Our research addresses the problem of managing and coordinating information resources used in information system (IS) development. In our approach, we argue that conceptual specifications of information systems are the fundamental constituents of information resources. Accordingly, the first challenge of the research is to manage and coordinate conceptual specifications used in IS development. For this reason, an "Information System upon Information Systems" (ISIS) is proposed to manage and coordinate information resources based on conceptual specifications of information systems. An ISIS is actually a typical information system, which supports other information systems in development. Correspondingly, this thesis presents a conceptual framework for building and managing the ISIS. This conceptual framework describes how to identify, modularize, manage, represent, and coordinate conceptual specifications used in IS development. In conclusion, the thesis ends with the current results, present works and remaining research issues.
INFORMATION SYSTEM UPON INFORMATION SYSTEMS: A conceptual framework

THESE
Présentée à la Faculté des sciences économiques et sociales de L'Université de Genève

par
Thang LE DINH

Pour l’obtention du grade de Docteur ès Sciences Economiques et Sociales, Mention : Systèmes d’Information

Membres du jury de thèse:
Mme Bich-Thuy DONG THI, professeur, co-directrice de thèse, Université de HoChiMinh ville, VietNam
M.Thibault ESTIER, professeur assistant, Université de Lausanne
M.Dimitri KONSTANTAS, professeur, Genève, président du jury
M.Michel LEONARD, professeur, Genève, directeur de thèse
Mme Jolita RALYTE, Genève, maître assistant
Mme Colette ROLLAND, professeur, Université de Paris Sorbonne, France

Thèse no 577
Genève, 2004
La Faculté des sciences économiques et sociales, sur préavis du jury, a autorisé l’impression de la présente thèse, sans entendre, par là, émettre aucune opinion sur les propositions qui s’y trouvent énoncées et qui n’engagent que la responsabilité de leur auteur.

Genève, le 19 Novembre 2004

Le doyen
Pierre ALLAN

Impression d’après le manuscrit de l’auteur.

ACKNOWLEDGMENTS

First of all, I would like to thank my advisor, Prof. Michel LEONARD for his invaluable help, generous support and extreme patient.

Accordingly, I would like to thank Prof. Bich Thuy DONG THI for her encouragement and guidance throughout my work at the Computer Science Center – University of Natural Sciences before doing my Ph.D.

I would like to thank other members of my committee, Prof. Dimitri KONSTANTAS, Prof. Colette ROLLAND, Prof. Thibault ESTIER, and Dr. Jolita RALYTE, for valuable comments and feedback on my thesis.

I am grateful to all the collaborators, past and present, from the Centre Universitaire d’Informatique and the MATIS Geneva team, for their friendship and thoughtfulness. I am grateful to the Abdelaziz KHADRAOUI, Huong Tram Hong LUU, Thanh Thoa PHAM THI, Mehdi SNENE, Slim TURKI, Nicolas ARNI-BLOCH, and Jorge PARDELLAS, for their discussions and interests in my research. Special thanks to Jolita RALYTE for her valuable feedback on several papers.

Very special thanks go to my parents and sisters for their continuing support and encouragement.

Finally, I am grateful to my beloved wife, Kim Ngan, and my two sons, Anh-Minh and Anh-Quan, for their unwavering love, support and patient throughout our years at Geneva.
ABSTRACT

INFORMATION SYSTEM UPON INFORMATION SYSTEMS: CONCEPTUAL FRAMEWORK

by Thang LE DINH

Our research addresses the problem of managing and coordinating information resources used in information system (IS) development. In our approach, we argue that conceptual specifications of information systems are the fundamental constituents of information resources. Accordingly, the first challenge of the research is to manage and coordinate conceptual specifications used in IS development.

For this reason, an Information System upon Information Systems (ISIS) is proposed to manage and coordinate information resources based on conceptual specifications of information systems. An ISIS is actually a typical information system, which supports other information systems in development.

Correspondingly, this thesis presents a conceptual framework for building and managing the ISIS. This conceptual framework describes how to identify, modularize, manage, represent, and coordinate conceptual specifications used in IS development.

In conclusion, the thesis ends with the current results, present works and remaining research issues.
# TABLE OF CONTENTS

## ACKNOWLEDGEMENTS

AC-...I

## ABSTRACT

AB-...IV

## TABLE OF CONTENTS

T-...V

## LIST OF ACRONYMS

LIST OF ACRONYMS............................X

## LIST OF FIGURES

LIST OF FIGURES.............................XI

## LIST OF TABLES

LIST OF TABLES...............................XIII

### CHAPTER 1. INTRODUCTION

1.1. Introduction..........................................................1
1.1.1. Context ......................................................................1
1.1.2. Motivation .................................................................1
1.2. Information resources at the Information level ....................2
1.2.1. Specifications .............................................................2
1.2.2. Importance of specifications .........................................3
1.2.3. Conceptual and Technical specifications ..........................4
1.2.4. Information layer .........................................................5
1.3. An Information System upon Information Systems ...............6
1.3.1. Definition .................................................................6
1.3.2. Objective of an ISIS ....................................................6
1.3.3. Challenges to an ISIS ..................................................7
1.4. Thesis outline ................................................................8
1.4.1. Conceptual framework for an ISIS ...............................8
1.4.2. Thesis structure ................................................................................................. 9

CHAPTER 2. STATE-OF-ART ................................................................. 11
2.1. Industrial experience ...................................................................................... 11
2.1.1. Context .......................................................................................................... 11
2.1.2. Products of the CSC .................................................................................. 12
2.1.3. Lessons learned .......................................................................................... 13
2.1.4. Discussion .................................................................................................... 17
2.2. State-Of-Art .................................................................................................... 17
2.2.1. Related work ................................................................................................ 18
2.2.2. Discussion .................................................................................................... 20

CHAPTER 3. SPECIFICATION IDENTIFICATION ......................... 24
3.1. Introduction ...................................................................................................... 24
3.2. The Static aspect ............................................................................................. 25
3.2.1. Static concepts ............................................................................................ 25
3.2.2. Interrelationships between static concepts .............................................. 27
3.2.3. Static diagrams .......................................................................................... 29
3.3. The Dynamic aspect ...................................................................................... 33
3.3.1. Dynamic concepts ...................................................................................... 33
3.3.2. Dynamic diagrams ..................................................................................... 40
3.4. The Integrity rule aspect ................................................................................ 43
3.4.1. Integrity rule concepts ................................................................................ 43
3.4.2. Interrelationships between the Integrity rule concepts .......................... 44
3.5. BNF representation ....................................................................................... 46
3.5.1. BNF representation for the Static aspect .................................................. 46
3.5.2. BNF representation for the Dynamic aspect ............................................. 47
3.5.3. BNF representation for the Integrity rule aspect ..................................... 47
3.5.4. An example about conceptual specifications of an IS. ............................ 48
3.6. Conclusion ...................................................................................................... 48

CHAPTER 4. SPECIFICATION MODULARIZATION ............... 49
4.1. Introduction ...................................................................................................... 49
4.2. Information System component .................................................................... 50
4.2.1. Context ........................................................................................................ 50
4.2.2. Proposition .................................................................................................. 50
4.2.3. Overview of an Information System component ..................................... 51
CHAPTER 5. SPECIFICATION MANAGEMENT .................. 77

5.1. Overview of an ISIS ....................................................... 77
5.2. Management of specifications ........................................ 79
  5.2.1. Organizational aspect of an ISIS ............................... 79
  5.2.2. An example about the Organizational aspect ................. 81
5.3. Architecture of an ISIS .................................................. 86
  5.3.1. Three layers of an ISIS ........................................... 86
  5.3.2. Overall architecture of an ISIS ................................ 88
  5.3.3. Coordinating with development tools .......................... 89
5.4. IS project management with the ISIS .............................. 93
  5.4.1. ISIS Development process ..................................... 94
  5.4.2. ISIS project management ....................................... 95
5.5. Conclusion ................................................................. 100

CHAPTER 6. SPECIFICATION REPRESENTATION ............... 102

6.1. Principles of the Representation layer .......................... 102
  6.1.1. Objects of an ISIS ................................................ 102
  6.1.2. Private and Common workspaces ................................ 103
  6.1.3. Interpretation of the Representation layer ................... 103
6.2. Structure of conceptual specifications ........................... 104
  6.2.1. Structure of specifications of the static concepts .......... 104
  6.2.2. Structure of specifications of the dynamic concepts .......... 107
  6.2.3. Structure of specifications of the integrity rule concepts ... 109
  6.2.4. An example about specification representation ............... 110
6.3. Behaviour of conceptual specifications ............................................ 112
  6.3.1. Conformity rules ............................................................................. 112
  6.3.2. Generic dynamic states ................................................................. 113
  6.3.3. Adaptation of a generic object life cycle ....................................... 116
6.4. Coherence of conceptual specifications ............................................ 117
  6.4.1. Identifying integrity rules .............................................................. 118
  6.4.2. Specifying integrity rules .............................................................. 119
  6.4.3. Enforcing integrity rules ............................................................... 119
6.5. Conclusion ......................................................................................... 121

CHAPTER 7. SPECIFICATION COORDINATION ......................... 122

7.1. Information coordination ................................................................. 122
  7.1.1. What is coordination ....................................................................... 122
  7.1.2. Theory of coordination ................................................................. 122
  7.1.3. Information coordination .............................................................. 123
  7.1.4. Information coordination in development ....................................... 124
7.2. Coordination in development supported by an ISIS .............................. 124
  7.2.1. An ISIS supporting coordination in development .......................... 124
  7.2.2. Intra-IS coordination and Inter-IS coordination ................................ 125
  7.2.3. Common objects and Private objects .............................................. 126
7.3. Intra-IS coordination .......................................................................... 128
  7.3.1. Interdependencies between specifications ...................................... 128
  7.3.2. Intra-IS coordination supported by the ISIS .................................... 132
7.4. Inter-IS coordination .......................................................................... 137
  7.4.1. Interdependences between IS components ...................................... 137
  7.4.2. Inter-IS coordination supported by the ISIS .................................... 141
7.5. Conclusion ......................................................................................... 148

CHAPTER 8. CONCLUSION .......................................................... 150

8.1. Conclusion ......................................................................................... 150
8.2. Present works ................................................................................... 151
8.3. Future works .................................................................................... 152

ANNEX 1. AN EXAMPLE ABOUT CONCEPTUAL SPECIFICATIONS OF AN INFORMATION SYSTEM..... 154

Specification of the Static aspect ............................................................... 154
  -- Atomic-classes .................................................................................. 154
-- Tuple-classes ................................................................. 154
-- Hyperclasses ............................................................... 155

Specification of the Dynamic aspect .................................... 155
-- Dynamic states ............................................................ 155
-- Methods 157
-- Events 158
-- Processes ................................................................. 158

Specification of the Integrity rule aspect .............................. 159
-- Integrity rules ............................................................ 159
-- Method actions ............................................................ 161

ANNEX 2. META-MODEL OF THE M7 REFERENCE MODEL.. 162

The Static aspect .................................................................. 162
Meta-model of the Static aspect ............................................. 162
Conformity rules ............................................................... 163

The Dynamic aspect ........................................................... 164
Meta-model of the Dynamic aspect ....................................... 164
Conformity rules ............................................................... 166

The Integrity rule aspect ....................................................... 167
Meta-model of the Integrity rule aspect ................................. 167
Conformity rules ............................................................... 168

