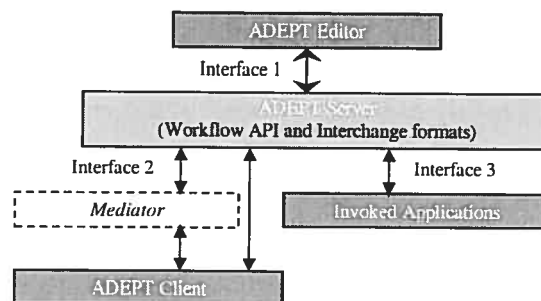


Figure 7.10. The Added *Mediator* Component within the ADEPT Structure

We use OPL Studio from ILOG [OPL04] to define OMs that are solved using the CPLEX optimization algorithms. Since our implementation is based on ADEPT which is implemented in Java and which uses an Oracle relational database, the advantage of OPL is twofold: (1) We can access its C++ API from Java code, relying upon the Java Native Interface (JNI). So, once a model is designed, compiled and tested in OPL Studio, it can be easily solved from a Java application by interfacing with OPL. (2) We can establish a connection to a database and initialize the model by reading the appropriate relational tables. Having this in mind, we implemented the ADEPT Resource Extraction Client and the Solution Provider in Java.

We have integrated ILOG JRules into our MTCT system to cover the rule management part of the system. JRules is a rule engine that combines rule-based techniques with object-oriented programming. Its advantage is that it can be easily accessed from a Java application. Hence, the Rule Client has been implemented in Java. Basically, a *context* is defined in this client. It serves as an interface between the Java application and the ILOG JRules engine. It comprises two containers: (1) a *working memory* which is the place where ILOG JRules stores all the objects with which it is currently working, *e.g.*, the workflow models and the workflow instances; and (2) an *agenda* which is the place that stores rule instances that are ready to be fired. Note, modification rules are designed, compiled and tested within ILOG JRules Builder before they are given as input to the Rule Client.

In Figure 7.11, we present a screenshot of the MTCT system. The main window in (a) shows the Workflow Monitoring and Control Tool. It provides functionality the system

administrator can use to modify the pool of the workflow instances. The first two windows (top right) are monitoring windows and show running workflow instances: a planned unavailability workflow instance, and one of the customer request processing instances that is going on. The three windows at the bottom right show the current reservation of the different resources. This information is automatically extracted and used by the Solution Provider component; however, the system administrator is also able to visualize it at any time. The last window here (bottom left) shows one of the possible windows the system administrator can access to make manual modifications to the pool of instances – the “Activity (re-)assignment” in this case. In fact, each time she chooses one of the six possible operation options, the corresponding window is opened. The two windows in (b) show the work-lists of two specific drivers. All necessary information is available for the execution of an activity related to a request processing instance. As we can see, activities related to a planned unavailability workflow instance are also communicated to drivers via their work-lists.

As a final note in this section, the performance of the system shall be briefly discussed. No controlled experiments have been done; yet some qualitative information can be given. Indeed, a performance evaluation of the system may be elaborated in terms of answering questions such as “how much time does it take to generate a solution using OMs?” and “how much time does it take to modify the pool of instances (*e.g.*, to instantiate a new workflow instance, to update/adapt already planned/instantiated ones)?” The time to adapt a workflow instance may be defined as a function of its complexity or of the complexity of the applied modification. Alternatively, a cost model can be considered to estimate the cost time (or resources) to introduce and enact basic adaptations. Based on our current prototype implementation, we encountered a performance problem that is mainly related to the continuous access to the database. In fact, some of the ADEPT API functions that are useful in our context are not implemented yet. Consequently, we sometimes had to manipulate the ADEPT database directly, especially when implementing the *Mediator* component. The performance of the system would be considerably enhanced if the functions of this component were inherently provided by the WfMS (*e.g.*, ADEPT).

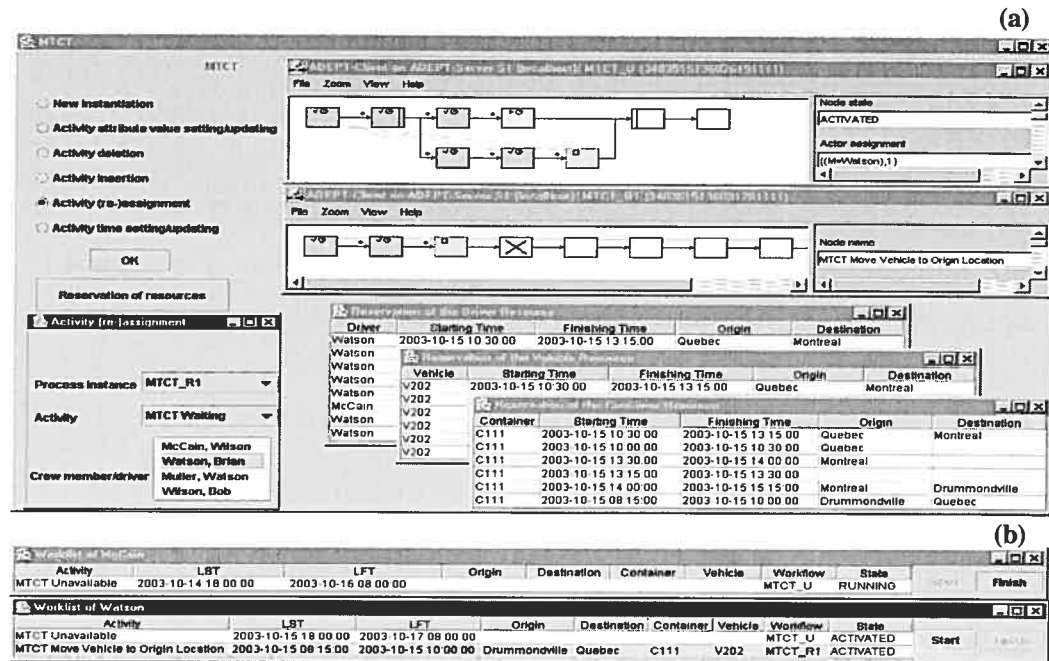


Figure 7.11. Screenshot of the MTCT System Version 0.1. (a) The Environment of the System Administrator, (b) The Environment of the Drivers

7.5 Conclusion

The experience and insights acquired with the realization of the MTCT system reach beyond this system in at least two ways:

First, many of the characteristics identified in the MTCT application can be identified as well in other applications. Hence, we may anticipate that the architecture described in this chapter can be adapted to other transportation applications. Local express-mail services and dial-a-ride services are examples of applications where the planning of activities can be solved as a Pick-up and Delivery Problem. Moreover, production systems in which assembly lines are involved could take advantage of this architecture. Indeed, in such systems, the management of limited shared resources and the management of processes are interrelated. On the one hand, the availability of resources may influence the activities scheduling within a process. On the other hand, planned processes reflect the reservation of resources.

Second, the complex MTCT application appeared to be well chosen because it allowed us to identify an interesting set of new requirements for enhanced workflow technology.

The ADEPT prototype WfMS used supports some of the investigated concepts and dynamic modifications required at the workflow instance level. Its flexibility helped in developing the MTCT system, yet its API had to be enriched with useful functionality, and workaround solutions were required to properly cope with the definition of a workflow model and with the (dynamic) management of instances. However, the implementation of the different system components would be considerably simplified and, as stated at the end of Section 7.4, the performance of the system would be substantially enhanced if these workflow concepts and functionality were inherently provided by the WfMS (*e.g.*, ADEPT).

Chapter 8 Extension of the Specification of the Workflow Reference Model

Workflow-based systems usually require the implementation of specific workflow client applications. Examples are the Workflow Monitoring and Control Tool implemented in the context of both the CONSENSUS and the MTCT systems, as well as the Rule Client and the Resource Extraction Client within the MTCT system. Moreover, applications implemented as workflow-based systems require specific workflow concepts to accommodate specific needs. Examples include the activity template concept introducing a standard way for defining activities in the context of one application, and the warm-up time concept allowing humans to be informed at the right moment about upcoming activities in the process. A prerequisite to let required functionality be correctly implemented within workflow client applications is to provide an appropriate workflow application programming interface (WAPI). This API should contain all the necessary functions allowing one to handle the workflow concepts and functionality required by the studied application.

In Chapter 6, we built up a list of such concepts and functionality. This list was motivated by the study of two complex applications. To precisely define the corresponding functions, we rely on an existing workflow API namely the established WfRM [WfMC95] of the WfMC [WfMC04] and extend it for our purposes.

In the following, a brief review of the WfRM in its current state is given (Section 8.1). The methodology used for the extension of the model and the extension itself are then presented and discussed in Section 8.2. The implementation related to the conceptual extension is presented in Section 8.3. This implementation enhances the functionality of the already existing WfMS ADEPT. Section 8.4 discusses and concludes the chapter.

Details regarding the extension of the WfRM conceptual specification are given in Appendix A. The extended specification is presented using the WfMC-description approach. This approach consists of defining the operations under their related interfaces. In addition, we provide three levels of details in Appendix A: a compressed summary, a detailed summary, and a detailed description of the extension. The detailed description groups the different operations under the functionality they allow. A UML specification of interfaces may be given instead. We opted, however, for the WfMC-description approach because we want to address the workflow community in the first place; a UML-description should be added eventually for other audiences.

8.1 Review of the Workflow Reference Model

The WfRM consists of a generic description of the structure of a WfMS, thus enabling individual specifications to be developed within its context. At the highest level, *all* WfMSs may be characterized as providing support in three functional areas [WfMC95]: (1) the build-time functions, concerned with defining and modeling the workflow process and its activities; (2) the run-time control functions, concerned with managing the workflow processes in an operational environment and with sequencing the activities to be handled as part of each process; and (3) the run-time interactions with human users and IT applications for processing the various activity steps. At a lower level, five main components are identified within the architecture of the WfRM [WfMC95]: (1) Process Definition Tools, (2) Workflow Client Applications, (3) Invoked Applications, (4) Other Workflow Enactment Services, and (5) Administration and Monitoring Tools. We introduced these components in Chapter 2. They are related to a *Workflow Enactment Service*, which ensures that the right activities are carried out in the right order and by the right people or applications. This service comprises at least one engine (the core of a WfMS, called the “workflow engine”). For the purpose of our work, we focus on the first three components:

- *Process Definition Tools* – They gather mainly the build-time functions concerned with modeling the workflow process and its constituent activities.

- *Workflow Client Applications* – They gather the run-time functions concerned with interacting with users and IT applications for completing the various activities. Work-lists that identify the work-items to be carried out by a specific user form part of this component.
- *Invoked Applications* – This component is responsible for the launching of applications associated with specific tasks.

While these three components address the main features we are concerned with during the build-time phase and the run-time phase, the two other components, that is, the “Other Workflow Enactment Services” and the “Administration and Monitoring Tools” components, deal with supplementary workflow features such as distributed workflows and workflows measurement and analysis.

Ten groups of operations (*i.e.*, API calls) support the interfaces that exist between each of the three interesting components and the Workflow Enactment Service (Table 8.1; groups (G1) to (G10)) [WfMC98]. The Workflow Enactment Service should not be confused with the fourth component of the WfRM (Other Workflow Enactment Services). As explained in Chapter 2, Section 2.4.1.1, the Workflow Enactment Service constitutes the core of a WfMS. It comprises at least one workflow engine and it provides the build-time and the run-time environments for the creation, management and execution of workflows.

Group (G1) provides two functions that allow a specific component to connect to and to disconnect from the workflow engine for a series of interactions. It is obvious that this group of operations appears in each of the three component interfaces.

Groups (G2), (G3) and (G4) are exclusively assigned to Interface 1 and gather a set of functions that deals with the definition of workflow models. Group (G2) supports the creation and the modification of a workflow process model, whereas group (G3) includes creating and deleting entities, and group (G4) allows for getting and setting the attributes of these entities. An entity is defined as a building block for a workflow definition [WfMC99a]. An activity, a transition and a participant specification are examples of entities. Note that an entity is always scoped by another entity.

Table 8.1. Groups of Operations Distributed within Interfaces 1, 2 and 3 of the WfRM

Groups of Operations	Process Definition Tools (Interface 1)	Workflow Client Applications (Interface 2)	Invoked Applications (Interface 3)
(G1) Connection Functions	✓	✓	✓
(G2) Process Modeling Functions	✓		
(G3) Entity Handling Functions	✓		
(G4) Entity Attribute Manipulation Functions	✓		
(G5) Process Control Functions		✓	
(G6) Process Status Functions		✓	
(G7) Activity Control Functions			✓
(G8) Activity Status Functions			✓
(G9) Work-list/Work-item Handling Functions		✓	
(G10) Administration Functions		✓	
<i>(G11) Classification Category Definition Functions</i>	✓		
<i>(G12) Activity Template Modeling Functions</i>	✓		
<i>(G13) Activity Template Attribute Manipulation Functions</i>	✓		

Groups (G5) to (G10) are assigned to Interfaces 2 and 3. Group (G5) allows the creation, the starting and the termination of a specific process instance, as well as the changing of its operational state. Group (G6) is intended to provide a view of current process instances allowing the verification of the work done, the work to be done, etc. Similarly, groups (G7) and (G8) allow respectively for changing the operational state of activity instances, and for providing a view at the activity instance level. We specify that groups (G5) and (G7) deal not only with process instances and activity instances, but with their attributes as well allowing the assignment of a specific value.

Group (G9) addresses work-items and allows for changing their states, reassigning them to different work-lists and assigning a specific value to their attributes.

Finally, group (G10) provides the functionality needed to perform the administration and maintenance of a workflow system. This includes functions that allow for aborting and terminating process instances.

8.2 The Proposed Extension

We now present the extension made to the WfRM to support the list of workflow concepts and functionality. Our methodology is best presented by a set of four questions: (1) Is there a need for a new component and a corresponding new interface? (2) Is there a need for a new group of operations? (3) Is there a need for new operations that will extend already existing groups of operations? (4) Which group of operations should be assigned to which interface to support a specific requirement?

No new components are added to the WfRM because the existing ones are defined on a sufficiently high level allowing for an extension within their context. Indeed, when a new concept is defined, a new group of operations is created and assigned to an existing interface. New operations are added to an existing group of operations to extend the functionality related to a specific existing concept.

All the new functions, data types and function error return codes that we define follow the naming conventions of the WfMC [WfMC97]. For example, a function name is preceded by “WM” meaning “Workflow Management”, a data type name is preceded by “WMT” meaning “Workflow Management Type” or by “WMTP” meaning “Workflow Management Type Pointer”, and function error return codes are fully capitalized. They also follow the traditional structure of the initial WAPI: components, interfaces, groups of operations, etc. Like the original WAPI specification, we do not explicitly include any requirements or provisions for process consistency. This is left up to specific implementations, and it is usually based on developed conditions ensuring the correctness of a process; *e.g.*, conditions specified in Criterion 6.1 ensuring the safe interruption of a process running activity.