ANNEX 3. COORDINATION OF OBJECT LIFE CYCLES OF GENERIC HYPERCLASSES ........................................ 170

ATOMIC-CLASS generic hyperclass ....................................... 170
HYPERCLASS generic hyperclass .......................................... 171
ATTRIBUTE generic hyperclass ............................................ 174
KEY generic hyperclass ..................................................... 175
EVENT generic hyperclass .................................................. 176
PROCESS generic hyperclass ............................................... 177
DYNAMIC STATE generic hyperclass ................................... 178
METHOD generic hyperclass .............................................. 179
INTEGRITY RULE generic hyperclass ................................... 180
SCOPE generic hyperclass .................................................. 181
RISK generic hyperclass .................................................... 182

REFERENCES ..................................................................... 184
# LIST OF ACRONYMS

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>4GL</td>
<td>Fourth Generation Language</td>
</tr>
<tr>
<td>BNF</td>
<td>Backus Naur Form</td>
</tr>
<tr>
<td>CASE</td>
<td>Computer Aided System Engineering</td>
</tr>
<tr>
<td>CBD</td>
<td>Component Based Development</td>
</tr>
<tr>
<td>CSC</td>
<td>Computer Science Centre of University of Natural Sciences at Ho Chi Minh city</td>
</tr>
<tr>
<td>CSC-ISIS</td>
<td>The Information System upon Information Systems for Computer Science Centre</td>
</tr>
<tr>
<td>DBMS</td>
<td>Database Management System</td>
</tr>
<tr>
<td>ERP</td>
<td>Enterprise Resource Planning</td>
</tr>
<tr>
<td>IDE</td>
<td>Integrated Development Environment</td>
</tr>
<tr>
<td>IS</td>
<td>Information system</td>
</tr>
<tr>
<td>ISC</td>
<td>Information System Component</td>
</tr>
<tr>
<td>IS-CBD</td>
<td>Information System Component Based Development</td>
</tr>
<tr>
<td>ISIS</td>
<td>Information System upon Information Systems</td>
</tr>
<tr>
<td>IT</td>
<td>Information technology</td>
</tr>
<tr>
<td>KBMS</td>
<td>Knowledge Based Management System</td>
</tr>
<tr>
<td>MATIS-IS</td>
<td>The Information system for the MATIS enterprise</td>
</tr>
<tr>
<td>MDA</td>
<td>Model Driven Architecture</td>
</tr>
<tr>
<td>MDIS project</td>
<td>“Manpower Development and promotion of Information System in Vietnam National University at Ho-Chi-Minh city” project</td>
</tr>
<tr>
<td>OID</td>
<td>Object Identifier</td>
</tr>
<tr>
<td>OLC</td>
<td>Object life-cycle</td>
</tr>
<tr>
<td>RZ</td>
<td>Responsibility Zone</td>
</tr>
<tr>
<td>SCM</td>
<td>Software Configuration Management</td>
</tr>
<tr>
<td>SLC</td>
<td>System Life-cycle</td>
</tr>
<tr>
<td>UNS-HCM</td>
<td>University of Natural Sciences at Ho-Chi-Minh city</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>1-1</td>
<td>The Information layer</td>
</tr>
<tr>
<td>2-1</td>
<td>Architecture of the IAS product.</td>
</tr>
<tr>
<td>2-2</td>
<td>The Cash-in-hand IAS component.</td>
</tr>
<tr>
<td>3-1</td>
<td>The class diagram for all the classes of the Sale/Collection process.</td>
</tr>
<tr>
<td>3-2</td>
<td>The class diagram for the Sale hyperclass.</td>
</tr>
<tr>
<td>3-3</td>
<td>The Object diagram of the Sale Order tuple-class</td>
</tr>
<tr>
<td>3-4</td>
<td>The HoS#1 hyper-object of the Sale hyperclass.</td>
</tr>
<tr>
<td>3-5</td>
<td>A dynamic state as a sub-hyperclass.</td>
</tr>
<tr>
<td>3-6</td>
<td>Dynamic states of the tuple-classes related to Sale/Collection process.</td>
</tr>
<tr>
<td>3-7</td>
<td>System life cycle.</td>
</tr>
<tr>
<td>3-8</td>
<td>Object life cycle.</td>
</tr>
<tr>
<td>4-1</td>
<td>Information System components</td>
</tr>
<tr>
<td>4-2</td>
<td>External view of IS components.</td>
</tr>
<tr>
<td>4-3</td>
<td>Internal view of an IS component</td>
</tr>
<tr>
<td>4-4</td>
<td>The three IS components of the MATIS-IS.</td>
</tr>
<tr>
<td>4-5</td>
<td>The Static aspects of the Sale management and the Inventory IS components.</td>
</tr>
<tr>
<td>4-6</td>
<td>The Dynamic aspects of the Sale management and Inventory management IS components</td>
</tr>
<tr>
<td>4-7</td>
<td>Information coordination between IS components and their responsibility zones</td>
</tr>
<tr>
<td>4-8</td>
<td>Overlap situations</td>
</tr>
<tr>
<td>4-9</td>
<td>Schemata and Data overlap layers.</td>
</tr>
<tr>
<td>4-10</td>
<td>Overlap situations between Sale management and Inventory management IS components</td>
</tr>
<tr>
<td>4-11</td>
<td>Collaboration between the Sale management and Inventory management IS components</td>
</tr>
<tr>
<td>4-12</td>
<td>Meta-model of the Component aspect</td>
</tr>
<tr>
<td>4-13</td>
<td>IS development with IS components.</td>
</tr>
<tr>
<td>5-1</td>
<td>Meta-model of the Organizational aspect.</td>
</tr>
<tr>
<td>5-2</td>
<td>The IAS#1 team and its contributors.</td>
</tr>
<tr>
<td>5-3</td>
<td>Information resources of the CSC-ISIS.</td>
</tr>
<tr>
<td>5-4</td>
<td>Activities of the standard development process of the CSC.</td>
</tr>
<tr>
<td>5-5</td>
<td>Some conceptual specifications of the IS#1.</td>
</tr>
<tr>
<td>5-6</td>
<td>Some technical specifications of the IS#1.</td>
</tr>
<tr>
<td>5-7</td>
<td>Roles of the CSC-ISIS</td>
</tr>
<tr>
<td>5-8</td>
<td>Overall architecture of an ISIS.</td>
</tr>
<tr>
<td>5-9</td>
<td>The ISIS and development tools.</td>
</tr>
<tr>
<td>5-10</td>
<td>Two levels of integration</td>
</tr>
<tr>
<td>5-11</td>
<td>Direct integration mode</td>
</tr>
<tr>
<td>5-12</td>
<td>Replication integration mode</td>
</tr>
<tr>
<td>5-13</td>
<td>Agent integration mode</td>
</tr>
</tbody>
</table>
# LIST OF TABLES

Table 2-1: Comparisons the ISIS approach with the other approaches. ........................................21
Table 3-1: Attributes of the Customer tuple-class ........................................................................28
Table 3-2: Attributes of the Sale hyperclass ................................................................................28
Table 3-3: The interpretations of the hoS#1 and hoS#2 hyper-objects ........................................32
Table 3-4: The dynamic states of the tuple-classes related to Sale/Collection process ....................36
Table 3-5: Methods of Sale order, Customer, Inventory item and Payment tuple-classes. ..............37
Table 3-6: The events related to Sale/Collection process .................................................................38
Table 3-7: Processes of the Sale hyperclass ....................................................................................40
Table 3-8: Some typical integrity rules ...........................................................................................43
Table 3-9: List of scopes ................................................................................................................44
Table 3-10: List of risks ..................................................................................................................46
Table 4-1: The Integrity rule aspect of the Sale management IS component ..................................58
Table 5-1: Main tasks of the Analysis and Design phases ..............................................................100
Table 5-2: The tasks and involved human resources for the Analysis and Design phases ..............100
Table 6-1: Generic methods of a generic hyperclass of the ISIS .....................................................115
Table 6-2: Specific methods of the Hyperclass generic hyperclass of the ISIS ...............................117
Table 6-3: Integrity rules originating from the validity rules related to the Hyperclass generic hyperclass ..................................................................................................................118
Table 6-4: Integrity rules originating from the completeness rules related to Hyperclass generic hyperclass ........................................................................................................................119
Table 6-5: Specifications of some typical integrity rules .................................................................119
Table 6-6: The periods for the Hyperclass generic hyperclass ......................................................120
Table 7-1: Impact-analysis of changes of dynamic states of objects of Hyperclass and Attribute generic hyperclasses ...........................................................130
Table 7-2: Coordination rules concerning the coordination of objects of Hyperclass generic hyperclass with objects of Attribute generic hyperclass. .................................132
Table 7-3: Specifications of some integrity rules originated from coordination rules. ..................133
Table 7-4: Periods for validating integrity rules originating from Intra-IS coordination rules. ................133
Table 7-5: Class cross-analysis table. ...........................................................................................139
Table 7-6: Pre- & Post-hyperclass cross-analysis table .................................................................140
Table 7-7: Process cross-analysis table ........................................................................................140
Table 8-1: The validity rules of the Static aspect ...........................................................................164
Table 8-2: The completeness rules of the Static aspect .................................................................164
Table 8-3: The validity rules of the Dynamic aspect ....................................................................166
Table 8-4: The completeness rules of the Dynamic aspect ...........................................................167
Table 8-5: The validity rules of the Integrity rule aspect..................................................168
Table 8-6: The completeness rules of the Integrity rule aspect..........................................169
Table 8-7: Impact-analysis table concerning the changes of dynamic states of
objects of Atomic-class generic hyperclass. .................................................................170
Table 8-8: Coordination rules concerning the Atomic-class generic hyperclass..............170
Table 8-9: Impact-analysis table concerning the changes of dynamic states of
objects of Hyperclass generic hyperclass. .................................................................171
Table 8-10: Coordination rules concerning the Hyperclass generic hyperclass ..........174
Table 8-11: Impact-analysis table concerning the changes of dynamic states of
objects of Attribute generic hyperclass. .....................................................................174
Table 8-12: Coordination rules concerning the Attribute generic hyperclass...........175
Table 8-13: Impact-analysis table concerning the changes of dynamic states of
objects of Key generic hyperclass..............................................................................176
Table 8-14: Coordination rules concerning the Key generic hyperclass..................176
Table 8-15: Impact-analysis table concerning the changes of dynamic states of
objects of Event generic hyperclass. ..........................................................................176
Table 8-16: Coordination rules concerning the Event generic hyperclass.............176
Table 8-17: Impact-analysis table concerning the changes of dynamic states of
objects of Process generic hyperclass. .......................................................................177
Table 8-18: Coordination rules concerning the Process generic hyperclass...........177
Table 8-19: Impact-analysis table concerning the changes of dynamic states of
objects of Dynamic state generic hyperclass. .........................................................178
Table 8-20: Coordination rules concerning the Dynamic state generic
hyperclass....................................................................................................................179
Table 8-21: Impact-analysis table concerning the changes of dynamic states of
objects of Method generic hyperclass. .......................................................................179
Table 8-22: Coordination rules concerning the Method generic hyperclass...........180
Table 8-23: Impact-analysis table concerning the changes of dynamic states of
objects of Integrity rule generic hyperclass. ............................................................181
Table 8-24: Coordination rules concerning the Integrity rule generic hyperclass...181
Table 8-25: Impact-analysis table concerning the changes of dynamic states of
objects of Scope generic hyperclass. ..........................................................................181
Table 8-26: Coordination rules concerning the Scope generic hyperclass...............181
Table 8-27: Impact-analysis table concerning the changes of dynamic states of
objects of Risk generic hyperclass. ...........................................................................182
Table 8-28: Coordination rules concerning the Risk generic hyperclass..............183
Chapter 1.
INTRODUCTION

1.1. Introduction

1.1.1. Context

Nowadays, information systems (IS) have performed increasingly important roles in both business and governmental sectors. For developing information systems, most enterprises have supported by various heterogeneous tools and systems. However, these tools and systems are not designed to coordinate with the others. The matter is that Information technology (IT) vendors are deploying these products at a blistering rate, solving one set of problems, but creating another: the need to coordinate them.

On the other hand, today IS managers, developers and users want to manage and integrated access to all information resources, including both types of resources: data and their descriptions: specifications. These information resources are stored not just in traditional databases, but also in other platforms such as file systems, knowledge-base systems, workflow bases, and document systems.

For this reason, there is an essential need to integrate these information resources to construct a global and unique view of the enterprise. However, the semantic mismatch and the redundancy of information resources at different levels of abstraction make the coordination of these systems and tools become an important challenge for IS community.

1.1.2. Motivation

Our research domain aims at satisfying the growing tendency to require more control and more coordination of systems and tools used in IS development process. These requirements contain numerous potential advantages, but also lead to certain challenges in the capacity to manage effectively such environments.

Indeed, there are various approaches that have been focused on providing more connectivity among systems and tools at the Informatics level (concerning with the
Information System Upon Information Systems

technical environment). Unfortunately, connectivity maybe is not sufficient because systems and tools weren’t designed to interoperate.

Therefore, our research focuses on providing management and coordination of information resources at the Information level, a higher level of abstraction than the Informatics level. According to our standpoint, management and coordination of information resources is better performed at the Information level, which represents the semantic content of information, independent with technologies and choices of implementation [LeDinh 04].

Consequently, the objective of our research is to provide an architecture framework, which aims at supporting management and coordination of information resources, used in IS development, at the Information level.

At this point, our approach shares the common strategy of the approach of Model Driven Architecture (MDA) [Frankel 03], which supports IS development by employing a model centric and generative development process. Both the two approach aim at encouraging IS professionals to invest in the Information level, which is independent with technology choice.

The rest of this chapter continues to clarify our proposition. Section 1.2 discusses about information resources at the Information level, in particular the conceptual specifications. Section 1.3 introduces an architecture, called the Information System Upon Information Systems (ISIS), that aims at supporting management and coordination of conceptual specifications. Section 1.4 continues with the proposed research of this thesis: the conceptual framework for ISIS.

1.2. Information resources at the Information level

1.2.1. Specifications

Our proposition leads to the fundamental issue of “what an information resource at Information level” is. In fact, a current matter of argument within IS development today is “should we design artefacts or experience rather than design information and its interactions?”

We often use the term “Data” for business data, which are used by an enterprise and stored in information systems. We use the term “Specification” for data about business data.

Specifications include physical data (contained in information systems) and knowledge (contained in development tools and IS professionals) that represents
information about business data. Specifications are often used in IS development process, from inside or outside the enterprise.

Indeed, information systems are developed in order to support the activities that occur in business domains. Consequently, conceptual specifications of the concepts from the business domains are bound to play an important role in the deliverables that are consumed or produced in the IS development process. These conceptual specifications can be found in requirement and design documents, the implemented system, as well as the user manuals for using the IS.

For instance, when developing a business information system to, concepts from business domains such as “Sale order”, “Customer”, and “Payment” play a crucial role. During the development process, the requirements and the design of the IS are likely to be expressed in terms of the conceptual specifications of those concepts. The specifications may concern with the structure of information such as the classes “Sale order”, “Payment”, and “Customer”. They may also concern with the behaviour of information such as the processes “Receive demand”, “Pay a sale order”, “Create a new customer”, and so on. Furthermore, those specifications may even be reflected in the user manuals of the information system.

1.2.2. Importance of specifications

In the long history of IS development, specifications have been considered as important information resources that might cover long-term vision of IS. However, once information systems operated, these resources often fell into oblivion. There weren’t so much effort to keep these information resources up-to-date.

In our approach, specifications, especially conceptual specifications, are used to manage the coordination and evolution of information systems.

Various factors lead to the need for specifications such as the follows:

- **Capturing knowledge.** When an enterprise loses an IS professional, it also loses the important knowledge about its business processes and its business information systems. Specifications can be used to capture certain knowledge hold by IS professionals and make it accessible.
- **Coordinating with decision support systems.** Recently, enterprises have built decision support systems to help them make better strategic decisions. Consequently, there is a need for specifications to help manage the data used in those systems and methods for locating them.
- **Satisfying users’ requirements.** Besides, users’ requirements are increased and evolved all the time. Indeed, majority of current information systems cannot satisfy the unfulfilled needs of users. Specifications can overcome this challenge. Specifications can form the semantic layer between the technical systems and the business users. In other words, specifications
translate the technical terminology used in computerized systems into terms a business user can understand and vice versa.

- *Integrating systems and tools.* As a matter of fact, there is a lot of information systems and supporting tools, especially legacy systems, are non-integrated and inflexible. Those systems and tools construct their own islands and don’t communicate easily with the others. When there is a need for a change, it is hardly difficult to anticipate and control the effects of this change to the others. Specifications can help to solve the problems of semantic mismatches and data redundancy between those systems and tools.

### 1.2.3. Conceptual and Technical specifications

Conceptual specifications of the concepts from business domain are not only the important information resources used in IS development. Because the IS will be implemented in several technology platforms. There is a need for additional specifications: the specifications at the implementation level.

For this reason, according to levels of abstraction, we propose two types of specifications: Conceptual and Technical specifications.

**Technical specifications** provide information about information systems at the Informatics level. Those specifications are useful for developers and technical users to develop, maintain and grow information systems over time. They are mostly used in the implementation and deployment phases. For examples, some popular technical specifications are databases, tables, views, procedures, triggers, reports, and forms.

**Conceptual specifications** provide information about information systems at the Information level such as information structure, information behaviour and information coherence. Those specifications are useful for business analysts and business users to control and evolve information systems independently of technology choices. They are widely used in the requirement analysis and design phases. Examples about conceptual specifications are classes, domains, processes, methods, dynamic states and integrity rules.

In our approach, *conceptual specifications*, which represent the semantic content of information at the Information level, are the *key resources* used in IS development process. Those key resources are often used to construct other information resources, which are produced and consumed by development activities.

Most technical specifications are the implementations of conceptual specifications. Therefore, once the coordination of conceptual specifications is settled at the Information level, the coordination of the technical specifications at the Informatics level will be solved accordingly.

It is the reason why, in this thesis, we mainly focus on the conceptual specifications that represent the concepts from business domain.
1.2.4. Information layer

To support management and coordination of conceptual specifications, there is a need for a new architecture for representing, managing and coordinating them. This architecture will participate in the IS infrastructure hierarchy of each enterprise and coexist with other infrastructures for information systems. The proposed architecture forms a new layer: the Information layer.

The objective of the Information layer is to represent conceptual specifications for effective IS development. Consequently, the Information layer should be technology-independent and organization-independent.

We argue that the appropriate management of specifications during IS development will help to construct the IS that fits the needs of the business. The management of specifications includes the deliberate activities such as identifying, representing, coordinating and evolving specifications.

The Information layer can play the intermediate role between the two traditional layers of information systems: the Business layer and the Informatics layer. The Business layer aims at representing the reality world and the Informatics layer aims at representing the artificial world.

Figure 1-1 presents the Information layer, which aims at supporting more coordination within and between the Business and the Informatics layers.

![Diagram of Information layer]

Figure 1-1: The Information layer

On the other hand, in the long history of IS development, there are often the gaps between the reality world and the artificial world. Some classical approaches have built around a simple translation of the reality world into the artificial world (or vice versa) that may lead to the renovation of information systems all the time.

Therefore, there is a new responsibility for the IT staff of an enterprise: to analyse and to take into account this gap in IS development process. In other words, the reality world needs to understand and coordinate with the artificial world and vice versa. For this reason, the intermediate architecture such as the Information layer is very important in order to overcome the gaps between the reality world and the artificial world.
1.3. An Information System upon Information Systems

1.3.1. Definition

Being as a new IS infrastructure, the Information layer must coexist with information systems at the Informatics layer. In other words, conceptual specifications must coexist and coordinate with their implementations (i.e. technical specifications) at the Informatics level.

Moreover, to work with heterogeneous, complex and distributed informatics environment as well as the diversity of business environment, the Information layer itself needs an information system to support its activities. The required information system, located at the Information layer, aims at supporting traditional information systems at the Informatics layer. It is the reason why this information system is called Information System upon Information System (ISIS).

In other words, an ISIS is defined as an information system that supports the management and coordination of conceptual specifications of information systems.

1.3.2. Objective of an ISIS

As a whole, an ISIS may support the management and coordination of information resources used in the development as well as the exploitation of traditional information systems.

Supporting coordination of information resources in development. At first, an ISIS can support management and coordination of conceptual specifications within the development of IS. The ISIS can significantly reduce both the time and cost of analysis and development by documenting conceptual specifications and their implementations such as data structures, data behaviours as well as transformation rules.

Documenting conceptual specifications is extremely important, because without the ISIS, those specifications may reside partially or totally in the collective memory of the IT staff. Because the results of the analysis and development are captured and retained in the ISIS, the enterprise gains the benefits of long-lasting investment returns.

Furthermore, in the cases of large-scale and enterprise-wide information systems, the ISIS can help to coordinate between several development projects, which are building or maintaining information systems in parallel.
The ISIS can also integrate specifications and services from different tools and systems used in development process such as *Computer Aided System Engineering (CASE) tools, Fourth Generation language (4GL) tools, and Database Management Systems (DBMS).*

**Supporting coordination of information resources in exploitation.**

Once information systems are deployed and exploited, the ISIS can help to integrate and coordinate between information systems in exploitation. Besides, the ISIS can help to coordinate information systems with other systems such as *Decision Support Systems (DDS)* and their underlying tools such as *Data Warehouse* and *Reporting tools.*

Moreover, the ISIS can also support the evolution of conceptual specifications. This capability helps to reduce the costs of future releases and to avoid various developmental errors when evolving and growing information systems.

At the time being, our research concerns with the first objective of an ISIS: supporting the management and coordination of information resources in development. For this reason, the rest of this thesis will concentrate on this topic.

### 1.3.3. Challenges to an ISIS

As mentioned above, the main objective of an ISIS is to support the activities of the Information layer that concerns with the management and coordination of specifications.

We consider the management of specifications in the context of IS development includes the following activities:

- Identifying conceptual specifications,
- Working with parts of IS as a set of conceptual specifications,
- Managing conceptual specifications,
- Representing and enforcing conceptual specifications, and
- Coordinating conceptual specifications

With the focus on support information systems in development, our approach will face the following challenges in supporting these activities: *Specification identification, Specification modularization, Specification management, Specification representation, and Specification coordination.*

**Specification identification.**

The first challenge is how to identify conceptual specifications without concerning about the constraints under which an IS has been developing. In fact, despite the diversity of IS development methods, there are some categories of conceptual specifications that are invariant. For instance, the specifications of a class in various object-oriented methods are almost the same. The challenge here is how to identify those categories of conceptual specifications.
**Specification modularization.**
The second challenge is how to modularize the IS development process. In other words, an ISIS must allow to work with a part of an IS, in particular a set of conceptual specifications as a semantic unit. In our approach, an *Information System component* can be used to represent a semantic unit. Each information system component has a particular purpose. Since even the smallest enterprise has multiple purposes, with different areas of concerns, it will potentially have many different information system components.