8.2.1 Extension of Interface 1 (Process Definition Tools)

The extension brought to Interface 1 introduces mainly the two concepts discussed in Chapter 6: the activity template concept and the template classification. New data types are defined to support these concepts at the build-time level. A total of three new groups of operations are added: (G11), (G12), and (G13) of Table 8.1. Groups (G11) and (G12) gather respectively operations for the creation/deletion of a classification category, and operations for the creation/deletion of an activity template and for its assignment to/detraction from a classification category. Since we associate attributes with an activity template, group (G13) is also added to Interface 1. Group (G13) comprises operations that allow (1) for inserting/deleting an input/output attribute to/from an activity template already created, (2) for setting an input/output attribute of an activity template, (3) for assigning a workflow participant to an activity template, and (4) for setting a time attribute of an activity template. Time attributes comprise the duration of an activity, its starting/finishing time and its WUT. Absolute values for these time attributes are required. The definition of a new group of operations, however, is necessary when dealing with activity time attributes as a new concept. The discussion of activity temporal aspects as a concept was given in Section 6.3.3.

One may think of considering an activity template as an entity and of using the functions defined within the existing groups (G3) and (G4) instead of defining the new groups (G12) and (G13). This is not possible, however, because an entity is always scoped by another entity (cf. Section 8.1), whereas an activity template is defined as a standalone activity, not being part of any workflow definition (cf. Section 6.3.1). Hence, the defined functions in (G3) and (G4) require a scoping entity as an input parameter. A scoping entity cannot be provided for an activity template.

Finally, two operations allowing the assignment/detraction of a process definition to/from a classification category are added to the already existing group (G2) of Interface 1 (cf. Table 8.1).

8.2.2 Extension of Interface 2 (Workflow Client Applications) and Interface 3 (Invoked Applications)

The extension brought to Interfaces 2 and 3 is obviously related to the dynamic modification of process instances. The WfMC specifies that *some* WfMSs may allow dynamic alterations to process definitions from the run-time operational environment [WfMC95]. Since the run-time operational environment is involved within the second and third functional areas (cf. Section 8.1), a WfMS supporting dynamic alterations could be seen as a system that extends these two functional areas by a set of run-time process modification functions that allow users to modify instances of the original model. Indeed, a Workflow Client Application is defined as the component supporting interactions with user interface desktop functions. It is responsible, together with the Invoked Applications component, for the execution of workflow activities. Consequently, a possibility to permit the dynamic modification of process instances is to add to the interfaces that exist between each of these components and the Workflow Enactment Service a set of operations for the insertion, the moving, and the deletion of a particular entity within a workflow instance, and for the creation, the setting and the deletion of a particular entity attribute.

A number of operations are added to the three groups: (G5), (G7) and (G9). The operations added to group (G7) basically allow dynamic modifications that concern activity instances (cf. Section 6.4):

- The dynamic insertion of a new activity instance
- The dynamic deletion of an activity
- The dynamic move of an activity
- The dynamic insertion/setting/updating of input attributes
- The dynamic (re-)assignment of activities to a participant
- The dynamic setting/updating of time attributes

The “insert” operation (cf. Section 6.4.1.1) takes as input, among other things, an activity instance that corresponds to the activity to be inserted. On the one hand, the activity instance may already exist within the run-time environment. A *WMGetActivityInstance* operation, already defined within group (G6), is used so that the specific activity in-

stance to be inserted can be obtained. On the other hand, we may want to create an activity instance from an activity template. For this reason, an operation that allows this creation (*i.e.*, *WMCreateActivityInstance*) as well as operations that deal with the list of activity templates (open/close the list and fetch/get activity template from the list) are also added to group (G7).

The operations added to group (G5) deal with process instances. They allow the dynamic insertion of a block of activities (*cf.* Section 6.4.1.2), as well as the storage of the process definition that corresponds to a modified process instance. Finally, one operation is added to group (G9). It allows the deletion of a work-item from a given work-list (*cf.* Section 6.4.5). Other functions related to the dynamic management of work-lists include the reassignment of a work-item to another work-list and the update of a work-item attribute. These functions are already provided by the original specification of the WfMC [WfMC98].

8.2.3 Discussion of Already Existing Components

As pointed out in Chapter 2, the Process Definition Tools address aspects beyond the definition of processes. These aspects cover the classification of resources and the analysis of processes. Following the same idea, the Workflow Client Applications component should be extended to address functionality beyond the mere workflow monitoring and control. Hence, the resources classified at build-time should be tracked during run-time, and the workflow instances should sometimes be *automatically* monitored and controlled.

In the MTCT system discussed in Chapter 7, the Resource Extraction Client and the Rule Client are considered as Workflow Client Applications. Specific functions are required to deal with resources and rules, respectively. Connection functions are necessary to connect each of the Resource Extraction Client and the Rule Client to the workflow engine. These functions are gathered within group (G1) (*cf.* Table 8.1) already provided by the original conceptual specification. Other functions are required for the extraction of resources:

- A function to extract the resources involved in a specific workflow instance or in a specific workflow activity.
- A function to get the reservation of resources at a specific time.
- A function to get the shift of a specific resource.

Functions required in the context of the Rule Client are usually provided by the API of the underlying rule engine (*e.g.*, ILOG JRules used in the MTCT system). They do not need to be supported on the WAPI side. Examples of such functions are:

- A function that *gives as input* a rule base to the Rule Client.
- A function that *parses* the rule base.
- A function that *creates an execution context* (which contains initially the entire rule set).
- A function that *asserts* the process templates, the process instance manager, and the attributes to the context.
- A function that *fire rules* in the context.

8.3 Functionality Extension of a WfMS

The implementation of specific workflow clients for the studied applications has necessitated the implementation of the set of requirements identified in Chapter 6 and to which no direct or workaround solutions could be found in ADEPT.

Not considering the minimal difference in the structure and interplay of functions, the ADEPT API provides most of the functionality requested by the original specification of the WfRM (the WAPI). Moreover, it offers additional features that correspond to some of the functionality studied in Chapter 6: (1) the activity template concept and (2) functions for structural modifications (insertion and deletion).

All new functions we implemented are collected in a *Mediator* component that extends the existing ADEPT API and contributes to the current pool of available functions by running in parallel to ADEPT (*cf.* Chapter 7; Figure 7.10). The ADEPT Client uses the extended Interfaces 2 and 3 by accessing the Mediator functions as well as the original functions in the ADEPT Server. It should be stated that this solution could not be a de-

finitive one; it was merely adopted for simplicity. For further genuine implementations, the ADEPT API must be extended by modifying the source code of the system.

More details regarding the implemented functions of the Mediator component are given below. The current state of the implementation extending ADEPT covers mainly the dynamic modification of activity attributes (Section 8.3.2) and the dynamic management of work-lists (Section 8.3.3). A check for the compliance of the already supported structural modifications (the dynamic insertion and the dynamic deletion of an activity) to the extended WfRM was successful (Section 8.3.1). A discussion of the current implementation that extends ADEPT is finally given (Section 8.3.4).

8.3.1 Structural Modifications

The *WMInsertActivityInstance* and the *WMDeleteActivityInstance* functions were therefore realized with a reasonable effort by calling respectively the *dynamicInsert* function and the *dynamicDelete* function from the ADEPT API. In our conceptual specification, an activity template must be instantiated first to obtain an activity instance that then can be inserted. In the ADEPT API, *dynamicInsert* takes as a parameter the activity instance to be inserted. We, however, do not implement the *WMCreateActivityInstance* function in this context. We simply apply the workaround solution described in Section 6.4.1.1. It consists of defining an activity template within a workflow model *W* such that the activity instance to be inserted can be created by instantiating *W*. We consciously accept this difference between our conceptual specification and the actual implementation.

8.3.2 Activity Attributes Modification

The functions related to the dynamic modification of activity attributes that have been implemented are the following:

- *WMAssignActivityInstanceAttribute* – This function is responsible for dynamically inserting an attribute or setting/updating its value in an activity instance. In case the attribute provided as a parameter to the function already exists, its definition is changed according to the provided parameters: attribute type and attribute length. In case the specified attribute does not exist yet, it is added to the named activity instance. If there is a value submitted in the corresponding pa-

parameter of the function, it is set or updated within the attribute. For consistency reasons some internal checks with the database are done to ensure that the inserted or updated values comply with the specified types (*e.g.*, string/long).

- *WMDeleteActivityInstanceAttribute* – This function dynamically deletes an activity instance attribute. If we are dealing with an input attribute, this attribute is simply removed from the corresponding activity instance. If, however, we are dealing with an output attribute O, all input attributes consuming from O are deleted in every activity instance in the sequel to ensure consistency on a basic level.
- *WMAssignActivityInstanceParticipants* – This function dynamically sets or updates the participant(s) assigned to an activity instance. According to the WfMC standard, up to ten participants can be assigned to one activity instance [WfMC98].
- *WMAssignActivityInstanceDuration* – This function dynamically sets or updates the minimum or maximum duration of an activity instance. Only an absolute value, *i.e.*, the number of minutes, can be submitted for the duration in the corresponding parameter of the function.
- *WMAssignActivityInstanceTime* – This function dynamically sets or updates the earliest/latest starting/finishing time of an activity instance. Only an absolute value, *i.e.*, a fixed date, can be submitted for the time in the corresponding parameter of the function.

The WUT is not practically set/updated using the function defined within the conceptual specification. A workaround solution has been proposed in Section 6.3.3.3.

8.3.3 Work-lists Management

The function related to the dynamic management of work-lists that has been implemented is the *WMDeleteWorkitem*. This function dynamically deletes a work-item from a work-list. The work-item should not be in a running state. The deletion of an activity in a running state, studied formally in Section 6.4.2.1, has not been implemented yet.

8.3.4 Discussion of Current Implementation

The ADEPT API has not been designed to add new functionality such as presented above. Consequently, it was impossible to implement our desired functions relying exclusively on the ADEPT API. Sometimes, we were obliged to directly access the database where ADEPT stores the workflows and their related data. Thus, we evade all consistency checks within ADEPT. Although some basic measures have been taken to ensure consistency, there are still some leaks such that our approach cannot be considered a complete solution that ensures correctness and consistency. Therefore, with the current implementation, the responsibility is shifted to the user of the functions to invoke the latter wisely. Hence, the implementation of the CONSENSUS and the MTCT workflow clients include many verifications defined in the context of the respective applications. These verifications are made before invoking a specific function. As an example, in the context of the MTCT application, verifications regarding time consistencies for complete transportation solutions are done by the optimization part of the MTCT system. Such transportation solutions are reflected by specific workflow instances. The *WMAAssignActivityInstanceDuration* function (resp. the *WMAAssignActivityInstanceTime* function), when invoked on these instances, takes the duration (resp. the time) given by the calculated consistent solution.

8.4 Conclusion

We proposed in this chapter an extension to the WfRM specification. The reference model initially provides a basic architecture that can be used as a standard for the development of a WfMS. The discussed extension mainly proposes a set of functions addressing the concepts and functionality studied in Chapter 6.

On the one hand, new data types are defined to support the two concepts: the workflow template concept and the template classification. Functions for the manipulation of these concepts are defined within new groups of operations assigned to Interface 1.

On the other hand, functions for the dynamic modification of process instances are defined and distributed within existing groups of operations (Interfaces 2 and 3). These functions mainly support the creation, the insertion, and the deletion of a particular ac-

tivity instance, as well as the insertion, the assignment, and the deletion of a particular activity instance attribute during run-time at the process instance level. We highlight the fact that the “Entity Handling Functions” group and the “Entity Attribute Manipulation Functions” group assigned to Interface 1 gather similar functions. Indeed, the creation, the retrieval, and the deletion of a particular entity, as well as the retrieval, the setting, and the deletion of a particular entity attribute are possible during build-time at the process level.

We think that the separation that is made between the operations of Interface 1 and those of Interfaces 2 and 3 should be removed to allow the dynamic modification of process instances. A similar argumentation is given by Han and Sheth in [HS98]. The authors talk rather about the separation that exists between build-time and run-time in terms of workflow models. They specify that this is a barrier to be removed to allow the adaptation of workflows.

It is, however, obvious that during run-time we need to be stricter than during build-time regarding the preservation of workflow consistency and correctness. Indeed, the operations of Interface 1 deal with the modeling of workflows. Usually, at build-time, the consistency and correctness verifications are only checked once the workflow model is completely defined and ready to be saved as a model to be instantiated. At run-time, these verifications are done more frequently. Each time a modification is brought to a workflow instance, verifications are done and the modification is forbidden, if the consistency and the correctness of the model are violated.

The extension that we proposed in this chapter distinguishes between build-time and run-time operations. It was made this way because of two reasons: first, to respect the initial conceptual specification of the WfRM that separates the build-time and the run-time functional areas, and second, to emphasize the differences that exist between the build-time and the run-time phases regarding the frequency of the consistency and the correctness verifications.

In this chapter, we sum up and discuss our work by referring to the research objectives and major contributions exposed in Chapter 1. Then, we detail further research issues that need to be addressed in future work.

9.1 Summary and Discussion

Workflow technology offers little adequate support to requirements inherent to non-trivial socio-technical systems. In this thesis, we studied two applications that call for such systems: the combined negotiation application and the multi-transfer container transportation application. These applications have served to investigate the needs for a clarified and a refined set of concepts and functionalities for workflow management systems. This set was motivated, on the one hand, by the requirements of the two applications and their respective support systems towards workflow technology, and on the other hand, by the constraints of today's WfMSs with respect to these applications and systems. In the following, a systematic summary of the thesis is provided and a discussion of remaining issues is provided.

9.1.1 The CONSENSUS and the MTCT Applications as Drivers of Sophisticated Requirements for Workflow Technology

The combined negotiation support system (CONSENSUS) based on a WfMS was studied in detail. This system was developed to help a user model and enact a specific kind of e-negotiations: combined negotiations. A combined negotiation is modeled as a workflow that captures the sequencing of individual negotiations as well as the dependencies among them. At run-time, software agents participate in negotiations as actors in the workflow. It appeared that this system requires support for dynamic modifications induced by unexpected events that can occur during negotiations. We realized that current

WfMSs such as IBM MQ Series Workflow and BEA's WLPI, support in a limited way this kind of dynamism, slightly reducing the benefits of the workflow-based CONSENSUS approach to e-negotiations.

Another complex socio-technical application, the multi-transfer container transportation (MTCT) application, exhibits inherently dynamic requirements for workflows. A workflow-oriented system for the processing of customer requests for container transportation was devised. This processing is achieved by specific sequences of interdependent activities that need to be created just-in-time and then (automatically) adapted to deal with unexpected events that may occur. The creation and the adaptation of activity sequences are based on an optimized resource management and activity scheduling.