**Specification management.**
The fourth challenge is how to identify the responsible roles, to organize the development teams, and to assign the responsibility zones for development teams. A *responsibility zone*, which is a group of contributors concerning with a part of business domain, handles an information system component.

**Specification representation.**
The third challenge is how to represent conceptual specifications effectively and explicitly. In reality, an ISIS should represent the formal explicit specifications of IS concepts. Those formal specifications express an abstract model of how people think about objects in the information system context, usually restricted to a particular domain of application.

**Specification coordination.**
The fifth challenge is how to coordinate conceptual specifications within and between responsibility zones. This challenge includes the issues of how to identify the interdependencies between conceptual specifications within and between responsibility zones of information system components, and how to enforce coordination policies to manage those interdependencies.

### 1.4. Thesis outline

#### 1.4.1. Conceptual framework for an ISIS

As mentioned above, our research focuses on supporting management and coordination of conceptual specifications used in development.

The objective of this thesis is to propose a framework for building and managing an ISIS that supports management and coordination of conceptual specifications used in development process. Therefore, this framework is called “*Conceptual framework for Information System upon Information Systems*”. 
At first, the framework itself must be generic and comprehensive. The framework should include the conceptual structure and logical structure. The conceptual structure of the framework is for understanding conceptual specifications. The logical structure is for descriptive representations of conceptual specifications without concerning to the development processes or tools used for producing them.

Secondly, the framework should be helpful for sorting out the complex technology and methodology choices. It will be significant to business management as well as IS management.

The framework, therefore, needs to address the five challenges as mentioned in Section 1.3.3. For this reason, the main objectives of the framework are Specification identification, Specification modularisation, Specification management, Specification representation, and Specification coordination.

Indeed, there are a large amount of industrial standards concerning conceptual specifications. The conceptual framework aims at providing a global view of standards at high level of abstraction. In other words, the framework helps IS professionals to understand and analyse various standards. It also encourage IS professionals to create their own strategy, customizing and evolving the ISIS’ framework, for managing and coordination their information systems based on conceptual specifications.

1.4.2. Thesis structure

To achieve the main objectives of the framework, the remaining of this thesis is proposed as followings:

- Chapter 2 concerns with the State-of-Art, including the related works and our industrial experiences about management and coordination of conceptual specifications in IS development.

- Chapter 3, titled Specification identification, addresses the issue of how to identify different categories of conceptual specifications. This part of the framework aims to define a set of key concepts that constructs a reference model, which can be used to identify different categories of conceptual specifications of traditional information systems.

- Chapter 4, titled Specification modularization, concerns with how to work with a part of an IS, in particular a unique and coherent set of conceptual specifications. This chapter proposes the concept of Information System Component and how to use this concept in IS development.

- Chapter 5, titled Specification management, introduces the principles of an ISIS and how an ISIS manages specifications. This chapter also propose some
extended key concepts concerning the management of conceptual specifications.

- Chapter 6, titled *Specification representation*, deals with the representation of conceptual specifications, including the structure, behaviour and coherence of conceptual specifications.

- Chapter 7, titled *Specification coordination*, concerns with the coordination of specifications within the development of an IS, called *Intra-IS coordination*. Thus, it also discuss about the coordination of specifications between the developments of different information systems, called *Inter-IS coordination*.

- Finally, chapter 8 comes to end with the conclusion, present works and future directions.
We try to realize our research by both experimenting or by reasoning. For this reason, the chapter begins with the industrial experience in information system development and then continues with the State-Of-Art about the management and coordination of information resources.

2.1. Industrial experience

This section discusses about the industrial experience about the management and coordination of information resources in IS development. Firstly, the context of the survey is introduced. Secondly, industrial experience is presented. Finally, lessons learned are also mentioned.

2.1.1. Context

The survey of industrial experience is realized in the scope of the MDIS project, which stands for Manpower Development and promotion of Information System. MDIS project is a cooperation project between University of Geneva - Switzerland and University of Natural Sciences - Ho Chi Minh city (UNS-HCM) - Viet Nam.

In the frame of this project, the Computer Science Center (CSC) of UHS-HCM will play the main role as a centre of competence in information system domain for South Viet Nam in the near future.

To become the competence centre, the CSC has to improve the consulting activities for its numerous industrial partners. One of the most important challenges is how to conduct the IS development and maintenance services for its partners.

The author of this thesis is also the founder of the Information System group (IS group) of the CSC that aims at providing consulting activities in information system domain. This group was created in 1995 and has been grown up quickly. Until 1999, it has about 20 IS professionals and provided services for more than 50 industry partners.
As the consequence, there is a need for the research and the development of an application for CSC, in particular for the IS group, to manage its development process.

2.1.2. Products of the CSC

The principal product of the Information System group is the IAS product that aims at supporting the principal business activities of a business organization. This product is the most successful product of the CSC with about 50 industrial partners and 70 installations in their head office and branches.

The overview architecture of the IAS product is showed as in Figure 2-1. An IAS product may includes several components, called IAS components, such as General Ledger, Cash in hand, Cash in bank, Cash in Advance, Material inventory, Product inventory, Account Receivable, Account Payable, Payroll and Fix-Assets.

![Figure 2-1: Architecture of the IAS product.](image)

The development strategy of the IS group is « object-oriented methodology for design, relational DBMS for implementation ». Consequently, the IAS product is designed based on the M7 method [LeDinh 95] and has been implemented in different relational DBMS platforms.

Each IAS component is belonged to a specific type of accounting source documents and will be responsible by a specified type of role in the accounting department. Besides, each IAS component has its own interface and processes. The structure of an IAS component is represented by a module, which is defined as a mechanism that can make the associative abstraction built over the class structure. The behaviour of an IAS component is represented by a life cycle corresponding to the class representing the main source document of the component.
Figure 2-2 shows the interface of the *Cash-in-hand IAS* component. This component is based on the Cash-in-hand documents such as *Receipts* or *Payments*, whose data stored in the *Cash transaction*, *Account list* and *Partner* classes. *Cash-in-hand accountants* assume all the responsibilities for the content of this component.

In practice, depending on the size of the information system, an IAS product can be developed by one or several development teams in parallel or in sequence. Accordingly, an IAS component is managed by an IS professional or a development team. The information overlap between IAS components is managed basically based on the organizational and accounting rules.

### 2.1.3. Lessons learned

During the period of survey, lasting from October 1995 until December 2000, the IS group was concerned with the learning about the management and coordination of information resources used in the IS development process.

The goals during this period were to answer the following questions:

- How to deal with the diversity of development environments?
- How to use conceptual specifications in IS development efficiently?
- How to work with a part of an IS?
- How to coordinate within and between different parts of an IS?

#### 2.1.3.1. Lesson 1: Dealing with the diversity of development environments

The question of the first lesson was how to deal with the diversity of development environments.
A development environment was defined as a development method and the platform supporting this method. In fact, we did know that homogeneous development environments make the development process much easier to manage and control.

However, a young software house such as the CSC had to deal with the heterogeneous development environments according to the following reasons:

- First, as a matter of fact, the CSC had to deal with the giant IT vendors (the giants). Therefore, it had to work with different platforms of the giants in order to receive the supports and to find new industrial partners, which were also partners of the giants.
- Second, the IS professionals of the CSC (such as business analysts, system analysts, team leaders, developers) were graduated from different sources. Consequently, they had different competences on different methods and platforms. In a short term, it was hard to select a standard environment and to train all the IS professionals to work efficiently with this new environment.
- Third, the industrial partners of the CSC might have their selected platforms. They often asked for the products, which were developed on those platforms.
- Fourth, the IT vendors were deploying their platforms at a blistering rate, so that the products of the CSC had to be upgraded frequently.

Therefore, the CSC had tried to be independent with development environments.

At first, it captured the knowledge about its products and make this knowledge available for its IS professionals. Most of the knowledge was represented in the form of conceptual specifications.

Roughly speaking, the conceptual specifications had been used to represent the products as much as possible at the Information level, independent with the Informatics level. Implementing the products in a platform was simply to map the conceptual specifications into the technical specifications of the platform and then implementing them.

Secondly, the IS group had defined a set of its own mapping rules. In general, an object (representing a conceptual specification) at the Information level was transformed into an object (representing a technical specification) at the Informatics level. Roughly speaking, the one-to-one mapping was preferred.

If there was a case of one-to-many mapping, the object at the Information level was refined into several objects to guarantee the one-to-one mapping.

Thirdly, the standard client-server architecture: “PC clients with DBMS servers” was preferred for the IAS products. Based on this architecture, the IAS products were able to work with different DBMS platforms effectively. Consequently, the IAS products had different versions developed with different platforms provided by the giant IT vendors (such as Microsoft, Sybase, IBM and Oracle).
2.1.3.2. Lesson 2: Using conceptual specifications in IS development?

The second question was how to use conceptual specifications in IS development.

We learned very early that the conceptual specifications played the important role, not only in modelling phase, but also across the development phases.

The first challenge was how to deal with the diversity of information resources produced by development methods. Indeed, we defined the key specifications such as the specification of a class, a process, or an integrity rule. An information resource was a composition of several key specifications. For instance, a data schema was a composition of several specifications of classes.

The second challenge was the semantic mismatch between the specifications. For example, there was the semantic mismatch between the term “Entity Type” and the term “Class” in development methods. However, there wasn’t too much differences between those terms used by IS professionals of the CSC. They did use the different terms for a “common” category of specifications.

Indeed, the project leaders of the CSC had tried to use a common “ontology” based on the M7 method. That common ontology contained the basic concepts such as tuple-class, attribute, atomic-class, module, key, event, process, dynamic state, and integrity rule. Then, they tried to convert to suitable terms when discussing with IS professionals.

Third challenge was how to present the knowledge about working practice. The CSC tried to capture the working practice from their industrial partners and to document them. This working practice was represented in different forms such as interface screens and their interactions, user guide documentation, training courses, and interaction styles. Thanks to those efforts, users of the IAS products could use the products efficiently and they could also migrate to the same product developed on other platforms without difficulty.

Fourth challenge was how the conceptual specifications might co-exist with business data. Indeed, each IAS product had its own dictionary to store all the necessary specifications of business data. The processes of IAS products used the dictionary when processing business data. Consequently, when maintaining IAS products, IS professional might modify the dictionary instead of modifying the source codes of the processes.

2.1.3.3. Lesson 3: Working with a part of an IS

The question was how to work with a part of an IS, or a subset of conceptual specifications.
As mentioned before, an IAS product was composed by several IAS components. Each IAS component included a set of classes, a set of processes operated on those classes, and a set of integrity rules concerned by the processes.

Depending on the implementing platform, IAS components could be implemented together or separately. For instance, IAS components were implemented separately in the file management systems such as MS Foxpro; but they were implemented together in the relational database management systems such as MS SQL server, SyBase, DB2 or Oracle.

An IAS component was generally developed by a development team. Each IAS component might be analysed, designed, tested, implemented, deployed, and exploited independently with other IAS components. The project leader of the IAS product assumed the responsibility to coordinate between these teams.

2.1.3.4. Lesson 4: Coordinating within and between different parts of an IS?

The question of this lesson was how to coordinate specifications within and between different parts of an IS.

The first challenge was how to coordinate specifications within an IAS component. There were often the complaints that a system analyst modified a conceptual specification without informing the developer and vice versa. Or a developer modified the dictionary without warning to the tester. The solution used by the CSC was to appoint the owner of each specification at each phase of the development process. Only the owner could modify the specification and he had the responsibility to inform to the others when there were the modifications, or the others have the facility to monitor the change of the specification. For example, at the implementation phase, only the developer could modify the specification of the “Partner” class. If he decided to modify the size of the attribute: “Partner-name” of this class, he had to inform to the others (tester, system analyst or team leader), who concerned to this class.

The second challenge was how to coordinate specifications between several teams developing IAS components. Depending on the organization structure and accounting system style of the partner, for each specification, the project leader proposed a team, which played the role of the owner. For instance, the Cash-in-hand team was usually the owner of the specification of the Cash-transaction class, and the General-Ledger team was the owner of the specification of the Account-list class. If the Cash-in-hand team intended to modify the specification of the Account-list class, they had to send the request to the General Ledger team, which was the owner of this specification. If the owner’s team approved the request, the requester’s team could modify this specification in its own workspace.
2.1.4. Discussion

The lessons learned are useful not only for the CSC, but also for other software houses that provide IT services for business organizations. They are positive lessons that introduce the heuristic solution of the CSC to overcome its challenges.

However, for the long vision, these questions need to settle in a more generic way. It is the reason why the Information System upon Information Systems and its conceptual framework are proposed in this thesis.

The answer for the first question can be found at Chapter 3, concerning with conceptual specifications. The answer of the second question can be found at Chapter 5 and Chapter 6, which concern with the ISIS and how to represent conceptual specifications. The answer of the third question can be found at Chapter 4, concerning with Information System components. The answer of the fourth question can be found at Chapter 7, concerning with the coordination of conceptual specifications.

2.2. State-Of-Art

In this thesis, we prefer the term of “management” than the term of “handling”. Indeed, during IS development, there often occurs a lot of handling information resources instead of proper management of information resources. We define management of information resources as the thoughtful activities of identifying, capturing, representing, validating, and evolving information resources.

Thus, we also prefer the term of “coordination” than “integration”. Indeed, coordination of information resources is the act of integrating and coordinating them. Therefore, the “coordination” term is used for replacing “integration” term in the remaining of this thesis.

Furthermore, we prefer the term of “information resource” than “artefact”. “Information resources” at the Information level in our approach can be considered as “design artefact” in the RUP approach [Royce 98]. There are other categories of artefacts such as “implementation artefacts”, “deployment artefacts”, “management artefacts”, and “operational artefacts”.

In our approach, it is argued that management and coordination of information resources plays an important role during the entire life cycle of IS development. In order to gain successful management and coordination of all information resources, enterprises need to define the management and coordination strategies that must solve both business and technical problems in a coherent manner. Many projects have been failed because there is the lack of a strategic view, not so much attention in evolution issues, and disregard business processes and information structures.
In the next section, we present the State-Of-Art about the approaches concerning management and coordination of information resources used in IS development.

### 2.2.1. Related work

In the following, we discuss about approaches concerning the management and coordination of information resources used in IS development.

These approaches are classified according to the three categories:

- **Knowledge management approaches** concerns with the knowledge about development projects.
- **Artefact management approaches** concerns with the artefacts produced and consumed by the activities of development projects.
- **Meta data management approaches** concerns with the information about business data stored in information systems.

#### 2.2.1.1. Knowledge management.

Knowledge management approaches concerns with how to identify, represent, validate and access the knowledge about development projects. The typical approaches are the **LaSSIE**, **CODE4**, and **ConceptBase**.

**LaSSIE.**

The LaSSIE approach [Devanbu 91] uses a frame-based knowledge representation language **KANDOR** to store information about a large software project in order to provide semantic retrieval. In the KANDOR language, a frame is considered as a complex description that expresses constraints on members of the class that it denotes. By this way, frames form the taxonomy of classes, with the most general class “THING” at the top, and the most specific classes at the bottom. Besides, the two-place relations, called slots, describe the attributes of class members.

The LaSSIE system provides a knowledge-based explicitly representation of information about a large software project, and an interactive interface to give IS professionals to access directly to the information.

**CODE4.**

The knowledge representation used by the CODE4 approach [Skuce 95] has its roots in ideas borrowed from frame-based, conceptual graphs, object-oriented and description logic systems. The basic unit of knowledge is called the **concept**. A concept represents a “thing”: either a collection of similar thing or a particular thing.

The CODE4 system is a general-purpose knowledge management system that intended to assist with the common knowledge processing needs of anyone who
desires to analyse, store, or retrieve conceptual knowledge in software development, in the construction of term banks, or in the development of ontology for natural language understanding.

**ConceptBase.**
The Conceptbase approach [Jarke 95, Jeusfeld 98] has adopted the Telos language, which represents knowledge about information systems. Indeed, Telos [Mylopoulos 90] is a language intended to support the IS development. The language can be used to represent knowledge about the different worlds related to the IS development process such as the *subject world* (application domain), the *usage world* (users and working environment), the *system world* (informatics environment) and the *development world* (development teams and methodologies). In the Telos language, structured objects, which are built from two kinds of primitive units, construct a knowledge base. There are two types of primitive units: individuals and attributes. *Individual* represents entities, while *attributes* represent binary relationships between entities and other relationships.

ConceptBase is a typical Knowledge Based Management System (KBMS) that aims at supporting the cooperative development and evolution of information systems. Indeed, ConceptBase is the implementation of the Telos language and based on the integration of deductive and object-oriented technologies.

### 2.2.1.2. Meta data management.

Meta data management approaches aim at providing a repository to manage, validate, use and evolve meta data stored in tools and systems. There are two categories: *Data dictionary* and *Meta data repository*.

**Data dictionary.**
Data dictionary provides a centralized repository of information about database to assist database administrators and developers in managing, controlling, and validating the collection and the use of data.

**Meta data repository.**
Meta data repository aims at integrating and managing specifications using standard meta data models and interchange format standards [Marco 00]. There are two competitive organizations for meta data standards: the *Meta Data Coalition* (MDC) with *Open Information Model* [Bernstein 99] and *Meta Data Interchange Specification* [MDIS 97]; and the Object Management Group (OMG) with *Common Warehouse Metamodel* [CWM 03], *Meta Object Facilities* [MOF 02] and *eXtensible Metadata Interchange* [XMI 00].
2.2.1.3. Artefact management.

Artefact management approaches are primarily designed to serve as repositories of all the artefacts produced and consumed during the life cycle of an IS development project. These artefacts can be schema, documentations, programs, reports, and tools that intend to provide support for all activities of IS development, beginning with requirement analysis through to system implementation.

The typical approaches are the DAIDA environment, Software Information base and Software Configuration Management.

DAIDA environment.
DAIDA is an environment, developed in an ESPRIT project, that bases on State-Of-Art languages for requirements modelling, design and implementation of information systems [Jarke 92]. Users of DAIDA environment are offered three languages through which they can elaborate requirements, design and implementation specifications. Besides, this environment also supports programming-in-the-large (objects can be organized in a modular way) and programming-in-the-many (versions can be created cooperatively).

Software Information Base (SIB).
Software Information Base is a software repository system that provides organization, storage, management, and access facilities for reusable software artefacts [Constantopoulos 95]. The SIB supports the representation of information about requirements, designs and implementations of software artefacts. The SIB is structured as a directed attributed graph, with nodes describing software artefacts and edges representing semantic relationships between them. A software artefact is assumed to have its own representation, external to the SIB, for example: a program or a diagram.

Software Configuration Management (SCM) provides facilities for identifying, monitoring and controlling configuration, managing as well as supporting teamwork [Stephen 02].

2.2.2. Discussion

The major differences between the Information System upon Information Systems (ISIS) approach and the other approaches can be derived from the main characteristics of the ISIS approach:

- Representing specifications in a coherent manner, including the structure, behaviour and coherence of specifications;
- Providing the capacity to work with a subset of specifications as parts of an IS;
• Supporting the management and coordination of specifications throughout all phases of the IS development process.

Table 2-1 presents the major differences of the ISIS approach with the other approaches.

**Table 2-1: Comparisons the ISIS approach with the other approaches.**

<table>
<thead>
<tr>
<th>Approaches</th>
<th>Representing structure of specification</th>
<th>Representing behaviour of specification</th>
<th>Representing coherence of specification</th>
<th>Supporting the management of specifications process</th>
<th>Supporting the coordination of specifications</th>
<th>Working with parts of an IS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Concept management approach</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>- LaSSIE</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- CODE4</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>- ConceptBase</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Meta data management approach</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data Dictionary</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meta Data repository systems</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td>Yes (partial)</td>
<td>Yes (partial)</td>
</tr>
<tr>
<td><strong>Artefact management approach</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DAIDA</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td>Yes (partial)</td>
<td>Yes (partial)</td>
</tr>
<tr>
<td>Software Information Base (SIB)</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Software Configuration Management (SCM) tools</td>
<td>Yes</td>
<td></td>
<td></td>
<td>Yes (partial)</td>
<td>Yes (partial)</td>
<td>Yes (partial)</td>
</tr>
<tr>
<td><strong>Information System upon Information Systems (ISIS)</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Firstly, there is a little focus on representing the behaviour of specifications. In the ISIS approach, the behaviour of specifications is represented by the possible dynamic states of specifications and the transitions between those dynamic states.

Secondly, in the ISIS approach, the coherence of specifications is represented by a set of completeness and validity rules. However, the related approaches don’t concern so much about the completeness rules.

Thirdly, the ISIS approach aims at supporting the management of specifications throughout the IS development process, meanwhile some approaches focus only on certain phases of the development process. For instance, the Meta data repository systems aim at the design phase; the Data dictionary approach aims at the implementation and exploitation phases; the Concept management approach aims at supporting the conceptual specifications at the requirement and design phases.

Fourthly, there is not so much concern about supporting the coordination of specifications within a development team and between different teams working in parallel. Moreover, the coordination of specifications in the industrial approaches is not so flexible generally based the check-in check-out mechanism. In the ISIS approach, the coordination is based on the information overlaps and overlap protocols to manage them.

Lastly, there is a little focus on working with parts of an IS. There is only the DAIDA approach that deals with the structured and complex objects to represent artefacts. This approach also provides the facility to represent objects in a modular way.

The ISIS approach aims at supporting to develop information systems based on component of information systems and to manage specifications based on components of the ISIS:

- A component of information system (or Information system component) is an artefact of an IS that includes the static, dynamic and integrity rules aspects of IS. An IS can be constructed by a set of IS components. An IS component can be considered as an autonomous IS and can be developed independently with the other IS components. This makes the ISIS approach conform to the transition to an iterative life-cycle process. Iterations establish precedents from which the IS components, the corresponding development processes, and the development plans can be elaborated in evolving levels of detail.

- A component of the ISIS concerns with a category of specifications. Specifications of a component of the ISIS can be represented by the hyper-objects of this component. A hyper-object is a set of objects built with an object and the other objects, which can be reached from that object by following its navigation links [Turki 02]. Representing specifications using hyper-objects makes the ISIS approach more flexible than other approaches. IS professionals can easily accommodate the approach to their enterprises. They can customize the information about the structure and behaviour of each
category of specifications. They can also modify the information about the interrelationships between those categories of specifications.

In conclusion, the ambition of the ISIS approach is to provide a distributed component-based architecture that supports management and coordination of conceptual specifications used in IS development in depth and in width.