In the first system, the integration of a WfMS (ADEPT) that supports some of the required dynamic modifications at the workflow instance level increases the benefits of the CONSENSUS approach. In the second system, the ADEPT WfMS prototype has been used as well. Its flexibility helped in designing the MTCT system, yet its API had to be enriched with useful functionality, and new solutions were sometimes required to properly cope with the definition of workflow models and with the (dynamic) management of instances.

9.1.2 The Identification and the Accommodation of Sophisticated Requirements for Workflow Technology

The experience and insights acquired with the realization of these two applications go beyond the CONSENSUS and the MTCT projects in leading to the "wish list" of clarified and refined workflow concepts and functionalities (cf. Chapter 6). Each of these concepts and functionalities has been studied and corresponding solutions have been proposed.

Indeed, direct solutions that respectively address the activity template concept and the activity duration were possible using a state-of-the-art WfMS (ADEPT). Workaround solutions were, however, proposed in ADEPT to support other concepts namely the template classification, the activity starting/finishing time, and the activity warm-up time concept:

- The template classification is usually offered in commercial WfMSs. It is however missing in ADEPT. Saving workflow templates and activity templates with a specific prefix remedies this lack.
- In the literature, a distinction is done between dependant dates between activities and absolute dates assigned to activities as a starting/finishing time. On the one hand, while not supported by commercial WfMSs, dependant dates are well defined in ADEPT. On the other hand, absolute dates are not support by ADEPT although they are less complicated to deal with when compared to dependant dates. We found a solution based on dependant dates to cover absolute dates in ADEPT.
- The warm-up time (WUT) concept is of utmost importance. To our knowledge, it is not supported yet by any WfMS. The WUT of an activity should be known such that early information about this activity is provided at the right time to the right workflow participant. Preparation activities were proposed to support this concept. Nevertheless, two noticeable shortcomings were recognized: the lack of a *just-in-time* notification and the complication of the workflows. We tried to deal with the first shortcoming by proposing an intermediate work-list with a listener process, and with the second shortcoming by suggesting to define the preparation activities in the background of the initial workflow (*i.e.*, the workflow not including preparation activities), or to separate between the initial workflow and the workflow that defines preparation activities.

The basic dynamic activity insertion and dynamic activity deletion functionalities are already well discussed in the literature and direct solutions can be found in adaptive WfMSs such as in ADEPT. A refinement of both functionality is however required in the context of complex, yet representative, process-oriented applications:

- We discussed solutions for the dynamic insertion of a *new* activity instance and for the dynamic insertion of a *block of activities*. The activity template concept is used to accommodate the insertion of a new activity instance. Hence, this remedies the “write after write” problem encountered when an already existing activity instance is inserted within the workflow instance. The dynamic insertion of a

block of activities is accomplished step-by-step using the defined dynamic insertion of a single activity operation. Many problems were identified to this solution including the high number of interaction with the system and the lack of operations allowing for the insertion of decision nodes present in complex modeling structures.

- An extension of an existing formal meta-model, that is the WSN-Nets formalism, was elaborated to support a refinement of the dynamic activity deletion functionality: the safe activity interruption in case of exceptional situations. This novel functionality corresponds to the deletion of an activity in a running state. The latter was not tolerated yet in the current adaptive workflow technology. The support of this functionality appears to be, however, extremely important because most exceptional situations occur while an activity is in progress, and adequate solutions need to be provided in the sequel. We may talk about a *forward recovery*. For this purpose, besides modeling logical work units as process activities, we have introduced another level of granularity by defining the atomic step concept. The latter is used to build up the basis for a two-dimensional data classification scheme. On the one hand, the definition of the data relevance dimension, distinguishing between exclusive application data and process relevant data, is considered at its pure level within the safely interruption criterion conditions statement. On the other hand, we dug deeper regarding the data update frequency dimension by defining safe interrupt points for each of the discrete and the continuous data update by activities. This has led to the formal definition of the activity safe point considered as the backbone for the safely interruption criterion. Preserving this criterion, in turn, guarantees that if an activity is safely interrupted all necessary data is kept and can be used to figure out an adequate solution for the respective exceptional situation.
- The dynamic move of an activity was replaced by a workaround solution: inserting the activity to be moved at its new position and then deleting this activity. Incorrect data flow conflicts may be detected if we apply these two operations in the opposite way (*i.e.*, the deletion before the insertion). We specified that a re-

laxation should be done to the consistency verifications applied for the delete operation in the context of a move operation.

The functionalities already discussed tackle workflow structural modifications. Moreover, in this thesis, we motivated and we analyzed attribute modifications, namely the insertion and the deletion of activity attributes:

- The insertion of activity attributes *on the fly* is motivated by the unavailability of the information required for the definition of these attributes at the workflow modeling level. The activity to which an attribute is to be inserted should not be in a running state.
- The activity attribute deletion operation is mainly required so that an activity deletion operation is not needlessly forbidden. A distinction is done between the deletion of input attributes and the deletion of output attributes. The former does not necessitate any consistency verification, while the latter requires data dependency verifications comparable to ones carried out in the context of an activity deletion.

The setting and updating of (time) attributes as well as the (re-)assignment of activities to participants at the workflow execution level were addressed. The issue of properly implementing these functionalities in a specific workflow client was discussed. As a consequence to the dynamic modification of attributes, the appropriate adaptation of work-lists is considered.

Finally, it has been argued that each of the discussed functionality may require to be applied either manually or automatically to a workflow instance, and that a WfMS should facilitate the integration of a tool for the automatic application of this functionality.

9.1.3 The Extension of the WfRM to Adequately Support Enhanced Workflow Technology

An extension of the Workflow Reference Model (WfRM) has been proposed to accommodate the refined set of workflow concepts and functionality. First, we have extended the WfRM by proposing a new overall architecture framework for adaptive workflows.

Second, we have expanded the existing specification by defining black box functions. The latter were either assigned to an existing group of operations or they were gathered under new groups of operations. The extended WfRM should facilitate the implementation of original WfMSs or the review of existing WfMS versions leading to enhanced systems.

9.1.4 Further Discussion

When studying a specific functionality, we realized that novel workflow concepts must emerge. Indeed, the definition of the atomic step concept was essential to formally specify the criterion behind the safe activity interruption functionality. It is, hence, important to keep in mind that novel workflow concepts may be discovered not only when applying the direct approach of studying a specific application, but also indirectly from deeply addressing a required workflow functionality.

We think that state-of-the-art WfMSs should provide innovative workflow concepts and functionality but without forgetting about basic ones. In spite of the fact that ADEPT covers interesting workflow concepts (activity template, time edges, etc.) and it provides advanced functionality (the dynamic activity insertion and deletion), we observed that it lacks to offer some of the basic WfMSs features such as the template classification and the support of absolute dates. Unfortunately, though simple, these features are desirable for the development of many workflow-based applications.

9.2 Research Perspectives

We are firmly convinced that the following stimulating problems need to be formally addressed. The results should then be implemented within a powerful workflow engine:

- Various research groups have already formally studied the consistency verifications related to basic workflow structural modifications: the insertion and the deletion operations. This is done based on specific formalisms such as the WSM-Nets formalism. This formalism or a similar one should be used as a basis to formally specify the criteria for a correct application of the refined structural modification functionality that were identified and informally addressed within

this thesis: the insertion of a *new* activity instance, the insertion of a *block of activities*, and the move of an activity. The safe activity interruption functionality was already formally addressed within this thesis. There are, however, still interesting questions concerning the implementation of this functionality: questions related to modification authorization, modification analysis, and usability.

- The WUT concept should be formally studied taking into account the solutions proposed to deal with the shortcomings of the current workaround solution.

The functions related to the dynamic modification of activity attributes that we now provide via a *Mediator* component running in parallel to ADEPT, need to be implemented within the core of ADEPT.

If implemented within ADEPT, the above features may judiciously contribute to the “Next Generation Enterprise Process Management System” project launched by the DBIS department at University of Ulm [DBIS04].

Extended transactional issues (*e.g.*, semantic rollback) may be studied as well. Indeed, at the workflow modification level, only forward recoveries were addressed in this thesis. As an example, if a problem occurs, the current activity is interrupted and a new solution is proposed in the sequel of the workflow instance. Rolling-back issues are interesting to be studied as well. For example, once an activity is interrupted, a backward recovery is proposed to cope with the triggering event. A rollback (*i.e.*, backward recovery) is specifically interesting if no safe interruption of an activity is possible. As an example, if the interruption of the “move (1.5,3.5) \rightarrow (13,8)” activity in Figure 6.15 (cf. Chapter 6) is not safe, a solution could be to go back and (1) to return the merchandise to the origin location, or (2) to keep the merchandise in a depot as long as no delivery solution is possible. This can be done using compensation activities. An example of a compensation activity in the context of the combined negotiation application is “breaking the commitment” of an already committed “e-negotiation” activity (cf. Chapter 4). The discussed facilities are crucial for realizing real-world adaptive enterprise applications.

Finally, several interesting issues in the context of the MTCT system should be investigated. Among these issues is the support of unexpected events such as delayed vehicles,

crew member desistance and technical problems. The only event supported up to now by the MTCT system is the “arrival of a new customer request”. Another issue is the distributed work-lists that should be investigated to dispatch work on a network of several computers, which could be located at different terminals/vehicles. Modification rules are another important research issue. New rules that would bring structural modifications to workflow instances should be developed. It is interesting to define more complex optimization models taking into account complex path scenarios. Solutions coming from these optimization models will potentially be translated into novel and challenging structural modifications of workflow instances. Finally, at the implementation level, the performance of the MTCT system would be considerably enhanced if the functions of the *Mediator* component were inherently provided by the WfMS on which the system relies.

We rigorously encourage researchers to deal with each of the above research perspectives. Moreover, we encourage them to study further practical applications to discover and to propose solutions for additional factual needs for workflow technology. We strongly believe that the enhancement of any technology should mainly derive from practice.

References

- [Aal00] van der Aalst, W.M.P., Loosely Coupled Interorganizational Workflows: Modeling and Analyzing Workflows Crossing Organizational Boundaries. *Information and Management*, 37(2):67-75, March 2000.
- [Alg00] Algorithms and Theory of Computation Handbook, *CRC Press LLC*, 1999. Appearing in the Dictionary of Computer Science, Engineering and Technology, *CRC Press LLC*, 2000.
- [AAE+96] Alonso, G., Agrawal, D., El Abbadi, A., Kamath, M., Günthör, R., and Mohan, C., Advanced Transaction Models in Workflow Contexts. In *Proceedings of the 12th International Conference on Data Engineering (ICDE'96)*, 574-581, New Orleans, LA, February 1996.
- [AAH98] Adam, N.R., Atluri, V., and Huang, W.K., Modeling and Analysis of Workflows using Petri Nets. *Journal of Intelligent Information Systems*, 10(2):131-158, March 1998.
- [AB02] van der Aalst, W.M.P., and Basten, T., Inheritance of Workflows: An Approach to Tackling Problems Related to Change. *Theoretical Computer Science*, 270(1-2):125-203, 2002.
- [ABE+00] van der Aalst, W.M.P., Barthelmess, P., Ellis, C.A., and Wainer, J., Workflow Modeling using Procllets. In *Etzion, O. and Scheuermann, P. (Eds.): Proceedings of the Fifth International Conference on Cooperative Information Systems (CoopIS'00)*, 198-209, Eliat, Israel, September 2000. LNCS 1901.
- [AH02] van der Aalst, W.M.P. and van Hee, K., Workflow Management: Models, Methods, and Systems. *The MIT Press*, 368 pp., 2002. ISBN 0-262-01189-1.
- [AK00] van der Aalst, W.M.P. and Kumar, A., XML Based Schema Definition for Support of the Inter-organizational Workflow. In *Proceedings of the 21st International Conference on Application and Theory of Petri Nets (ICATPN'00)* (Meeting on XML/SGML based Interchange Formats for Petri Nets), Aarhus, Denmark, June 2000. On-line at <http://www.daimi.au.dk/pn2000/Interchange/papers/det_01.pdf>.
- [AM97] Alonso, G. and Mohan, C., Workflow Management Systems: The Next Generation of Distributed Processing Tools. In *Jajodia, S. and Kerschberg, L. (Eds.): Advanced Transaction Models and Architectures*, Chapter 1, 35-62, Kluwer Academic Publishers, 1997.

- [AM00] Agostini, A., and De Michelis, G., Improving Flexibility of Workflow Management Systems. In *van der Aalst, W.M.P., Desel, J., and Oberweis, A. (Eds.): Business Process Management – Models, Techniques, and Empirical Studies*, 218-234, LNCS 1806 Springer-Verlag, 2000.
- [Bla00] Blake, M.B., WARP: An Agent-Based Process and Architecture for Workflow-oriented Distributed Component Configuration. In *Proceedings of the 2000 International Conference on Artificial Intelligence (ICAI'00)* (Session on Software Agent-Oriented Workflow), 205-213, Las Vegas, NV, June 2000.
- [Bla02] Blake, M.B., An Agent-Based Cross-Organizational Workflow Architecture in Support of Web Services. In *Proceedings of the 11th International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02)*, Pittsburgh, PA, June 2002. IEEE. On-line at <http://www.cs.georgetown.edu/~blakeb/pubs/blake_WETICE2002_final.pdf>.
- [BAK01] Benyoucef, M., Alj, H., and Keller, R.K., An Infrastructure for Rule-Driven Negotiating Software Agents, In *Proceedings of the 12th International Workshop on Database and Expert Systems Applications (DEXA'01)*, 737-741, Munich, Germany, September 2001. IEEE. Presented in 2nd e-Negotiations Workshop.
- [BAL+02] Benyoucef, M., Alj, H., Levy, K., and Keller, R.K., A Rule-driven Approach for Defining the Behavior of Negotiating Software Agents. In *Proceedings of the Fourth International Conference on Distributed Communities on the Web (DCW'02)*, 165-181, Sydney, Australia, April 2002. Springer-Verlag. LNCS.
- [BAV+01] Benyoucef, M., Alj, H., Vézeau, M., and Keller, R.K., Combined Negotiations in E-Commerce: Concepts and Architecture. *Electronic Commerce Research Journal*, 1(3):277-299, July 2001. Special Issue on Theory and Application of Electronic Market Design. Baltzer Science Publishers.
- [BBK01] Benyoucef, M., Bassil, S., and Keller, R.K., Workflow Modeling of Combined Negotiations in E-Commerce. In *Proceedings of the Fourth International Conference on Electronic Commerce Research (ICECR-4)*, 348-359, Dallas, TX, November 2001.
- [BBK+02a] Bassil, S., Benyoucef, M., Keller, R.K., Kropf, P., Addressing Dynamism in E-negotiations by Workflow Management Systems. In *Proceedings of the 13th International Workshop on Database and Expert Systems Applications (DEXA'02)*, 655-659, Aix-en-Provence, France, September 2002. IEEE. Presented in 3rd e-Negotiations Workshop.
- [BBK+02b] Bassil, S., Bourbeau, B., Keller, R.K., and Kropf, P., Fleet Management and Dynamic Workflows. *Technical Report GELO-152*, Université de Montréal, Canada, June 2002.