This architecture aims at constructing the Information layer as a new IS infrastructure that manages the structure, behaviour and coherence of conceptual specifications of information system.

Furthermore, to support to work in a modular way, the architecture also includes a set of common and domain specific components of the ISIS, and the coordination policies between the components.
3.1. Introduction

The chapter introduces the principle of the M7 reference model, which can be used to identify different categories of conceptual specifications of information systems. This reference model aims at providing a uniform framework to study the identification, representation, coordination and validation of conceptual specifications.

In general, a reference model is a description of all of the possible software concepts, and the interrelationships between them. Therefore, concerning with IS engineering, the M7 reference model must focuses on the concepts of information system (in short IS concepts) and their interrelationships that which represent different aspects of information system.

In fact, information system development is modelling, specifying and then implementing IS concepts. In general, IS concepts may express:

- **Structure of information** representing by the Static aspect of IS, for example the “Sale order” class.
- **Transformation of information** representing by the Dynamic aspect of IS, for example the “Paying sale order” process.
- **Coherence of information** representing by the Integrity rule aspect of IS, for example the business rule that states: “the total amount of all the payments of a sale order must not exceed the amount of that sale order”.

In other words, an IS concept can be represented by a concept belonged to the Static aspect, the Dynamic aspect, or the Integrity rule aspect of IS.

Despite the diversity of IS development methods and conceptual models, there are some IS concepts that are invariant.

For this reason, the M7 reference model is constructed by a unique and coherent set of the invariant IS concepts and their interrelationships based on the object-oriented paradigm. Its concepts can be found in almost object-oriented methods such as OOA/D [Coad 90, Coad 91], Booch [Booch 91], OMT [Rumbaugh 91], Shlaer / Mellor [Shlaer 89, Shlaer 92],OOSE [Jacobson 92], Rolland [Rolland 93], and UML [Booch 97, Rumbaugh 98].
The M7 reference model is based on the *M7 object-oriented method* [Estier 91, 96; LeDinh 95], which aims at supporting the development, integration and evolution of enterprise wide information systems. This method proposes several models to represent different aspects of an IS, in particular the Static, Dynamic, and Integrity rule aspects.

The organization of the rest of this chapter is as follows. Section 3.2 begins with the IS concepts of the Static aspect. Section 3.3 continues with the IS concepts of the Dynamic aspect. Finally, Section 3.4 presents the IS concepts of the Integrity rule aspect.

### 3.2. The Static aspect

The **Static aspect** concerns with the *structure of information*. The conceptual specifications of the Static aspect describe what type of information exists, their structures as well as their interrelationships.

Consequently, this section begins with the IS concepts of the Static aspect (in short: the *static concepts*) and then continues with their interrelationships.

The static concepts presented in this section will help to identify the categories of conceptual specifications such as *Atomic-class, Hyperclass, Attribute, Key*, and *Sub-hyperclass*.

#### 3.2.1. Static concepts

Persistent memorization in an IS of a concept from business domain and its related phenomena in the living world will be obtained by the presence of one or more structured objects.

An object type and a set of objects of this type define a class. In the M7 reference model, there are three kinds of classes: *Atomic-class, Tuple-class* and *Hyperclass*.

Tuple-classes and hyperclasses are the fundamental elements of conceptual models, to which we can associate objects, on which we can define object life-cycle and integrity rules.

##### 3.2.1.1. Atomic-class

**Atomic-classes** are defined as primitive classes, which are indecomposable. Objects of atomic-classes have a particular characteristic: their logical identifier is also their value.
An atomic class is different from a domain. A domain is just a set of atomic data values; meanwhile an atomic class is a class with objects having their object identifier as well as their value.

Example 3-1: The atomic-classes.
Let’s discuss about a typical information system for a small trade company named MATIS. The IS for MATIS (called MATIS-IS) aims to support its traditional business processes such as Sale/Collection, Purchase/Acquisition, and Conversion processes [Hollander 00].

This example concerns with the Sale/Collection process, in particular the direct sale activities of this company. In general, the information about the business activities can be represented in the MATIS-IS by the atomic-classes such as Code, Name, Date, Address, Telephone, etc.

3.2.1.2. Tuple-class

A tuple-class contains objects having the same structure and the same behaviour. The interpretation of a tuple-class is a set of objects identified by their object identifiers (OID).

A structure of a tuple-class is characterized by a set of attributes. An attribute of a tuple-class is a function that corresponds every object of this tuple-class (called origin tuple class) to a set of objects of a tuple-class or an atomic class (called termination class).

The behaviour of a tuple-class is represented by a set of methods. Attributes and methods will be discussed later in this chapter.

In principle, a tuple-class aims to be persistent as a unit of data storage and manipulation at the Informatics level. For instance, a tuple-class can be implemented as a table in the relational Database Management Systems (DBMS), an object type in object-relational DBMSs, or a class in object-oriented DBMSs.

Example 3-2: The tuple-classes.
Concerning the MATIS-IS, there are the tuple-classes that represent the structure of information related to the Sale/Collection process such as Customer, Inventory item, Sale order, Sale order items and Payment.

3.2.1.3. Hyperclass

In the case of large-scale information systems, there is a need to work with a subset of tuple-classes as a tuple-class. For this reason, we define the hyperclass concept.
A hyperclass is an associative abstraction built over the class structure that associates with a particular semantic context. In the operational point of view, a hyperclass is a subset of tuple classes that all connected by navigation links to a key tuple class (in short: key class) without ambiguity [Andany 91; Turki 02].

An object of a hyperclass, called a hyper-object, is built with an object of the key class and all the objects of the tuple-classes of the hyperclass that can be reached from that object of the key class by following its navigation links.

Consequently, a tuple-class can be considered as a particular case of a hyperclass. For this reason, in the M7 reference model, a Tuple-class concept is considered as a specialisation of a Hyperclass concept.

**Notes:** Henceforth, in the following sections:
- The term “a class” means “a hyperclass or a tuple class”
- The term “an object” means “a hyperobject of a hyperclass or an object of a tuple class”
- The term “a category” means “a hyperclass, a tuple-class, or an atomic-class”

**Example 3-3: The Sale hyperclass.**
Return to our example, in the MATIS-IS, there are several hyperclasses associated with particular semantic contexts, which can be defined based on a sale order, a purchase order, an inventory item, a customer, or a supplier.

Let’s define a Sale hyperclass built over the set of classes related to a sale order. From a specific sale order, we can get information about its customer, inventory items, inventory in stock, and payments using the Sale hyperclass.

### 3.2.2. Interrelationships between static concepts

The interrelationships between the static concepts lead to other IS concepts such as Attribute, Key and Sub-hyperclass.

A hyperclass uses attributes and methods to model a structure of a concept or a fact in the living world. An attribute of a hyperclass is a function corresponds every object of this hyperclass to a set of objects of the termination class.

A hyperclass can use object identifier or logical identifiers. A key of a hyperclass is a logical identifier, which can be used to distinguish one hyper-object from others. A key of a hyperclass is defined by a set of attributes of this hyperclass.

A hyperclass can be declared as a sub-hyperclass of the other hyperclass, which is called super-hyperclass. In the M7 method, the specialisation relationship has a strong consequence: the hyper-objects of a sub-hyperclass are also the hyper-objects of its super-hyperclass.
Each sub-hyperclass has the same attributes as its super-hyperclass and it can possess supplementary attributes for its hyper-objects. Consequently, the hyper-objects of a sub-hyperclass can possess more characteristics than the hyper-objects of its super-hyperclass.

3.2.2.1. Attribute

Attribute of a hyperclass designates the navigation link between this hyperclass as the origin hyperclass to a category as the termination category. A category can be a hyperclass, a tuple-class or an atomic-class.

Example 3-4: The Customer telephone attribute of the Customer tuple-class
The Customer tuple-class has the attributes such as Customer-code, Customer-name, Customer-address, Customer-telephone and ActivatedTime (Table 3-1).

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Origin Hyperclass</th>
<th>Termination Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer code</td>
<td>Customer</td>
<td>Code</td>
</tr>
<tr>
<td>Customer name</td>
<td>Customer</td>
<td>Name</td>
</tr>
<tr>
<td>Customer address</td>
<td>Customer</td>
<td>Address</td>
</tr>
<tr>
<td>Customer telephone</td>
<td>Customer</td>
<td>Telephone</td>
</tr>
<tr>
<td>ActivatedTime</td>
<td>Customer</td>
<td>DateTime</td>
</tr>
</tbody>
</table>

Table 3-1: Attributes of the Customer tuple-class

Let’s continue to specify attributes of a hyperclass. The Sale hyperclass has the attributes such as has-Sale order, has-Items, has-Customer, and has-Payments (Table 3-2).

<table>
<thead>
<tr>
<th>NameAttribute</th>
<th>Origin Hyperclass</th>
<th>Termination Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>has Customer</td>
<td>Sale</td>
<td>Customer</td>
</tr>
<tr>
<td>has SaleOrder</td>
<td>Sale</td>
<td>Sale order</td>
</tr>
<tr>
<td>has SaleOrderItems</td>
<td>Sale</td>
<td>Sale order item</td>
</tr>
<tr>
<td>has Items</td>
<td>Sale</td>
<td>Inventory Item</td>
</tr>
<tr>
<td>has Payments</td>
<td>Sale</td>
<td>Payment</td>
</tr>
</tbody>
</table>

Table 3-2: Attributes of the Sale hyperclass

3.2.2.2. Key

A key of a hyperclass is defined by a set of special attributes can be used to distinguish one hyper-object from other hyper-objects in the same hyperclass. A key can be considered as logical identifier of a hyper-object in its hyperclass.
Example 3-5: The CustomerKey and the SaleKey keys.
The key of the Customer tuple-class is CustomerKey defined by only one attribute: CustomerCode. On the other hand, the key of the Sale hyperclass is SaleKey defined by an attribute: has-SaleOrder.

3.2.2.3. Sub-hyperclass

Thanks to the mechanism of generalisation/specialisation, a hyperclass can define its sub-hyperclasses. The interpretation of a sub-hyperclass is exactly the set of all identifiers of the interpretation of its super-hyperclass for which the specialisation condition evaluates to be “true”.

On the contrary, the super-hyperclass of a sub-hyperclass is the hyperclass from which the sub-hyperclass is directly derived. Ancestors of a sub-hyperclass are its direct super-hyperclass and the super-hyperclasses of that super-hyperclass. Descendants of a hyperclass are the (direct and indirect) sub-hyperclasses of that hyperclass.

In other words, a hyper-object of a sub-hyperclass is also a hyper-object of its super-hyperclass when the specialisation condition is true. The specialisation condition is a Boolean expression based on attributes of the direct super-hyperclass. That type of specialisation is called dynamic specialisation. Dynamic specialisation can be used to define other IS concepts such as sub-hyperclass, or dynamic state of objects (being mentioned later).

By the way of consequences, a sub-hyperclass have all the properties of its ancestors and it also may have its own properties. Besides, each sub-hyperclass has one and only one direct super-hyperclass. In other words, the M7 method does not support multiple inheritances.

Example 3-6: The LastYearSO sub-hyperclass.
The Sale Order (SO) tuple-class has the sub-hyperclass: LastYearSO. The specialisation condition is “Year of the Sale Order Date is equal this year subtracted 1”.