- [BBK+03] Bassil, S., Bourbeau, B., Keller R.K., and Kropf, P., A Dynamic Approach to Multi-transfer Container Management, In *Proceedings of the Second International Workshop on Freight Transportation and Logistics (ODYSSEUS'03)*, Mondello (Palermo), Sicily, Italy, May 2003.
- [BKK04] Bassil, S., Keller R.K., and Kropf, P., A Workflow-Oriented System Architecture for the Management of Container Transportation, In *Proceedings of the Second International Conference on Business Process Management (BPM'04)*, 116-131, Potsdam, Germany, June 2004. LNCS 3080.
- [BKL+00] Benyoucef, M., Keller, R.K., Lamouroux, S., Robert, J., and Trussart, V., Towards a Generic E-Negotiation Platform, In *Proceedings of the Sixth International Conference on Re-Technologies for Information Systems*, 95-109, Zurich, Switzerland, February 2000.
- [BPEL03] Business Process Execution Language for Web Services version 1.1 (2003). On-line at <http://www-128.ibm.com/developerworks/library/specification/ws-bpel/>.
- [BRK+05] Bassil, S., Rinderle, S., Keller, R.K., Kropf, P., and Reichert, M., Preserving the Context of Interrupted Business Process Activities, In *Proceedings of the Seventh International Conference on Enterprise Information Systems (ICEIS'05)*, Miami, FL, May 2005. To appear.
- [Cra02] Crainic, T.G., Long-Haul Freight Transportation. In *R.W. Hall (Ed.): Handbook of Transportation Science*, Second Edition, Kluwer Academic Publishers, 2002.
- [CCP+98] Casati, F., Ceri, S., Pernici, B., and Pozzi, G., Workflow Evolution. *Data and Knowledge Engineering*, 24(3):211-238, 1998.
- [CHR+98] Cichocki, A., Helal, A., Rusinkiewicz, M., and Woelk, D. (Eds.), Workflow and Process Automation: Concepts and Technology. *Kluwer Academic Publishers*, Vol. 432, 136 pp., 1998. ISBN 0-7923-8099-1.
- [CKL+03] Curbera, F., Khalaf, R., Leymann, F., and Weerawarana, S., Exception Handling in the BPEL4WS Language. In *Proceedings of the International Conference on Business Process Management (BPM'03)*, 276-290, Eindhoven, The Netherlands, June 2003. LNCS 2678.
- [CKO92] Curtis, B., Kellner, M., and Over J., Process Modeling. *Communications of the ACM*, 35(9):75-89, September 1992.
- [CKW+04] Chiu, D.K.W., Kwok, B.W.C., Wong, R.L.S., Cheung, S.C., Kafeza, E., and Kafeza, M., Alerts for Healthcare Process and Data Integration. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS-37)*, Big Island, Hawaii, January 2004.
- [CS96] Chang, J.W. and Scott, C.T., Agent-based Workflow: TRP Support Environment (TSE). In *Proceedings of the Fifth International World Wide Web Conference*, Paris, France, May 1996. On-line at http://www5conf.inria.fr/fich_html/papers/P53/Overview.html.

- [DBIS04] Abteilung Datenbanken und Informationssysteme – Universität Ulm (2004). On-line at <<http://www.informatik.uni-ulm.de/dbis/index-en.htm>>.
- [DNR90] Downson, M., Nejme, B., and Riddle, W., Concepts for Process Definition Support. In *Proceedings of the Sixth International Software Process Workshop*, 87-90, Hakodate, Japan, October 1990. IEEE Computer Society Press.
- [DR98] Dadam, P. and Reichert, M., The ADEPT WfMS Project at the University of Ulm. In *Proceedings of the First European Workshop on Workflow and Process Management (WPM'98)* (Workflow Management Research Projects), Zurich, Switzerland, October 1998. On-line at <<http://www.informatik.uni-ulm.de/dbis/papers/abstracts/DaRe98.ps.html>>.
- [DRK00] Dadam, P., Reichert, M., and Kuhn, K., Clinical Workflows – The Killer Application for Process-oriented Information Systems? In *Proceedings of the Fourth International Conference on Business Information Systems (BIS'00)*, 36-59, Poznan, Poland, April 2000.
- [Elm92] Elmagarmid, A.K. (editor), Database Transaction Models for Advanced Applications, *Morgan Kaufmann Publishers*, 610 pp., 1992. ISBN 1-55860-214-3.
- [Ens98] FileNet Ensemble User Guide. FileNet Corp., Costa Mesa, California, 1998.
- [EK00] Ellis, C.A. and Keddara, K., A Workflow Change Is a Workflow. In *van der Aalst, W.M.P., Desel, J., and Oberweis, A. (Eds.): Business Process Management – Models, Techniques, and Empirical Studies*, 201-217, LNCS 1806 Springer-Verlag, 2000.
- [EKR95] Ellis, C.A., Keddara, K., and Rozenberg, G., Dynamic Change within Workflow Systems. In *Proceedings of the Conference on Organizational Computing Systems (OCS'95)*, 10-21, Milpitas, California, 1995. ACM Press.
- [EM97] Ellis, C. and Maltzahn, C., The Chautauqua Workflow System. In *Proceedings of the 30th International Conference on System Sciences (HICSS'97)*, 427-437, Maui, HI, January 1997.
- [EP02] Eder, J. and Pichler, H., Duration Histograms for Workflow Systems. In *Proceedings of the Working Conference on Engineering Information Systems in the Internet Context (EISIC'02)*, 239-253, Kanazawa, Japan, September 2002.
- [EPG+03] Eder, J., Pichler, H., Gruber, W., and M. Ninaus, Personal Schedules for Workflow Systems. In *Proceedings of the International Conference on Business Process Management (BPM'03)*, 216-231, Eindhoven, The Netherlands, June 2003. LNCS 2678.
- [EPP+99] Eder, J., Panagos, E., Pezewaunig, H., and Rabinovich, M., Time Management in Workflow Systems. In *Proceedings of the Third International*

- Conference on Business Information Systems (BIS'99)*, 265-280. Poznan, Poland, April 1999. (Springer-Verlag)
- [FH92] Feiler, P.H. and Humphrey, W.S., Software Process Development and Enactment: Concepts and Definitions. *Technical Report SEI-92-TR-004*, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1992. On-line at <<http://www.sei.cmu.edu/publications/documents/92.reports/92.tr.004.html>>.
- [GGP+98] Gendreau, M., Guertin, F., Potvin, J.-Y., and Séguin, R., Neighborhood Search Heuristics for a Dynamic Vehicle Dispatching Problem with Pick-ups and Deliveries. *Technical Report CRT-98-10*, Centre de Recherche sur les Transports, Université de Montréal, Canada, 1998.
- [GHS95] Georgakopoulos, D., Hornick, M., and Sheth, A., An Overview of Workflow Management: From Process Modeling to Workflow Automation Infrastructure. *Distributed and Parallel Databases*, 3(2):119-153, April 1995.
- [GR01] Groote, J.F. and Reniers, M.A., Algebraic Process Verification. In *Bergstra, J.A., Ponse, A., and Smolka, S.A. (Eds.): Handbook of Process Algebra*, 1151-1208, Amsterdam, The Netherlands, 2001.
- [GS87] Garcia-Molina, H., Salem, K., Sagas. In *Proceedings ACM SIGMOD International Conference on Management of Data*, 249-259, San Francisco, CA, May 1987.
- [GT98] Georgakopoulos, D. and Tsalgatidou, A., Technology and Tools for Comprehensive Business Process Lifecycle Management. In *Dogac, A., Kalinichenko, L., Ozsu, T., and Sheth, A. (Eds.): Workflow Management Systems and Interoperability*. NATO SI Series F. Springer-Verlag, 1998. On-line at <<http://cgi.di.uoa.gr/~afrodite/nato.pdf>>.
- [Har87] Harel, D., State Charts: A Visual Formalism for Complex Systems. *Science of Computer Programming*, 8(3):231-274, June 1987.
- [Hen88] Hennessy, M., Algebraic Theory of Processes. *The MIT Press*, 272 pp., 1988. ISBN 0-262-08171-7.
- [Hol97] Hollingsworth, D., Workflow – A Model for Integration. *ICL Systems Journal*, 12(2):213-232, November 1997.
- [HH99] Handl, D. and Hoffmann, H.-J., Workflow Agents in the Document-centered Communication in MALL2000 Systems. In *Proceedings of the First International Workshop on Agent-Oriented Information Systems (AOIS'99)*. Seattle, WA, May 1999. On-line at <<http://www.aois.org/99/handl.html>>.
- [HJ98] Horn, S., and Jablonski, S., An Approach to Dynamic Instance Adaption in Workflow Management Applications. In *Proceedings of the Workshop Towards Adaptive Workflow Systems at the Conference on Computer Supported Cooperative Work (CSCW'98)*, ACM Press, Seattle, WA, November 1998. On-line at <<http://ccs.mit.edu/klein/cscw98/paper21/>>.

- [HS98] Han, Y. and Sheth, A., A Taxonomy of Adaptive Workflow Management. In *Proceedings of the ACM Conference on Computer-Supported Cooperative Work (CSCW'98)*, Seattle, WA, November 1998. On-line at <<http://ccs.mit.edu/klein/cscw98/paper03/>>.
- [Ibm04] IBM Software – WebSphere MQ Workflow (2004). On-line at <<http://www-306.ibm.com/software/integration/wmqwf/>>.
- [Inc02] TIBCO InConcert Concepts. *TIBCO Software Inc.*, Cambridge, MA, January 2002.
- [Int04] InterNeg (2004). On-line at <<http://interneg.carleton.ca/interneg/>>.
- [JRules04] ILOG JRules (2004). On-line at <<http://www.ilog.com/products/jrules/>>.
- [JB96] Jablonski, S. and Bussler, C., Workflow Management – Modeling Concepts, Architecture and Implementation. *International Thompson Computer Press*, 351 pp., 1996. ISBN 1850322228.
- [Kra00] Kradofer, M., A Workflow Metamodel Supporting Dynamic, Reuse-Based Model Evolution. *Doctoral Thesis*, University of Zurich, Switzerland, 2000.
- [KBB98] Kammer, P.J., Bolcer, G.A., and Bergman, M., Requirements for Supporting Dynamic Adaptive Workflow on the WWW. In *Proceedings of the Workshop on Adaptive Workflow Systems (CSCW'98)*, Seattle, WA, November 1998. On-line at <<http://www.ags.uci.edu/~pkammer/papers/cscw98.pdf>>.
- [KG99] Kradofer, M. and Geppert, A., Dynamic Workflow Schema Evolution Based on Workflow Type Versioning and Workflow Migration. In *Proceedings of the Fourth International Conference on Cooperative Information Systems (CoopIS'99)*, 104-114, Edinburgh, UK, September 1999. IEEE Computer Society Press.
- [KSS02] Kelton, W.D., Sadowski, R.P., and Sadowski, D.A., Simulation with Arena. *McGraw-Hill*, 631 pp., 2002 (2nd edition). ISBN 0-07-239270-3.
- [KZ02] Kumar, A. and Zhao, J.L., Workflow Support for Electronic Commerce Applications. *Decision Support Systems*, 32(3):265-278, May 2002. On-line at <<http://shell.bpa.arizona.edu/~lzhao/dss02-print.pdf>>.
- [Ley95] Leymann, F., Supporting Business Transactions via Partial Backward Recovery in Workflow Management Systems. In *Proceedings of the Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'95)*, 51-70, Dresden, Germany, 1995.
- [LO01] Lenz, K., and Oberweis, A., Modeling Interorganizational Workflows with XML Nets. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, January 2001. On-line at <<http://csdl.computer.org/comp/proceedings/hicss/2001/0981/07/09817052.pdf>>.

- [LR99] Leymann, F. and Roller, D., Production Workflow, Concepts and Techniques. *Prentice-Hall PTR*, 479 pp., 1999. ISBN 0-130-21753-0.
- [LS97] Lei, K. and Singh, M., A Comparison of Workflow Metamodels. In *Proceedings of the Workshop on Behavioral Modeling and Design Transformations: Issues and Opportunities in Conceptual Modeling at the 16th International Conference on Conceptual Modeling / the Entity Relationship Approach (ER'97)*, Los Angeles, CA, November 1997. On-line at <<http://osm7.cs.byu.edu/ER97/workshop4/lis.html>>.
- [Man99] Mann, J., Workflow and Enterprise Application Integration. *EAI Journal*, 49-53, September/October 1999. On-line at <http://www.bijonline.com/PDF/mann_1.pdf>.
- [Mar01] Marjanovic, O., Methodological Considerations for Time Modeling in Workflows. In *Proceedings of the 12th Australasian Conference on Information Systems (ACIS'01)*, Coffs Harbour, Australia, December 2001.
- [McC92] McCready, S., There is More than One Kind of Workflow Software. *Computerworld*, November 2:86-90, 1992.
- [Men02] Meng, J., Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules. *Doctoral Thesis*, University of Florida, Gainesville, FL, 2002.
- [Mil80] Milner, R., A Calculus of Communicating Systems. 1980. LNCS 92.
- [Mit98] Mitrović-Minić, S., Pickup and Delivery Problem with Time Windows: A Survey. *Technical Report SFU CMPT TR 1998-12*, Simon Fraser University, Canada, 1998.
- [Moai04] Moai LiveExchange (2004). On-line at <<http://www.moai.com/>>.
- [Mos82] Moss, J., Nested Transactions and Reliable Distributed Computing. In *Proceedings of the Second Symposium on Reliability in Distributed Software and Database Systems*, 33-39, Pittsburgh, PA, July 1982.
- [MB00] McClintock, C. and Berlioz, C.A., Implementing Business Rules in Java. *Java Developers Journal*, 5(5):8-16, 2000.
- [MGM99] Maes, P., Guttman, R.H., and Moukas, A.G., Agents that Buy and Sell: Transforming Commerce as we Know it. *Communications of the ACM*, 42(3):81-91, 1999.
- [MO99] Marjanovic, O. and Orłowska, M.E., On Modeling and Verification of Temporal Constraints in Production Workflows, *Knowledge and Information Systems*, 1(2):157-192, 1999.
- [MR99] Müller, R., and Rahm, E., Rule-Based Dynamic Modification of Workflows in a Medical Domain. In *Buchmann, A.P. (Ed.): Proceedings of the Datenbanksysteme in Büro, Technik und Wissenschaft (BTW'99)*, 429-448, Freiburg im Breisgau, Germany, March 1999. On-line at <<http://dol.uni-leipzig.de/pub/showDoc.Fulltext?lang=de&doc=1999-14&format=pdf&compression=>>>.

- [MS02] Mangan, P. and Sadiq, S., A Constraint Specification Approach to Building Flexible Workflows. *Journal of Research and Practice in Information Technology*, 35(1):21-39, 2002.
- [MWW98] Muth, P., Weissenfels, J., and Weikum, G., What Workflow Technology Can Do For Electronic Commerce. In *Proceedings of the EURO-MED NET Conference*, Nicosia, Cyprus, March 1998. On-line at <<http://citeseer.ist.psu.edu/muth98what.html>>.
- [MWW+98] Muth, P., Wodtke, D., Weissenfels, J., Weikum, G., and Kotz Dittrich, A., Enterprise-wide Workflow Management based on State and Activity Charts. In *Dogac, A., Kalinichenko, L., Tamer Ozsu, M., and Sheth, A. (Eds.): Advances in Workflow Management Systems and Interoperability*, 281-303, NATO Advanced Study Institute, Springer-Verlag, 1998.
- [NBB+03] Neumann, D., Benyoucef, M., Bassil, S., and Vachon, J., Applying the Montreal Taxonomy to State of the Art E-Negotiation Systems. *Group Decision and Negotiation Journal*, 12(4):287-310, July 2003. (Published in cooperation with the Institute for Operations Research and the Management Sciences - Informs - and its Section on Group Decision and Negotiation.). Kluwer Academic Publishers.
- [NDS96] Ngu, A.H.H., Duong, T., and Srinivasan, U., Modeling Workflow using Tasks and Transactions. In *Proceedings of the Seventh International Workshop on Database and Expert Systems Applications (DEXA '96)*, Zurich, Switzerland, September 1996. On-line at <<http://www.cse.unsw.edu.au/~anne/work3/work3.html>>.
- [OPL04] ILOG OPL Studio (2004). On-line at <<http://www.ilog.com/products/oplstudio/>>.
- [Par81] Park, D., Concurrency and Automata on Infinite Sequences. In *Proceedings of the Fifth GI-Conference on Theoretical Computer Science*, 167-183, 1981. LNCS 104.
- [Petri04] Petri Nets World: Online Services for the International Petri Nets Community (2004). On-line at <<http://www.daimi.au.dk/PetriNets>>.
- [PEL97] Pozewaunig, H., Eder, J., and Liebhart, W., ePERT: Extending PERT for Workflow Management Systems. In *Proceedings of the First East-European Symposium on Advances in Database and Information Systems (ADBIS'97)*, 217-224, St.-Petersburg, Russia, September 1997.
- [PSR92] Potvin, J.-Y., Shen, Y., Rousseau, J.-M., Neural Network for Automated Vehicle Dispatching. *Computers and Operations Research* (Special issue on neural networks and operations research), 19(3-4):267-276. April/May 1992.
- [RD98] Reichert, M., and Dadam, P., ADEPTflex: Supporting Dynamic Changes of Workflow without Losing Control. *Journal of Intelligent Information Systems*, 10(2):93-129, 1998.

- [RRD03a] Reichert, M., Rinderle, S., and Dadam, P., ADEPT Workflow Management System: Flexible Support for Enterprise-wide Business Processes (Tool Presentation). In *Proceedings of the First International Conference on Business Process Management (BPM'03)*, 370-379, Eindhoven, The Netherlands, June 2003. LNCS 2678.
- [RRD03b] Rinderle, S., Reichert, M., and Dadam, P., Evaluation of Correctness Criteria for Dynamic Workflow Changes. In *Proceedings of the First International Conference on Business Process Management (BPM'03)*, 41-57, Eindhoven, The Netherlands, June 2003. LNCS 2678.
- [RRD04a] Rinderle, S., Reichert, M., and Dadam, P., Correctness Criteria for Dynamic Changes in Workflow Systems – A Survey. In *Data & Knowledge Engineering* (Special issue on advances in business process management), 50(1):9-34, 2004.
- [RRD04b] Rinderle, S., Reichert, M., and Dadam, P., Flexible Support of Team Processes by Adaptive Workflow Systems. In *Distributed and Parallel Databases*, 16(1):91-116, 2004.
- [RRD04c] Rinderle, S., Reichert, M., and Dadam, P., On Dealing with Structural Conflicts between Process Type and Instance Changes. In *Proceedings of the Second International Conference on Business Process Management (BPM'04)*, 274-289, Potsdam, Germany, June 2004. LNCS 3080.
- [RT02] Reichert, M. and Tarabrin, A., ADEPT Release version 2.0 – Short User Guide, Department DBIS, University of Ulm, Germany, October 2002.
- [RXZ04] Ray, I., Xin, T., and Zhu, Y., Ensuring Task Dependencies During Workflow Recovery. In *Proceedings of the Fifteenth International Workshop on Database and Expert Systems Applications (DEXA'04)*, 24-33, Zaragoza, Spain, September 2004. LNCS 3180.
- [Sad99] Sadiq, S., Workflows in Dynamic Environments - Can they be Managed? In *Proceedings of the Second International Symposium on Cooperative Database Systems for Advanced Applications (CODAS'99)*, 165-176, Woollongong, Australia, March 1999.
- [Sie96] Siebert, R., Adaptive Workflow for the German Public Administration. In *Proceedings of the Workshop on Adaptive Workflow at the First International Conference on Practical Aspects of Knowledge Management (PAKM'96)*. Basel, Switzerland, 1996. On-line at <<http://www.informatik.uni-stuttgart.de/ipvr/as/publikationen/Sieber96b.html>>.
- [Str99] Strobel, M., Effects of Electronic Markets on Negotiation Processes – Evaluating Protocol Suitability. *Technical Report 93237*, IBM, Zurich Research Laboratory, Switzerland, 1999.
- [Sur01] A Survey of Auctions (2001). On-line at <<http://www.agorics.com/Library/auctions.html>>.

- [SAA99] Sheth, A.P., van der Aalst, W., and Arpinar, I.B., Processes Driving the Networked Economy. In *IEEE Concurrency*, 7(3):18-31, July-September 1999.
- [SL95] Sandholm T. and Lesser, V., Issues in Automated Negotiation and Electronic Commerce: Extending the Contract Net Framework. In *Proceedings of the First International Conference on Multi-Agent Systems (ICMAS'95)*, 328-335, San Francisco, CA, June 1995.
- [SMO00] Sadiq, S., Marjanovic, O., and Orłowska, M., Managing Change and Time in Dynamic Workflow Processes. *The International Journal of Cooperative Information Systems*, 9(1-2):93-116, 2000.
- [SO99a] Sadiq, S. and Orłowska, M.E., Architectural Considerations in Systems Supporting Dynamic Workflow Modifications. In *Proceedings of the Workshop on Software Architectures for Business Process Management at the 11th Conference on Advanced Information Systems Engineering (CaiSE'99)*, Heidelberg, Germany. June 1999. On-line at <http://www.dstc.edu.au/praxis/publications/ssadiq_sabpm_1999.pdf>.
- [SO99b] Sadiq, W. and Orłowska, M.E., On capturing Process Requirements of Workflow Based Business Information Systems. In *Proceedings of the Third International Conference on Business Information Systems (BIS'99)*, 195-209, Poznan, Poland, April 1999. (Springer-Verlag)
- [SRK+01] Stricker, C., Riboni, S., Kradolfer, M., and Taylor, J., Market-based Workflow Management for Supply Chains of Services. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, January 2001. On-line at <<http://anaisoft.unige.ch/public-documents/deliverables/hicss33.pdf>>.
- [SS95] Savelsbergh, M.W.P. and Sol, M., The General Pickup and Delivery Problem. *Transportation Science*, 29:17-29. 1995. On-line at <<http://www.isye.gatech.edu/~mwps/publications/ts29.pdf>>.
- [SSO01] Sadiq, S., Sadiq, W., and Orłowska, M., Pockets of Flexibility in Workflow Specifications. In *Proceedings of the 20th International Conference on Conceptual Modeling (ER'01)*, 513-526, Yokohama, Japan, November 2001.
- [Tib04] TIBCO Software Inc. (2004). On-line at <<http://www.tibco.com>>.
- [Tra04] Transports Québec: Carte routière, les distances routières entre les principales agglomérations (2004). On-line at <http://www.mtq.gouv.qc.ca/images/information/carte_routiere/PDF/carton_les_distances.pdf>.
- [Tri98] Trilling G., Génération automatique d'horaires de médecins de garde pour l'hôpital Côte-des-Neiges de Montréal. *Technical Report CRT-98-05*, Centre de Recherche sur les Transports, Université de Montréal, Canada, January 1998.
- [Tsa93] Tsang, E., Foundations of Constraint Satisfaction. *Academic Press, London and San Diego*, 421 pp., 1993. ISBN 0-12-701610-4.

- [TCD93] Taleb-Ibrahimi, M., de Castilho, B., and Daganzo, C.F., Storage Space Versus Handling Work in Container Terminals. *Transportation Research Part B: Methodological*, 27(1):13-32, 1993.
- [UML04] Unified Modeling Language Resource Center (2004). On-line at <<http://www-306.ibm.com/software/rational/uml/>>.
- [VA04] Verbeek, H.M.W. and van der Aalst, W.M.P., Woflan Home Page (2004). On line at <<http://tmitwww.tm.tue.nl/research/woflan/>>.
- [VHA02] Verbeek, H.M.W., Hirnschall, A., and van der Aalst, W.M.P., XRL/Flower: Supporting Inter-Organizational Workflows Using XML/Petri-net Technology. In *Proceedings of the Workshop on Web Services, e-Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Semantic (WES'02) – Held in conjunction with the 14th Int'l Conf. on Advances Information Systems Engineering (CAiSE'02)*, 93-108, Toronto, Canada, May 2002. LNCS 2512.
- [Wat01] Watts, A., Comparison of Staffware and MQ Series Workflow. *White Paper*, Kraftware Systems Limited, July 2001. On line at <<http://www.kraftwaresystems.co.uk/pdf/staffware-mqseriesworkflowcomparison.pdf>>.
- [Web04] BEA Systems – BEA WebLogic Integration (2004). On-line at <<http://www.beasys.com/products/weblogic/integration/index.shtml>>.
- [Wes01] Weske, M., Formal Foundation and Conceptual Design of Dynamic Adaptations in a Workflow Management System. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, Maui, Hawaii, January 2001. On-line at <<http://csdl.computer.org/comp/proceedings/hicss/2001/0981/07/09817051.pdf>>.
- [WfMC95] Workflow Management Coalition, The Workflow Reference Model. *WFMC-TC-1003*, Version 1.1, January 1995. On-line at <<http://www.wfmc.org/standards/docs/tc003v11.pdf>>.
- [WfMC97] Workflow Management Coalition, Workflow Client Application (Interface 2) Application Programming Interface (WAPI) Naming Conventions. *WFMC-TC-1013*, Version 1.4, November 1997. On-line at <<http://www.wfmc.org/standards/docs/tc013v14a.pdf>>.
- [WfMC98] Workflow Management Coalition, Workflow Management Application Programming Interface (Interface 2&3) Specification. *WFMC-TC-1009*, Version 2.0, July 1998. On-line at <<http://www.wfmc.org/standards/docs/if2v20.pdf>>.
- [WfMC99a] Workflow Management Coalition, Interface 1: Process Definition Interchange Process Model. *WFMC-TC-1016-P*, Version 1.1, October 1999. On-line at <http://www.wfmc.org/standards/docs/TC-1016-P_v11_IF1_Process_definition_Interchange.pdf>.
- [WfMC99b] Workflow Management Coalition, Terminology and Glossary. *WFMC-TC-1011*, Version 3.0, February 1999. On-line at <http://www.wfmc.org/standards/docs/TC-1011_term_glossary_v3.pdf>.

- [WfMC01] Workflow Management Coalition, Interoperability Workflow-XML Binding, *WFMC-TC-1023*, Version 1.1, November 2001. On-line at <<http://www.wfmc.org/standards/docs/Wf-XML-11.pdf>>.
- [WfMC04] The Workflow Management Coalition (2004). On-line at <<http://www.wfmc.org>>.
- [WARIA04] Workflow and Reengineering International Association (2004). On-line at <<http://www.waria.com>>.
- [WHF+95] Weil, G., Heus, K., François, P., and Poujade, M., Constraint Programming for Nurse Scheduling. *Engineering in Medicine and Biology*, 14(4):417-422, 1995.
- [WI98] Weber, M., Illmann, T., Using Java for the Coordination of Workflows in the WWW. In *Tagungsband der Fachtagung "Interaktion im Web – Innovative Kommunikationsformen"*, Marburg, Germany, May 1998. On-line at <<http://medien.informatik.uni-ulm.de/forschung/publikationen/interaktion98.pdf>>.
- [WMW98] Weissenfels, J, Muth, P., and Weikum, G, Flexible Worklist Management in a Light-Weight Workflow Management System. In *Proceedings of the Workshop on Workflow Management Systems at the Sixth International Conference on Extending Database Technology (EDBT'98)*, 29-38, Valencia, Spain, March 1998.
- [WS97] Worah, D. and Sheth, A., Transactions in Transactional Workflows. In *Jajodia S. and Kerschberg, L (Eds.), Advanced Transaction Models and Architectures*, Chapter 1, 3-34, Kluwer Academic Publishers, 1997.
- [WSC102] Web Services Choreography Interface 1.0 (2002). On-line at <<http://www.w3.org/TR/wsci/>>
- [WSCL02] Web Services Conversation Language 1.0 (2002). On-line at <<http://www.w3.org/TR/wscl10/>>.
- [WSDL01] Web Services Description Language 1.1 (2001). On-line at <<http://www.w3.org/TR/wsdl>>.
- [WSW+70] Wilson, N.H.M., Sussman, J.M., Wong, H.-K., and Higonnet, T., Scheduling Algorithms for a Dial-A-Ride System. *Technical Report TR-70-13*, Department of Civil Engineering, MIT, Cambridge, MA, 1970.
- [WWW98] Wurman, P.R., Wellman, M.P., and Walsh, W.E., The Michigan Internet AuctionBot: A Configurable Auction Server for Human and Software Agents, In *Proceedings of the Second International Conference on Autonomous Agents*, 301-308, Minneapolis, MN, May 1998.
- [W3C04] The World Wide Web Consortium (2004). On-line at <<http://www.w3.org>>.
- [XML04] Extensible Markup Language (2004). On-line at <<http://www.w3.org/XML>>.