3.2.3. Static diagrams

Conceptual diagrams are graphical presentations of a set of modelling concepts that can be used to visualize an information system (IS) from different perspectives. Actually, the Static aspect of IS can be viewed using one of the following diagrams: Class diagram or Object diagram.
3.2.3.1. **Class diagram**

The **Class diagram** shows a set of hyperclasses, tuple-classes, atomic-classes and their interrelationships. Class diagrams can be used to model the static view of an IS, or a part of an IS such as a hyperclass.

In class diagrams, atomic-classes are represented by ellipses with dot line, and tuple-classes are represented by ellipses with solid line (see Figure 3-1 and Figure 3-2). Depending on the level of granularity, class diagrams may include atomic-classes or not.

**Example 3-7: The Class diagram.**
Firstly, the Figure 3-1 presents a Class diagram for all the hyper-classes, tuple-classes and atomic-classes concerning the Sale/Collection process of the MATIS-IS.

![Diagram](image)

**Figure 3-1:** The class diagram for all the classes of the Sale/Collection process.

The **Customer** tuple-class has the **Customer Code**, **Customer Name**, **Customer Address**, **Customer Telephone** and **Customer Activated Time** attributes.

The **Sale order** tuple-class has the **SO Number**, **SO Date**, **SO Amount**, **SO Received Time**, **SO Cancelled Time**, **SO Accepted Time**, **SO Delivered Time**, **SO Paid Time**, and **SO Item Returned Time** attributes. Furthermore, the **Sale order item** tuple class has the **of Sale order**, **Order Quantity**, **SO Item Amount**, and **SO_Items** attributes.
The Inventory Item tuple class has the Item Code, Item name, Item Description, Item Unit Price, Quantity In Stock, and ItemActivatedTime attributes.

The Payment tuple class has the Payment Code, Payment Date, Payment Amount, Approved Time, and paymentFor attributes.

On the other hand, the Figure 3-2 introduces the Class diagram for a part of the MATIS-IS: the Sale hyperclass (specified in Example 3-3). This hyperclass has the Sale order tuple class as its key class. From this key class, one can navigate to other tuple-classes such as Customer, Payment, Sale order item and Inventory item.

![Figure 3-2: The class diagram for the Sale hyperclass.](image)

### 3.2.3.2. Object diagram

**Object diagram** shows the objects of classes and their relationships at a given moment in time. Object diagrams can be used to represent a snapshot of the objects in an IS or a part of an IS at a given point of time.

**Example 3-8:** The Object diagram.

The following example introduces the Object diagram of the Sale order tuple-class and its related tuple-classes (Figure 3-3).

![Figure 3-3: The Object diagram of the Sale Order tuple-class](image)
At the time being, there are the following objects:

- There are three objects of the Customer tuple-class: C#1, C#2 and C#3;
- The Sale order tuple-class has two objects: SO#1 and SO#2;
- There are two sale order items of SO#1: SO#1-ITM#1 (i.e. item #1 of sale order #1) and SO#1-ITM#2; and also two sale order items of SO#2: SO#2-ITM#2 and SO#2-ITM#3;
- The Inventory Item tuple-class has four objects: ITM#1, ITM#2, ITM#3, and ITM#4;
- The Payment tuple-class has two objects PY#1 and PY#2.

Concerning the Sale hyperclass, it is supposed that there are two hyper-objects hoS#1 (hyper-object having the key class: SO#1) and hoS#2 (hyper-object having the key class: SO#2). The interpretation of each hyper-object is a set of objects that can be reached from the object of the key class and a set of navigation links used to navigate between these objects (see Figure 3-4).

![Diagram of the HoS#1 hyper-object of the Sale hyperclass.](image)

Figure 3-4: The HoS#1 hyper-object of the Sale hyperclass.

The interpretations of these two hyper-objects: hoS#1 and hoS#2 are specified in Table 3-3. Each interpretation includes a set of objects and a set of links.

<table>
<thead>
<tr>
<th>Name</th>
<th>Objects</th>
<th>Links</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyper-object hoS#1</td>
<td>{SO#1}, {SO#1-ITM#1, SO#1-ITM#2}, {ITM#1, ITM#2}</td>
<td>{SO#1 → C#1}, {SO#1 → SO#1-ITM#1, SO#1 → SO#1-ITM#2}, {SO#1-ITM#1 → ITM#1, SO#1-ITM#2 → ITM#2}</td>
</tr>
<tr>
<td>Hyper-object hoS#2</td>
<td>{SO#2}, {SO#2-ITM#2, SO#2-ITM#3}, {ITM#2, ITM#3}, {PY#1, PY#2}</td>
<td>{SO#2 → C#2}, {SO#2 → SO#2-ITM#2, SO#2 → SO#2-ITM#3}, {SO#2-ITM#2 → ITM#2, SO#2-ITM#3 → ITM#3}, {SO#2 → PY#1, SO#2 → PY#2}</td>
</tr>
</tbody>
</table>

Table 3-3: The interpretations of the hoS#1 and hoS#2 hyper-objects.
For instance, the interpretation of the hoS#1 hyper-object includes a set of objects such as:

- The $SO\#1$ object of the $Sale\ order$ tuple class;
- The $C\#1$ object of the $Customer$ tuple class;
- The $SO\#1-ITM\#1, SO\#1-ITM\#2$ objects of the $Sale\ order\ item$ tuple class;
- The $ITM\#1, ITM\#2$ objects of the $Inventory\ item$ tuple class; and
- The $SO\#1-ITM\#2$ objects of the $Sale\ order\ item$ tuple class.

From the $SO\#1$ object of the $Sale\ order$ key class, we can use the link $\{SO\#1 \rightarrow C\#1\}$ to reach the $C\#1$ object of the $Customer$ tuple class. It is also possible to reach the $SO\#1-ITM\#1, SO\#1-ITM\#2$ objects of the $Sale\ order\ item$ tuple class by using the $\{SO\#1 \rightarrow SO\#1-ITM\#1, SO\#1 \rightarrow SO\#1-ITM\#2\}$ links. Continue with other links, we can reach all the objects of the hoS#1 hyper-object.

### 3.3. The Dynamic aspect

The **Dynamic aspect** concerns with the *transformation of information*, which represents the behaviour of the IS.

In the M7 method, there are two levels of behaviour:

- *Object behaviour* defined as the behaviour of objects of a class; and
- *System behaviour* defined as the behaviour of the IS or as a part of the IS.

This section begins with the object behaviour represented by the *Dynamic State* and Method concepts. Thus, the system behaviour represented by the Event and Process concepts is also discussed.

Accordingly, the Dynamic aspect of an IS can be represented by the categories of conceptual specifications such as *Dynamic state, Method, Event, and Process*.

#### 3.3.1. Dynamic concepts

The IS concepts of the Dynamic aspects (in short: *dynamic concepts*) include *Dynamic state, Method, Event, and Process* concepts.

The interaction between the IS and its organizational environment are performed by business events. When an event happens, its accompanied process is triggered. A process performs continuously methods of hyperclasses. A method is used to transit between the dynamic states of objects of hyperclasses.
3.3.1.1. Dynamic state

Dynamic states of an object are modes or situations during which certain methods are “enabled” and other methods are “disabled”.

At given point of time, concerning the dynamic state $s_i$ of the class $c_1$, objects of that class can be partitioned into two groups: first group containing objects that are at the $s_i$ dynamic state, and the second group that are not at the $s_i$ dynamic state (Figure 3-5).

![Figure 3-5: A dynamic state as a sub-hyperclass.](image)

For this reason, a dynamic state of a class is corresponding to a set of objects of this class that are at this dynamic state. In the M7 method, thanks to the dynamic specialisation concept, a dynamic state can be represented by a specialisation (i.e. a sub-hyperclass) of its hyperclass. The specialisation condition is the condition expressing “an object is at this dynamic state or not”.

**Example 3-9: The dynamic states.**

Dynamic states of the Sale order class will represent its behaviour corresponding to stages of the Sale/collection process. They are SO demand_received, SO demand_valid, SO demand_invalid, SO customer_valid, SO items_delivered, SO amount_paid, and SO items_returned. In the M7 method, these dynamic states can be presented by the sub-hyperclasses of the Sale order hyperclass as in Figure 3-6.

Similarly, the Customer class has dynamic states: Customer activated and Customer deactivated. The Payment class has a dynamic state: Payment approved. Finally, the Inventory item class has the following dynamic states: Inventory Item activated, Inventory item in stock, Inventory item out of stock, and Inventory Item deactivated.
Figure 3-6: Dynamic states of the tuple-classes related to Sale/Collection process.

The summary of dynamic states of the Sale order, Customer, Payment and Inventory item classes are described in the Table 3-4.

<table>
<thead>
<tr>
<th>Dynamic state</th>
<th>Is A Sub-hyperclass</th>
<th>Specialisation condition</th>
<th>Sub-hyperclass Of</th>
</tr>
</thead>
<tbody>
<tr>
<td>Demand (Sale order) is received</td>
<td>SO demand_received</td>
<td>SO ReceivedTime is not null</td>
<td>Sale order</td>
</tr>
<tr>
<td>All the items of a sale order are valid after validating</td>
<td>SO demand_valid</td>
<td>SO AcceptedTime is not null</td>
<td>SO demand-received</td>
</tr>
<tr>
<td>Some items of a sale order are invalid after validating</td>
<td>SO demand_invalid</td>
<td>SO CancelledTime is not null</td>
<td>SO demand-received</td>
</tr>
<tr>
<td>The demand is belonged to a customer activated</td>
<td>SO customer_valid</td>
<td>SO BelongToCustomer is not null</td>
<td>SO demand-valid</td>
</tr>
<tr>
<td>All the items of a Sale order are delivered</td>
<td>SO items_delivered</td>
<td>SO DeliveredTime is not null</td>
<td>SO customer-valid</td>
</tr>
<tr>
<td>Sale order is paid</td>
<td>SO amount_paid</td>
<td>SO PaidTime is not null</td>
<td>SO items-delivered</td>
</tr>
<tr>
<td>Items of a sale order are returned by customer</td>
<td>SO items_returned</td>
<td>SO ItemReturnTime is not null</td>
<td>SO amount-paid</td>
</tr>
<tr>
<td>A customer is activated (after the first sale)</td>
<td>Customer_activated</td>
<td>CustomerActivatedTime is not null</td>
<td>Customer</td>
</tr>
</tbody>
</table>
Information System Upon Information Systems

<table>
<thead>
<tr>
<th>A customer is deactivated</th>
<th>Customer_deactivated</th>
<th>Customer:ActivatedTime is null</th>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>A payment is approved</td>
<td>Payment_approved</td>
<td>PaymentApprovedTime is not null</td>
<td>Payment</td>
</tr>
<tr>
<td>An inventory item is activated</td>
<td>Inventory item activated</td>
<td>Item:ActivatedTime not null</td>
<td>Inventory item</td>
</tr>
<tr>
<td>An inventory item is deactivated</td>
<td>Inventory item deactivated</td>
<td>Item:ActivatedTime is null</td>
<td>Inventory item</td>
</tr>
<tr>
<td>An inventory item is in stock</td>
<td>Inventory item in stock</td>
<td>QuantityInStock is greater than zero</td>
<td>Inventory item activated</td>
</tr>
<tr>
<td>An inventory item is out of stock</td>
<td>Inventory item out of stock</td>
<td>QuantityInStock is equal zero</td>
<td>Inventory item activated</td>
</tr>
</tbody>
</table>

Table 3-4: The dynamic states of the tuple-classes related to Sale/Collection process.

3.3.1.2. Method

A method of a class is used to transit between the dynamic states of the objects of this class. In a clear manner, a method transfers a set of dynamic states to another set of dynamic states of a class.