Appendix A Extending the Workflow Reference Model: Workflow Management Application Programming Interface Specification

The Workflow Management Coalition [WfMC04] has developed a standard general model for Workflow Management Systems (WfMSs). This model, called the Workflow Reference Model (WfRM) [WfMC95], does not support many of the concepts and the functionality required by workflow-based complex socio-technical systems. This appendix, based on [BRK+03], presents an extension of the WfRM in order to accommodate these requirements. A compressed summary of the new or extended groups of operations is first given in Section A.1. Then, a detailed summary of these groups, showing the signature of each operation is given in Section A.2. In Section A.3, the detailed specification of the extended Workflow Management Application Programming Interface (WAPI, Interfaces 1, 2 and 3) is presented. Sections A.4 and A.5 provides the WAPI data types addendum and the WAPI error return codes addendum.

A.1 Compressed Summary of the Groups of Operations and Operations

Activity Control Functions (Interface 2&3)

WMOpenActivityTemplatesList: Specifies and opens query to produce a list of all activity templates that meet the selection criterion of the filter.

WMFetchActivityTemplate: Returns the next activity template from the set of activity templates that met the selection criterion stated in the WMOpenActivityTemplatesList call.

WMCloseActivityTemplatesList: Closes the query of activity templates.

WMGetActivityTemplate: Returns the activity template specified by its ID.

WMCreateActivityInstance: Creates an activity instance from an activity template.

WMInsertActivityInstance: Inserts an activity instance between two groups of existing activity instances.

WMDeleteActivityInstance: Deletes an activity instance.

WMMoveActivityInstance: Moves an existing activity instance from its original place between two groups of activity instances.

WMAssignActivityInstanceAttribute: Sets or changes attribute values.

WMInsertActivityInstanceAttribute: Inserts a new attribute into an activity instance.

WMDeleteActivityInstanceAttribute: Deletes an attribute assigned to an activity instance.

WMAssignActivityInstanceParticipants: Assigns one or up to ten workflow participants to an activity instance.

WMAssignActivityInstanceDuration: Assigns a duration to an activity instance.

WMAssignActivityInstanceTime: Assigns a (starting, finishing) time to an activity instance.

WMAssignActivityInstanceWUT: Assigns a warm-up duration to an activity instance.

Process Control Functions (Interface 2&3)

WMKeepProcessInstance: Stores the process definition corresponding to a modified process instance.

WMInsertProcessInstance: The process instance provided is inserted into another process instance as a sub-workflow.

Work-list/Work-item Handling Functions (Interface 2&3)

WMReassignWorkItem: Reassigns a work-item from one workflow participant's work-list to another workflow participant's work-list.

WMAssignWorkItemAttribute: Sets or updates the value of an attribute of a work-item.

WMDeleteWorkItem: Deletes a work-item in a given work-list.

Classification Category Definition Functions (New - Interface 1)

WMCreateClassificationCategory: Creates a new classification category.

WMDeleteClassificationCategory: Deletes a classification category.

Activity Template Modeling Functions (New - Interface 1)

WMCreateActivityTemplate: Creates an “empty” new activity template.

WMOpenActivityTemplate: Prepares for editing of an activity template.

WMCloseActivityTemplate: Allows the system to free up any resources that are maintained to handle the activity template.

WMAssignActivityTemplateClassificationCategory: Assigns an activity template to a classification category.

WMDetractActivityTemplateClassificationCategory: Detracts an activity template from a classification category.

WMDeleteActivityTemplate: Deletes an activity template.

Activity Template Attribute Manipulation Functions (New - Interface 1)

WMAssignActivityTemplateAttribute: Sets an attribute of an activity template.

WMInsertActivityTemplateAttribute: Inserts a new attribute into an activity template.

WMDeleteActivityTemplateAttribute: Deletes an attribute assigned to an activity template.

WMAssignActivityTemplateParticipants: Assigns one or up to ten workflow participants to an activity template.

WMAssignActivityTemplateDuration: Assigns a duration to an activity template.

WMAssignActivityTemplateTime: Assigns a (starting, finishing) time to an activity template.

WMAssignActivityTemplateWUT: Assigns a warm-up duration to an activity template.

Process Modeling Functions (Interface 1)

WMAssignProcDefClassificationCategory: Assigns a process definition to a classification category.

WMDeductProcDefClassificationCategory: Deducts a process definition from a classification category.

A.2 Detailed Summary of the Groups of Operations and Operations

Activity Control Functions (Interface 2&3)

WMErrRetType WMOpenActivityTemplatesList (
 // Specifies and opens query to produce a list of all activity templates that meet the selection criterion of the filter.

```

    in      WMTPSessionHandle psession_handle,
    in      WMTFilter pactivity_template_filter,
    in      WMTBoolean count_flag,
    out     WMTQueryHandle pquery_handle,
    out     WMTInt32 pcount
  )
  
```

WMErrRetType WMFetchActivityTemplate (
 // Returns the next activity template from the set of activity templates that met the selection criterion stated in the WMOpenActivityTemplatesList call.

```

    in      WMTPSessionHandle psession_handle,
    in      WMTQueryHandle pquery_handle,
    out     WMTActivityTemplate pactivity_template_buf_ptr
  )
  
```

WMErrRetType WMCloseActivityTemplatesList (
 // Closes the query of activity templates.

```

    in      WMTPSessionHandle psession_handle,
    in      WMTQueryHandle pquery_handle,
  )
  
```

WMErrRetType WMGetActivityTemplate (
 // Returns the activity template specified by its ID.

```

    in      WMTPSessionHandle psession_handle,
    in      WMTText activity_template_name,
    out     WMTActivityTemplateID pactivity_template_ID,
  )
  
```


WMTerrRetType **WMCreateActivityInstance** (

// Creates an activity instance from an activity template.

in WMTPSessionHandle **psession_handle**,
in WMTPActivityTemplateID **pactivity_template_id**,
out WMTPActivityInstanceID **pactivity_instance_id**
)

WMTerrRetType **WMInsertActivityInstance** (

// Inserts an activity instance between two groups of existing activity instances.

in WMTPSessionHandle **psession_handle**,
in WMTPProcInstID **pproc_inst_id**,
in WMTPActivityInstID **pactivity_instance_id**,
in WMTPActivityInstID[] **pbefore_activity_inst_id**,
in WMTPActivityInstID[] **pafter_activity_inst_id**
)

WMTerrRetType **WMDeleteActivityInstance** (

// Deletes an activity instance.

in WMTPSessionHandle **psession_handle**,
in WMTPProcInstID **pproc_inst_id**,
in WMTPActivityInstID **pactivity_inst_id**
)

WMTerrRetType **WMMoveActivityInstance** (

// Moves an existing activity instance from its original place between two groups of activity instances.

in WMTPSessionHandle **psession_handle**,
in WMTPProcInstID **pproc_inst_id**,
in WMTPActivityInstID **pactivity_inst_id**,
in WMTPActivityInstID[] **pbefore_activity_inst_id**,
in WMTPActivityInstID[] **pafter_activity_inst_id**
)

WMTerrRetType **WMAssignActivityInstanceAttribute** (

// Sets or changes attribute values.

in WMTPSessionHandle **psession_handle**,
in WMTPProcInstID **pproc_inst_id**,
in WMTPActivityInstID **pactivity_inst_id**,
in WMTPAttrName **pattribute_name**,
in WMTInt32 **attribute_type**,
in WMTInt32 **attribute_length**,
in WMTPText **pattribute_value**
)

```

WMTerrRetType WMInsertActivityInstanceAttribute (
// Inserts a new attribute into an activity instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length
)

```

```

WMTerrRetType WMDeleteActivityInstanceAttribute (
// Deletes an attribute assigned to an activity instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPAttrName pattribute_name
)

```

```

WMTerrRetType WMAssignActivityInstanceParticipants (
// Assigns one or up to ten workflow participants to an activity instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPWflParticipant[] pparticipants[10],
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id
)

```

```

WMTerrRetType WMAssignActivityInstanceDuration(
// Assigns a duration to an activity instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTInt32 duration_limit_type,
    in    WMTPInt16 pduration_value
)

```

```

WMTerrRetType WMAssignActivityInstanceTime(
// Assigns a (starting, finishing) time to an activity instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTInt32 time_period_type,
    in    WMTDate ptime_value
)

```

```

WMTErrRetType WMAssignActivityInstanceWUT(
// Assigns a warm-up duration to an activity instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTInt32 wut_limit_type,
    in    WMTPInt16 pwut_value
)

```

Process Control Functions (Interface 2&3)

```

WMTErrRetType WMKeepProcessInstance (
// Stores the process definition corresponding to a modified process instance.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    out   WMTPProcDefID pproc_def_id
)

```

```

WMTErrRetType WMInsertProcessInstance (
// The process instance provided is inserted into another process instance as a subwork-
// flow.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID phost_proc_inst_id,
    in    WMTPProcInstID pinsert_proc_inst_id,
    in    WMTPActivityInstID[] pbefore_activity_inst_id,
    in    WMTPActivityInstID[] pafter_activity_inst_id
)

```

Work-list/Work-item Handling Functions (Interface 2&3)

```

WMTErrRetType WMReassignWorkItem (
// Reassigns a work-item from one workflow participant's work-list to another workflow
// participant's work-list.
    in    WMTPSessionHandle psession_handle,
    in    WMTPWflParticipant psource_user,
    in    WMTPWflParticipant ptarget_user,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPWorkItemID pwork_item_id
)

```

```

WMTErrRetType WMAssignWorkItemAttribute (
// Sets or updates the value of an attribute of a work-item.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPWorkItemID pwork_item_id
    in    WMTPAttrName pattribute_name,

```

```

    in      WMTInt32 attribute_type,
    in      WMTInt32 attribute_length,
    in      WMTPText pattribute_value
  )

```

```

WMErrRetType WMDeleteWorkItem (
// Deletes a work-item in a given work-list.
    in      WMTSessionHandle psession_handle,
    in      WMTProcInstID pproc_inst_id,
    in      WMTWorkItemID pwork_item_id
)

```

Classification Category Definition Functions (New - Interface 1)

```

WMErrRetType WMCreateClassificationCategory (
// Creates a new classification category.
    in      WMTSessionHandle psession_handle,
    in      WMTText classification_category_name,
    out     WMTClassificationCategoryID pclassification_category_id
)

```

```

WMErrRetType WMDeleteClassificationCategory (
// Deletes a classification category.
    in      WMTSessionHandle psession_handle,
    in      WMTClassificationCategoryID pclassification_category_id
)

```

Activity Template Modeling Functions (New - Interface 1)

```

WMErrRetType WMCreateActivityTemplate (
// Creates an "empty" new activity template.
    in      WMTSessionHandle psession_handle,
    in      WMTName activity_template_name,
    out     WMTActivityTemplateID pactivity_template_id
)

```

```

WMErrRetType WMOpenActivityTemplate (
// Prepares for editing of an activity template.
    in      WMTSessionHandle psession_handle,
    in      WMTActivityTemplate pactivity_template,
    out     WMTPEntity pactivity_template_handle
)

```

```

WMErrRetType WMCloseActivityTemplate (
// Allows the system to free up any resources that are maintained to handle the activity
template.

```

```

    in    WMTPSessionHandle psession_handle,
    in    WMTPEntity pactivity_template_handle
)

```

```

WMTErrRetType WMAssignActivityTemplateClassificationCategory (
// Assigns an activity template to a classification category.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPClassificationCategoryID pclassification_category_id
)

```

```

WMTErrRetType WMDeductActivityTemplateClassificationCategory (
// Deducts an activity template from a classification category.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPClassificationCategoryID pclassification_category_id
)

```

```

WMTErrRetType WMDeleteActivityTemplate (
// Deletes an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id
)

```

Activity Template Attribute Manipulation Functions (New - Interface 1)

```

WMTErrRetType WMAssignActivityTemplateAttribute (
// Sets an attribute of an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplate pactivity_template,
    in    WMTPAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length,
    in    WMTPText pattribute_value
)

```

```

WMTErrRetType WMInsertActivityTemplateAttribute (
// Inserts a new attribute into an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length
)

```

```

WMTErrRetType WMDeleteActivityTemplateAttribute (
// Deletes an attribute assigned to an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPAttrName pattribute_name
)

```

```

WMTErrRetType WMAssignActivityTemplateParticipants (
// Assigns one or up to ten workflow participants to an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPWflParticipant[] pparticipants[10],
    in    WMTPActivityTemplateID pactivity_template_id
)

```

```

WMTErrRetType WMAssignActivityTemplateDuration (
// Assigns a duration to an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPInt32 duration_limit_type,
    in    WMTPInt16 pduration_value
)

```

```

WMTErrRetType WMAssignActivityTemplateTime (
// Assigns a (starting, finishing) time to an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPInt32 time_period_type,
    in    WMTPDate ptime_value
)

```

```

WMTErrRetType WMAssignActivityTemplateWUT (
// Assigns a warm-up duration to an activity template.
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPInt32 wut_limit_type,
    in    WMTPInt16 pwut_value
)

```

Process Modeling Functions (Interface 1)

```

WMTErrRetType WMAssignProcDefClassificationCategory (
// Assigns a process definition to a classification category.
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcDefID pproc_def_id,
    in    WMTPClassificationCategoryID pclassification_category_id
)

```

```
WMTErrRetType WMDetractProcDefClassificationCategory (  
// Detracts a process definition from a classification category.  
    in    WMTPSessionHandle psession_handle,  
    in    WMTProcDefID pproc_def_id,  
    in    WMTClassificationCategoryID pclassification_category_id  
)
```

A.3 Description of the Extended WAPI Specification

A.3.1 Inserting Activities

WMOpenActivityTemplatesList

(belongs to WAPI Activity Control Functions)

NAME

WMOpenActivityTemplatesList – Specifies and opens query to produce a list of all activity templates that meet the selection criterion of the filter.

DESCRIPTION

```

WMErrRetType WMOpenActivityTemplatesList (
    in      WMTPSessionHandle psession_handle,
    in      WMTFilter pactivity_template_filter,
    in      WMTBoolean count_flag,
    out     WMTQueryHandle pquery_handle,
    out     WMTInt32 pcount
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_filter	Filter associated with the activity templates.
count_flag	Boolean flag that indicates if the total count of activity templates should be returned.
pquery_handle	Pointer to a structure containing a unique query information.
pcount	Total number of activity templates that fulfil the filter condition.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_FILTER

```


WMFetchActivityTemplate (belongs to WAPI Activity Control Functions)

NAME

WMFetchActivityTemplate – Returns the next activity template from the set of activity templates that met the selection criterion stated in the **WMOpenActivityTemplatesList** call.

DESCRIPTION

```
WMErrRetType WMFetchActivityTemplate (
    in    WMTPSessionHandle psession_handle,
    in    WMTPQueryHandle pquery_handle,
    out   WMTPActivityTemplate pactivity_template_buf_ptr
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pquery_handle	Identification of the specific query handle returned by the WMOpenActivityTemplatesList query command.
pactivity_template_buf	Pointer to a buffer area provided by the client application where the activity template structure will be placed.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE
WM_NO_MORE_DATA
```

WMCloseActivityTemplatesList
(belongs to WAPI Activity Control Functions)

NAME

WMCloseActivityTemplatesList – Closes the query of activity templates.

DESCRIPTION

```
WMErrRetType WMCloseActivityTemplatesList (
    in WMTPSessionHandle psession_handle,
    in WMTPQueryHandle pquery_handle,
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pquery_handle	Identification of the specific query handle returned by the WMOpenActivityTemplatesList query command.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_QUERY_HANDLE
```

WMGetActivityTemplate
(belongs to WAPI Activity Control Functions)

WMGetActivityTemplate – Returns the activity template specified by its ID.

DESCRIPTION

```
WMErrRetType WMGetActivityTemplate (
    in      WMTPSessionHandle psession_handle,
    in      WMTText activity_template_name,
    out     WMTPActivityTemplateID pactivity_template_ID,
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_name	The name of the activity instance requested.
pactivity_template_id	Pointer to the ID of the activity template requested.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE_NAME

WMCreateActivityInstance
(belongs to WAPI Activity Control Functions)

NAME

WMCreateActivityInstance – Creates an activity instance from an activity template.

DESCRIPTION

```
WMTErrRetType WMCreateActivityInstance (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    out   WMTPActivityInstanceID pactivity_instance_id
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to the ID of the activity template to be instantiated.
pactivity_instance_id	Pointer to the ID of the activity instance that is created.

ERROR RETURN VALUE

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE

WMInsertActivityInstance
(belongs to WAPI Activity Control Functions)

NAME

WMInsertActivityInstance – Inserts an activity instance between two groups of existing activity instances.

DESCRIPTION

```

WMTErrRetType WMInsertActivityInstance (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPActivityInstID[] pbefore_activity_inst_id,
    in    WMTPActivityInstID[] pafter_activity_inst_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the activity template ID that is to be inserted.
pbefore_activity_inst_id[]	Pointer array to the activity instance IDs that are determined to be before the newly inserted activity instance.
pafter_activity_inst_id[]	Pointer array to the activity instance IDs that are determined to be after the newly inserted activity instance.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_BEFORE_ACTIVITY_INSTANCE
WM_INVALID_AFTER_ACTIVITY_INSTANCE

```

WMCreateActivityTemplate (Interface 1)

(belongs to Activity Template Modelling Functions in Interface 1)

NAME**WMCreateActivityTemplate** – Creates an “empty” new activity template.**DESCRIPTION**

```

WMErrRetType WMCreateActivityTemplate (
    in      WMTPSessionHandle psession_handle,
    in      WMTName activity_template_name,
    out     WMTPActivityTemplateID pactivity_template_id
)

```

Argument	Description
psession_handle	Pointer to the structure with the session information created by a call to WMConnect.
activity_template_name	The name for the template that is being created.
pactivity_template_id	Pointer to the new activity template ID for the activity template created.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE

WMOpenActivityTemplate (Interface 1)
 (belongs to Activity Template Modelling Functions in Interface 1)

NAME

WMOpenActivityTemplate – Prepares for editing of an activity template.

DESCRIPTION

This command tells the Enactment Service to prepare for editing of the specified activity template.

```
WMErrRetType WMOpenActivityTemplate (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplate pactivity_template,
    out   WMTPEntity pactivity_template_handle
)
```

Argument	Description
psession_handle	Pointer to the structure with the session information created by a call to WMConnect.
pactivity_template	Pointer to a structure containing the activity template to be edited.
pactivity_template_handle	Pointer to a buffer which will receive the entity representing the activity template.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE

WMCloseActivityTemplate (Interface 1)
 (belongs to Activity Template Modelling Functions in Interface 1)

NAME

WMCloseActivityTemplate – Allows the system to free up any resources that are maintained to handle the activity template.

DESCRIPTION

```
WMErrRetType WMCloseActivityTemplate (
    in    WMTPSessionHandle psession_handle,
    in    WMTPEntity pactivity_template_handle
)
```

Argument	Description
psession_handle	Pointer to the structure with the session information created by a call to WMConnect.
pactivity_template_handle	Pointer to a buffer which receives the contents of the activity template. It is assumed that the entity representing the activity template becomes inaccessible once the activity template is closed.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE

WMCreateClassificationCategory (Interface 1)

(belongs to Classification Category Definition Functions in Interface 1)

NAME**WMCreateClassificationCategory** – Creates a new classification category.**DESCRIPTION**

```

WMErrRetType WMCreateClassificationCategory (
    in      WMTPSessionHandle psession_handle,
    in      WMTText classification_category_name,
    out     WMTPClassificationCategoryID pclassification_category_id
)

```

Argument	Description
psession_handle	Pointer to the structure with the session information created by a call to WMConnect.
classification_category_name	The name for the classification category that is being created.
pclassification_category_id	Pointer to the new classification category ID for the classification category created.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE

WMDeleteClassificationCategory (Interface 1)

(belongs to Classification Category Definition Functions in Interface 1)

NAME**WMDeleteClassificationCategory** – Deletes a classification category.**DESCRIPTION**

```

WMErrRetType WMDeleteClassificationCategory (
    in      WMTPSessionHandle psession_handle,
    in      WMTPClassificationCategoryID pclassification_category_id
)

```

Argument	Description
psession_handle	Pointer to the structure with the session information created by a call to WMConnect.
pclassification_category_id	Pointer to the classification category ID for the classification category to be deleted.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_CLASSIFICATION_CATEGORY

WMAssignActivityTemplateClassificationCategory (Interface 1)
(belongs to WAPI Activity Template Modelling Functions)

NAME

WMAssignActivityTemplateClassificationCategory – Assigns an activity template to a classification category.

DESCRIPTON

Note that this function can be executed repeatedly to assign an activity template to more than one classification category.

```

WMTErrRetType WMAssignActivityTemplateClassificationCategory (
    in WMTPSessionHandle psession_handle,
    in WMTPActivityTemplateID pactivity_template_id,
    in WMTPClassificationCategoryID pclassification_category_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to the ID of the activity template that is to be assigned to a classification category.
pclassification_category__id	Pointer to the ID of the classification category to which the activity template is to be assigned.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE
 WM_INVALID_CLASSIFICATION_CATEGORY

WMDetractActivityTemplateClassificationCategory (Interface 1)
(belongs to WAPI Activity Template Modelling Functions)

NAME

WMDetractActivityTemplateClassificationCategory – Detracts an activity template from a classification category.

DESCRIPTION

```

WMTErrRetType WMDetractActivityTemplateClassificationCategory (
    in WMTPSessionHandle psession_handle,
    in WMTPActivityTemplateID pactivity_template_id,
    in WMTPClassificationCategoryID pclassification_category_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to the ID of the activity template that is to be detracted from a classification category.
pclassification_category_id	Pointer to the ID of the classification category from which the activity template is to be detracted.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE
 WM_INVALID_CLASSIFICATION_CATEGORY

WMAssignProcDefClassificationCategory (Interface 1)
(belongs to WAPI Process Modelling Functions)

NAME

WMAssignProcDefClassificationCategory – Assigns a process definition to a classification category.

DESCRIPTON

Note that this function can be executed repeatedly to assign a process definition to more than one classification category.

```
WMTErrRetType WMAssignProcDefClassificationCategory (
    in WMTPSessionHandle psession_handle,
    in WMTPProcDefID pproc_def_id,
    in WMTPClassificationCategoryID pclassification_category_id
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_def_id	Pointer to the ID of the process definition that is to be assigned to a classification category.
pclassification_category_id	Pointer to the ID of the classification category to which the process definition is to be assigned.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_PROCESS_DEFINITION
 WM_INVALID_CLASSIFICATION_CATEGORY

WMDetractProcDefClassificationCategory (Interface 1)
 (belongs to WAPI Process Modelling Functions)

NAME

WMDetractProcDefClassificationCategory – Detracts a process definition from a classification category.

DESCRIPTION

```

WMTErrRetType WMDetractProcDefClassificationCategory (
    in WMTPSessionHandle psession_handle,
    in WMTPProcDefID pproc_def_id,
    in WMTPClassificationCategoryID pclassification_category_id
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_def_id	Pointer to the ID of the process definition that is to be detracted from a classification category.
pclassification_category_id	Pointer to the ID of the classification category from which the process definition is to be detracted.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_PROCESS_DEFINITION
 WM_INVALID_CLASSIFICATION_CATEGORY

A.3.2 Deleting Activities and Templates

WMDeleteActivityInstance

(belongs to WAPI Activity Control Functions)

NAME

WMDeleteActivityInstance - Deletes an activity instance.

DESCRIPTION

```
WMTErrRetType WMDeleteActivityInstance (
    in WMTPSessionHandle psession_handle,
    in WMTPProcInstID pproc_inst_id,
    in WMTPActivityInstID pactivity_inst_id)
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance that is to be deleted.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
```

Note: More ERROR RETURN VALUE to be added. *E.g.*, Detect the deletion of an activity instance providing an attribute that is required by another activity instance in the process instance.

WMDeleteActivityTemplate (Interface 1)
 (belongs to Activity Template Modelling Functions in Interface 1)

NAME

WMDeleteActivityTemplate – Deletes an activity template.

DESCRIPTION

```
WMErrRetType WMDeleteActivityTemplate (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id
)
```

Argument	Description
psession_handle	Pointer to the structure with the session information created by a call to WMConnect.
pactivity_template_id	Pointer to the activity template ID for the activity template to be deleted.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE

A.3.3 Moving Activity Instances

WMMoveActivityInstance

(belongs to WAPI Activity Control Functions)

NAME

WMMoveActivityInstance – Moves an existing activity instance from its original place between two groups of activity instances.

DESCRIPTION

```

WMTErrRetType WMMoveActivityInstance (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPActivityInstID[] pbefore_activity_inst_id,
    in    WMTPActivityInstID[] pafter_activity_inst_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance that is to be moved.
pbefore_activity_inst_id	Pointer to the activity instance ID which is determined to be before the moved activity instance.
pafter_activity_inst_id	Pointer to the activity instance ID which is determined to be after the moved activity instance.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_BEFORE_INSTANCES
WM_INVALID_AFTER_INSTANCES

```

A.3.4 Setting and Updating Attribute Values

WMAssignActivityInstanceAttribute (already existing in [WfMC98], p. 52)
(belongs to WAPI Activity Control Functions)

NAME

WMAssignActivityInstanceAttribute – Sets or changes attribute values.

DESCRIPTION

```

WMTErrRetType WMAssignActivityInstanceAttribute (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length,
    in    WMTPTText pattribute_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance for which the attribute will be assigned.
pattribute_name	Pointer to the name of the attribute.
attribute_type	Type of the attribute.
attribute_length	Length of the attribute value.
ppattribute_value	Pointer to a buffer area where the attribute value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_ASSIGNMENT_FAILED

```

WMAssignActivityTemplateAttributeValue (Interface 1)
 (belongs to WAPI Activity Template Manipulation Functions)

NAME

WMAssignActivityTemplateAttributeValue - Sets an attribute of an activity template.

DESCRIPTION

```

WMTErrRetType WMAssignActivityTemplateAttribute (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplate pactivity_template,
    in    WMTPAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length,
    in    WMTPText pattribute_value
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template	Pointer to a structure containing the activity template from which the attribute is being retrieved.
pattribute_name	Pointer to the name of the attribute to put the value into.
attribute_type	Type of the attribute.
attribute_length	Length of the attribute value.
pattribute_value	Pointer to a buffer area where the attribute value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_ASSIGNMENT_FAILED
  
```

A.3.5 Inserting Attributes

WMInsertActivityInstanceAttribute

(belongs to WAPI Activity Control Functions)

NAME

WMInsertActivityInstanceAttribute – Inserts a new attribute into an activity instance.

DESCRIPTION

```
WMTerrRetType WMAssignActivityInstanceAttribute (
    in    WMTPSessionHandle psession_handle,
    in    WMTProcInstID pproc_inst_id,
    in    WMTActivityInstID pactivity_inst_id,
    in    WMTAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length
)
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing the unique process instance ID.
pactivity_inst_id	Pointer to a structure containing the activity instance identification for which the attribute will be assigned.
pattribute_name	Pointer to the name of the attribute.
attribute_type	Type of the attribute.
attribute_length	Length of the attribute value.

ERROR RETURN VALUE

```
WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_INSERTION_FAILED
```

WMInsertActivityTemplateAttribute (Interface 1)
 (belongs to WAPI Activity Template Manipulation Functions)

NAME

WMInsertActivityTemplateAttribute – Insert a new attribute into an activity template.

DESCRIPTION

Note that at the moment only fully specified attributes (name, type and length) can be inserted into an activity template.

```

WMTErrRetType WMInsertActivityTemplateAttribute (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTPAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to a structure containing the activity template identification for which the attribute will be assigned.
pattribute_name	Pointer to the name of the attribute.
attribute_type	Type of the attribute.
attribute_length	Length of the attribute value.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_INSERTION_FAILED
  
```

A.3.6 Deleting Attributes

WMDeleteActivityInstanceAttribute

(belongs to WAPI Activity Control Functions)

NAME

WMDeleteActivityInstanceAttribute – Deletes an attribute assigned to an activity instance.

DESCRIPTION

```

WMTErrRetType WMDeleteActivityInstanceAttribute (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPAttrName pattribute_name
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance the attribute is assigned to.
pattribute_name	Pointer to the name of the attribute to be deleted.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_ATTRIBUTE
  
```

WMDeleteActivityTemplateAttribute (Interface 1)
 (belongs to WAPI Activity Template Manipulation Functions)

NAME

WMDeleteActivityTemplateAttribute – Deletes an attribute assigned to an activity template.

DESCRIPTION

```

WMTErrRetType WMDeleteActivityTemplateAttribute (
    in      WMTTPSessionHandle psession_handle,
    in      WMTTPActivityTemplateID pactivity_template_id,
    in      WMTTPAttrName pattribute_name
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to the ID of the activity template the attribute is assigned to.
pattribute_name	Pointer to the name of the attribute to be deleted.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE
 WM_INVALID_ATTRIBUTE

A.3.7 Role/User Assignment

WMAssignActivityInstanceParticipants

(belongs to WAPI Activity Control Functions)

NAME

WMAssignActivityInstanceParticipants – Assigns one or up to ten workflow participants to an activity instance.

DESCRIPTION

```

WMTErrRetType WMAssignActivityInstanceParticipants (
    in    WMTPSessionHandle psession_handle,
    in    WMTPWflParticipant[] pparticipants[10],
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pparticipants[10]	The identification of the workflow participant(s) who are to be assigned. A field of the array is NULL for every participant less than 10.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance to which the participant(s) are to be assigned.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_INVALID_WORKFLOW_PARTICIPANT

```


WMAssignActivityTemplateParticipants (Interface 1)
 (belongs to WAPI Activity Template Manipulation Functions)

NAME

WMAssignActivityTemplateParticipants – Assigns one or up to ten workflow participants to an activity template.

DESCRIPTION

```

WMTErrRetType WMAssignActivityTemplateParticipants (
    in      WMTPSessionHandle psession_handle,
    in      WMTPWflParticipant[] pparticipants[10],
    in      WMTPActivityTemplateID pactivity_template_id
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pparticipants[10]	The identification of the workflow participant(s) who are to be assigned. A field of the array is NULL for every participant less than 10.
pactivity_template_id	Pointer to the ID of the activity template to which the participant(s) are to be assigned.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE
 WM_INVALID_WORKFLOW_PARTICIPANT

A.3.8 Time Attributes Assignment

WMAssignActivityInstanceDuration

(belongs to WAPI Activity Control Functions)

NAME

WMAssignActivityInstanceDuration – Assigns a duration to an activity instance.

DESCRIPTION

```

WMTErrRetType WMAssignActivityInstanceDuration (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTInt32 duration_limit_type,
    in    WMTPInt16 pduration_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance to which the duration is to be assigned.
duration_limit_type	Limit type of the duration, <i>minimum</i> or <i>maximum</i> duration.
pduration_value	Pointer to a buffer area where the duration value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_DURATION_ASSIGNMENT_FAILED

```

WMAssignActivityInstanceTime

(belongs to WAPI Activity Control Functions)

NAME

WMAssignActivityInstanceTime – Assigns a (starting, finishing) time to an activity instance.

DESCRIPTION

```

WMTErrRetType WMAssignActivityInstanceTime (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTInt32 time_period_type,
    in    WMTPDate ptime_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance to which the time is to be assigned.
time_period_type	Period type of the time, <i>earliest/latest starting/finishing</i> time.
ptime_value	Pointer to a buffer area where the time value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_TIME_ASSIGNMENT_FAILED

```

WMAssignActivityInstanceWUT

(belongs to WAPI Activity Control Functions)

NAME

WMAssignActivityInstanceWUT – Assigns a warm-up duration to an activity instance.

DESCRIPTION

```

WMTErrRetType WMAssignActivityInstanceWUT (
    in    WMTPSessionHandle psession_handle,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPActivityInstID pactivity_inst_id,
    in    WMTPInt32 wut_limit_type,
    in    WMTPInt16 pwut_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pactivity_inst_id	Pointer to the ID of the activity instance to which the warm-up duration is to be assigned.
wut_limit_type	Limit type of the duration, <i>minimum</i> or <i>maximum</i> warm-up duration.
pwut_value	Pointer to a buffer area where the warm-up duration value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_ACTIVITY_INSTANCE
WM_WUT_ASSIGNMENT_FAILED

```

WMAssignActivityTemplateDuration (Interface 1)
 (belongs to WAPI Activity Template Attribute Manipulation Functions)

NAME

WMAssignActivityTemplateDuration – Assigns a duration to an activity template.

DESCRIPTION

```

WMTErrRetType WMAssignActivityTemplateDuration (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTInt32 duration_limit_type,
    in    WMTPInt16 pduration_value
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to a structure containing the activity template identification for which the duration will be assigned.
duration_limit_type	Limit type of the duration, <i>minimum</i> or <i>maximum</i> duration.
pduration_value	Pointer to a buffer area where the duration value will be placed.

ERROR RETURN VALUE

WM_SUCCESS
 WM_INVALID_SESSION_HANDLE
 WM_INVALID_ACTIVITY_TEMPLATE
 WM_DURATION_ASSIGNMENT_FAILED

WMAssignActivityTemplateTime (Interface 1)

(belongs to WAPI Activity Template Attribute Manipulation Functions)

NAME**WMAssignActivityTemplateTime** – Assigns a (starting, finishing) time to an activity template.**DESCRIPTION**

```

WMTErrRetType WMAssignActivityTemplateTime (
    in    WMTPSessionHandle psession_handle,
    in    WMTPActivityTemplateID pactivity_template_id,
    in    WMTInt32 time_period_type,
    in    WMTPDate ptime_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to a structure containing the activity template identification for which the time will be assigned.
time_period_type	Period type of the time, <i>earliest/latest starting/finishing</i> time.
ptime_value	Pointer to a buffer area where the time value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE
WM_TIME_ASSIGNMENT_FAILED

```

WMAssignActivityTemplateWUT (Interface 1)

(belongs to WAPI Activity Template Attribute Manipulation Functions)

NAME**WMAssignActivityTemplateWUT** – Assigns a warm-up duration to an activity template.**DESCRIPTION**

```

WMTErrRetType WMAssignActivityTemplateWUT (
    in      WMTPSessionHandle psession_handle,
    in      WMTPActivityTemplateID pactivity_template_id,
    in      WMTInt32 wut_limit_type,
    in      WMTPInt16 pwut_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pactivity_template_id	Pointer to a structure containing the activity template identification for which the warm-up duration will be assigned.
wut_limit_type	Limit type of the duration, <i>minimum</i> or <i>maximum</i> warm-up duration.
pwut_value	Pointer to a buffer area where the warm-up duration value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_ACTIVITY_TEMPLATE
WM_WUT_ASSIGNMENT_FAILED

```

A.3.9 Keeping Modified Process Instances

WMKeepProcessInstance

(belongs to WAPI Process Control Functions)

NAME

WMKeepProcessInstance – Stores the process definition corresponding to a modified process instance.

DESCRIPTION

```

WMTErrRetType WMKeepProcessInstance (
    in    WMTPSessionHandle psession_handle,
    in    WMTProcInstID pproc_inst_id,
    out   WMTProcDefID pproc_def_id
)
  
```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing a unique process instance ID.
pproc_def_id	Pointer to the ID of the new process definition.

ERROR RETURN VALUE

WM_SUCCESS

WM_INVALID_SESSION_HANDLE

WM_INVALID_PROCESS_INSTANCE

Note: Is it possible that the process definition could not be created for another reason?

A.3.10 Inserting Sub-Workflows

WMInsertProcInstance

(belongs to WAPI Process Control Functions)

NAME

WMInsertProcInstance – The process instance provided is inserted into another process instance as a subworkflow.

DESCRIPTION

Note that the process instance has to start with a single start activity and end with one single end activity.

One single activity that represents both the start and the end activity is also allowed.

```

WMTErrRetType WMInsertProcessInstance (
    in      WMTPSessionHandle psession_handle,
    in      WMTPProcInstID phost_proc_inst_id,
    in      WMTPProcInstID pinsert_proc_inst_id,
    in      WMTPActivityInstID[] pbefore_activity_inst_id,
    in      WMTPActivityInstID[] pafter_activity_inst_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
phost_proc_inst_id	Pointer to a structure containing the unique ID of the process instance in which the sub-workflow is to be inserted.
pinsert_proc_inst_id	Pointer to a structure containing the unique ID of the process instance which is to be inserted.
pbefore_activity_inst_id[]	Pointer array to the activity instance IDs that are determined to be before the newly inserted activity instance.
pafter_activity_inst_id[]	Pointer array to the activity instance IDs that are determined to be after the newly inserted activity instance.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_BEFORE_ACTIVITY_INSTANCE
WM_INVALID_AFTER_ACTIVITY_INSTANCE

```

A.3.11 Managing Work-lists

WMReassignWorkItem (already existing in [WfMC98], p. 73)
(belongs to WAPI Work-list/Work-item Handling Functions)

NAME

WMReassignWorkItem – Reassigns a work-item from one workflow participant’s work-list to another workflow participant’s work-list.

DESCRIPTION

```

WMTErrRetType WMReassignWorkItem (
    in    WMTPSessionHandle psession_handle,
    in    WMTPWflParticipant psource_user,
    in    WMTPWflParticipant ptarget_user,
    in    WMTPProcInstID pproc_inst_id,
    in    WMTPWorkItemId pwork_item_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
psource_user	The identification of a workflow participant from which work is to be reassigned.
ptarget_user	The identification of a workflow participant to whom work is to be assigned.
pproc_inst_id	Pointer to a structure containing the unique ID of the process instance.
pwork_item_id	Pointer to a structure containing the unique ID of the work item that is to be reassigned.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_WORKITEM
WM_INVALID_SOURCE_USER
WM_INVALID_TARGET_USER

```

WMAssignWorkItemAttribute (already existing in [WfMC98], p. 78)
(belongs to WAPI Work-list/Work-item Handling Functions)

NAME

WMAssignWorkItemAttribute – Sets or updates the value of an attribute of a work-item.

DESCRIPTION

```

WMTerrRetType WMAssignWorkItemAttribute (
    in    WMTPSessionHandle psession_handle,
    in    WMTProcInstID pproc_inst_id,
    in    WMTWorkItemID pwork_item_id
    in    WMTAttrName pattribute_name,
    in    WMTInt32 attribute_type,
    in    WMTInt32 attribute_length,
    in    WMTPText pattribute_value
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing the unique ID of the process instance.
pwork_item_id	Pointer to a structure containing the unique ID of the work item for which an attribute will be set or updated.
pattribute_name	Pointer to the name of the attribute.
attribute_type	Type of the attribute.
attribute_length	Length of the attribute value.
pattribute_value	Pointer to a buffer area where the attribute value will be placed.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_WORKITEM
WM_INVALID_ATTRIBUTE
WM_ATTRIBUTE_ASSIGNMENT_FAILED

```

WMDeleteWorkItem

(belongs to WAPI Work-list/Work-item Handling Functions)

NAME**WMDeleteWorkItem** – Deletes a work-item in a given work-list.**DESCRIPTION**

```

WMTErrRetType WMDeleteWorkItem (
    in    WMTPSessionHandle psession_handle,
    in    WMTProcInstID pproc_inst_id,
    in    WMTWorkItemID pwork_item_id
)

```

Argument Name	Description
psession_handle	Pointer to a structure containing information about the context for this action.
pproc_inst_id	Pointer to a structure containing the unique ID of the process instance.
pwork_item_id	Pointer to a structure containing the unique ID of the work item that is to be deleted.

ERROR RETURN VALUE

```

WM_SUCCESS
WM_INVALID_SESSION_HANDLE
WM_INVALID_PROCESS_INSTANCE
WM_INVALID_WORKITEM

```

A.4 WAPI Data Types Addendum

Activity Template Data Types

```
typedef struct
{
    WMTText          activity_template_id[UNIQUE_ID_SIZE];
} WMTActivityTemplateID;
```

```
typedef WMTActivityTemplateID *WMTPActivityTemplateID;
```

```
typedef struct
{
    // This is the minimum list of elements at this time.
    // Future versions to provide extensibility for this structure.
    WMTText          activity_template_name[NAME_STRING_SIZE];
    WMTActivityTemplateID activity_template_id;
} WMTActivityTemplate;
```

```
typedef WMTActivityTemplate *WMTPActivityTemplate;
```

Classification Category Data Types

```
typedef struct
{
    WMTText          classification_category_id[UNIQUE_ID_SIZE];
} WMTClassificationCategoryID;
```

```
typedef WMTClassificationCategoryID *WMTPClassificationCategoryID;
```

```
typedef struct
{
    WMTText          classification_category_name[UNIQUE_ID_SIZE];
    WMTClassificationCategoryID classification_category_id;
} WMTClassificationCategory;
```

```
typedef WMTClassificationCategory *WMTPClassificationCategory;
```

Process Definition Data Types

```
typedef struct
{
    // This definition extends the definition of WMTProcDef
```

```

// given in the document [WfMC98], p. 14.
// Two new elements are added:
// The new element ancestor_proc_def_id contains the ID of the former process instance,
// if the latter has been modified to create the one at hand.
// This element is NULL, if there is no such ancestor.
    WMTText                process_name[NAME_STRING_SIZE];
    WMTProcDefID           proc_def_id;
    WMTProcDefState        state;
    WMTProcDefID           ancestor_proc_def_id;
} WMTProcDef;

```

Association Data Types

```

typedef struct
{
    WMTClassificationCategoryID  classification_category_id;
    WMTProcDefID                 proc_def_id;
} WMTAssociationProcDefClassificationCategory;

```

```

typedef struct
{
    WMTClassificationCategoryID  classification_category_id;
    WMTActivityTemplateID        activity_template_id;
} WMTAssociationActivityTemplateClassificationCategory;

```

Time Data Types

```

typedef WMTUInt32 WMTDate;
typedef WMTDate *WMTDate;

```

A.5 WAPI Error Return Codes Addendum

WM_INVALID_ACTIVITY_TEMPLATE

Indicates that the activity template ID that was passed as a parameter to an API call was not valid, or it was not recognized by the servicing workflow engine.

WM_INVALID_CLASSIFICATION_CATEGORY

Indicates that the classification category ID that was passed as a parameter to an API call was not valid, or it was not recognized by the servicing workflow engine.

WM_INVALID_BEFORE_INSTANCES

Can occur when an activity instance is inserted or moved.

Indicates that provided IDs for activity instances that are to be before the inserted/moved activity instance are not valid. *I.e.* the activity instance to be inserted or moved cannot be placed behind one or more of the specified before instances.

WM_INVALID_AFTER_INSTANCES

Can occur when an activity instance is inserted or moved.

Indicates that provided IDs for activity instances that are to be after the inserted/moved activity instance are not valid. *I.e.* the activity instance to be inserted or moved cannot be placed before one or more of the specified before instances.

WM_INVALID_WORKFLOW_PARTICIPANT

Indicates that at least one of the participants that was passed (in an array) as a parameter to an API call was not valid, or was not recognized by the servicing workflow engine.

WM_ATTRIBUTE_INSERTION_FAILED

Indicates that the workflow engine was not able to complete the attribute insertion requested.

WM_DURATION_ASSIGNMENT_FAILED

Indicates that the workflow engine was not able to complete the duration assignment requested.

WM_TIME_ASSIGNMENT_FAILED

Indicates that the workflow engine was not able to complete the time assignment requested.

WM_WUT_ASSIGNMENT_FAILED

Indicates that the workflow engine was not able to complete the WUT assignment requested.

References of Appendix A

- [BRK+03] Bassil, S., Rolli, D., Keller, R.K., and Kropf, P., Extending the Workflow Reference Model to Accommodate Dynamism: Workflow Management Application Programming Interface (Interfaces 1, 2, and 3) Specification. *Technical Report GELO-152*, Université de Montréal, Canada, March 2003.
- [WfMC95] Workflow Management Coalition, The Workflow Reference Model. *WFMC-TC-1003*, Version 1.1, January 1995. On-line at <<http://www.wfmc.org/standards/docs/tc003v11.pdf>>.
- [WfMC98] Workflow Management Coalition, Workflow Management Application Programming Interface (Interface 2&3) Specification. *WFMC-TC-1009*, Version 2.0, July 1998. On-line at <<http://www.wfmc.org/standards/docs/if2v20.pdf>>.
- [WfMC04] The Workflow Management Coalition (2004). On-line at <<http://www.wfmc.org>>.