A method may involve several attributes of the class to which it is belonged.

Example 3-10: Methods of the tuple-classes related to Sale/Collection process.

The Table 3-5 illustrates the methods of the Sale order, Customer, Payment and Inventory item hyperclasses that concerns with the Sale/Collective activities.

<table>
<thead>
<tr>
<th>Method</th>
<th>Name Method</th>
<th>Method Of Hyperclass</th>
<th>From State</th>
<th>To State</th>
<th>Involve Attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Receiving the demands (i.e. information about items) from customer</td>
<td>Receiving demand</td>
<td>Sale order</td>
<td>Sale order</td>
<td>SO demand received</td>
<td>SO Number, SO Date, of- Sale order, SO_Items, Quantity ordered, SO ReceivedTime</td>
</tr>
<tr>
<td>Checking the validity of demanded items; and the quantity of each item is less then the quantity in stock?</td>
<td>Checking stock</td>
<td>Sale order</td>
<td>SO demand received</td>
<td>SO demand valid, SO demand invalid</td>
<td>SO AcceptedTime, SO CancelledTime</td>
</tr>
<tr>
<td>Getting customer information from customer and Checking this customer is</td>
<td>Getting customer info</td>
<td>Sale order</td>
<td>SO demand valid</td>
<td>SO customer valid</td>
<td>Belonged to customer</td>
</tr>
</tbody>
</table>
### Table 3-5: Methods of Sale order, Customer, Inventory item and Payment tuple-classes.

<table>
<thead>
<tr>
<th>already existed?</th>
<th>Decreasing quantity in stock</th>
<th>Sale order</th>
<th>SO customer valid</th>
<th>SO items delivered</th>
<th>SO Delivered time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delivering demanded items to customer</td>
<td>Decreasing quantity in stock</td>
<td>Sale order</td>
<td>SO_items delivered</td>
<td>SO_amount paid</td>
<td>SO Item returned time</td>
</tr>
<tr>
<td>Receiving information about the payment</td>
<td>Receiving payment</td>
<td>Sale order</td>
<td>SO_amount paid</td>
<td>SO_item_returned</td>
<td></td>
</tr>
<tr>
<td>Receiving returned items</td>
<td>Receiving returned items</td>
<td>Sale order</td>
<td>SO_amount paid</td>
<td>SO_item_returned</td>
<td></td>
</tr>
<tr>
<td>Creating a new customer with the information about code, name and address</td>
<td>Creating new customer</td>
<td>Customer</td>
<td>Customer</td>
<td>Customer activated</td>
<td>Customer code, Customer name, Customer address, Customer telephone</td>
</tr>
<tr>
<td>Decreasing the quantity in stock of each item delivered</td>
<td>Decreasing quantity in stock</td>
<td>Inventory item</td>
<td>Inventory item in stock</td>
<td>Inventory item activated</td>
<td>QuantityInStock</td>
</tr>
<tr>
<td>Undo decreasing the quantity in stock of each item returned</td>
<td>Undo decreasing quantity in stock</td>
<td>Inventory item</td>
<td>Inventory item activated</td>
<td>Inventory item in stock</td>
<td>QuantityInStock</td>
</tr>
<tr>
<td>Approving a payment paid by customer</td>
<td>Approving-payment</td>
<td>Payment</td>
<td>Payment approved</td>
<td></td>
<td>Payment code, Payment date, Payment amount, Approved Time</td>
</tr>
</tbody>
</table>

#### 3.3.1.3. Event

**Event** is remarkable phenomena outside of the information system that may provoke a change of dynamic states inside it. In fact, the event structure helps to define the interface of the IS with its organizational and technical environments.

Correspondingly, the interaction of an IS with its environments can be represented by a set of events and their interrelationships. On the operational point of view, events represent the usage rules of the IS environment.

According to our opinion, the “event” concept is much more generic than the “use case” [Muller 97] concept in other methods. The “event” concept deals with the cooperation between the IS with working organisation aspects at the Informational level. Therefore, it is situated at level of informational responsibility zones. The “use case” is similar to a simple representation of the “event” concept at operational level [Léonard 00].
In the M7 method, the relationships between events, processes, dynamic states of objects, zones of responsibilities, as well as actors, are explicitly described. These relationships and the cooperation between the IS with the working organisation will be discussed later in the following chapters.

Example 3-11: The events related to the Sale/Collection process.

Continuing with our example about the MATIS enterprise, it is supposed that there are four business events related to the Sale/Collection process:

- The e1 event is occurred when a customer comes and asks for items;
- The e2 event is occurred when the customer accepts the unit prices informed by the seller, provides personal information and receives items;
- The e3 event is occurred when the customer pays the amount of the Sale order by cash or by credit card;
- The e4 event is occurred when the customer returns items because they do not meet the quality standard or the specification of the customer.

The e2 event is occurred after the e1 event and before the e3 event. Synchronisation and simultaneity between events are out of the scope of this report. More details about synchronisation and simultaneity between events can be found in [Estier 96].

The summary of the four events is presented in Table 3-6.

<table>
<thead>
<tr>
<th>Event</th>
<th>Description</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>e1</td>
<td>When the customer comes and asks for items</td>
<td>Demand-for-Items</td>
</tr>
<tr>
<td>e2</td>
<td>When the customer accepts the unit, provides personal information and receives items</td>
<td>Accept-unit-prices-and-Give-personal-information</td>
</tr>
<tr>
<td>e3</td>
<td>When the customer pays the amount of the Sale order (by cash or by credit card)</td>
<td>Pay-the-sale-order</td>
</tr>
<tr>
<td>e4</td>
<td>When the customer returns the items because their quality is not good</td>
<td>Return-items</td>
</tr>
</tbody>
</table>

Table 3-6: The events related to Sale/Collection process.

3.3.1.4. Process

A process is a feedback of the IS to the occurrence of an event. In fact, a process performs a transformation of a set of dynamic states of the IS.

Actually, to each event, designers will associate a process, which is going to take into account all the decisions of the responsibilities, and especially to take care of a large number of objects, which are concerned by that event, because they are in compatible states.

A process is attached to a hyperclass that must contain all the hyperclasses on which it operates. Every process is specified by a group of pre- and post-hyperclasses as well as by a set of methods to be performed.
The pre-hyperclasses act as a filter: a process can access only to objects of the hyperclasses, which appear in the list of pre-hyperclasses. Similarly, the post-hyperclasses indicate the list of hyperclasses whose objects are associated (or dissociated) by the process.

Furthermore, it is also important to specify the pre-condition and post-condition of each process. A pre-condition states the constraint that must be true before performing a process. On the contrary, a post-condition states the constraint that must be true after performing a process.

Example 3-12: The processes of the Sale/Collection activities.

Concerning the Sale/Collection activities, it is supposed that there are processes:

- The p1 process is to receive and validate a demand of customer;
- The p2 process is to get customer information and to deliver the demanded items;
- The p3 process is to decrease the quantity in stock of each demanded items;
- The p4 process is to receive the payment from customer; and
- The p5 process is to validate and then receive returned items from customer. It also undo decreases (increases) the quantity in stock of each returned items.

Concerning the relationships between these processes and the events as mentioned above, the “p1: Receiving and validating demand” process corresponds to the “e1: Demand for items” event; the “p2: Getting customer information and delivering” process corresponds to the “e2: Accept unit prices and provide customer information” event; the “p4: Receiving payment” process corresponding to the “e3: Pay the Sale order” event; and the “p5: Receiving returned items” process corresponding to the “e4: Return items” event.

The summary of the processes of the Sale/Collection activities is presented in Table 3-7.
<table>
<thead>
<tr>
<th>Process Description</th>
<th>Source Information</th>
<th>Status Information</th>
<th>Event Information</th>
<th>Action Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3. Decreasing quantity in stock</td>
<td>Inventory items in stock</td>
<td>Inventory items activated with QuantityInStock decreased</td>
<td>SO customer valid, Inventory item in stock, Sale order item</td>
<td>Decreasing quantity in stock</td>
</tr>
<tr>
<td>P4. Receiving payment</td>
<td>A sale order without payment info</td>
<td>A sale order with valid payment info</td>
<td>Pay-the sale-order</td>
<td>Receiving payment, Approving payment</td>
</tr>
<tr>
<td>P5. Receiving returned items</td>
<td>A sale order without item returned info</td>
<td>A sale order with item returned info</td>
<td>Return-items</td>
<td>Receiving return items, Undo decreasing quantity in stock</td>
</tr>
</tbody>
</table>

Table 3-7: Processes of the Sale hyperclass.

3.3.2. Dynamic diagrams

The main purpose of dynamic diagrams is to represent the system behaviour of the IS or the behaviour of an object of a class. Therefore, the Dynamic aspect of an IS can be represented by the following diagrams: System life-cycle (SLC) diagram and Object life-cycle (OLC) diagram.

3.3.2.1. System life-cycle diagram

System life-cycle diagram aims to represent the behaviour of an IS or a part of an IS. This diagram shows a set of events, their corresponding processes, and the corresponding pre-hyperclasses and post-hyperclasses of processes.

In the M7 method, it is possible to specify the Dynamic aspect of an IS by using the Node-Star diagram [Léonard 03]. Nodes are associated to (pre- and post-) hyperclasses and stars are associated to processes.

Figure 3-7 illustrates the dynamic diagram to present the System life cycle of the Sale/Collection activities. There are four events that may happen: the e1 vent “When a customer demands for items”; the e2 vent: “When a customer gives personal information and receive the items”; the e3 vent: “When a customer pays for the sale order”; and the e4 vent: “When a customer returns items”.

40
When an event happens, its corresponding process will be triggered. Consequently, there are four processes, which are triggered by events, such as:

- The “p1. Receiving-and-Validating-demand” process triggered by the “e1. Demand-for-items” event;
- The “p2. Getting-Customer-Info” process triggered by the “e2. Accept-Unit-prices-and-Give-personal-info” event;
- The “p4. Receiving-payment” process triggered by the “e3. Pay-the-sale-order” event; and
- The “p5. Receiving-Returned-Items” process triggered by the “e4. Return-items” event.

Besides, there is the “p3. Decreasing-quantity-in-stock” process, which isn’t triggered directly by any event.

Each process accesses objects of pre-hyperclasses as process input, and associates (dissociates) objects of post-hyperclasses as process output. For instance, the “p1. Receiving-and-Validating-demand” process accesses the objects of the Sale-order, Sale-order-item and Inventory-item tuple-classes (pre-hyperclasses). Then it associated the objects of the Sale-order-demand-valid, Sale-order-demand-invalid, Sale-order-item and Inventory-item-in-stock tuple-classes.
It means that from an object of Sale order (representing a demand) as input, there are two possibilities:

- The demand is invalid because their demanded items are not found or are out of stock. In this case, an object of Sale-order-invalid tuple-class is created; or
- The demand is valid. In case the demand is valid, an object of Sale-order-valid tuple-class is created. This object is associated with the objects of Sale-order-item tuple-class and those objects are associated with corresponding objects of the Inventory-item-in-stock tuple-class.

3.3.2.2. Object life-cycle diagram

The Object life-cycle diagram aims at representing the behaviour of objects of a hyperclass or a tuple class. It shows a set of dynamic states (in form of sub-hyperclasses) of a hyperclass or a tuple-class, and a set of methods used to transit between these dynamic states.

Once again, the Node-Star diagram can be used to present the object life cycle of a hyperclass. In this case, nodes are associated to sub-hyperclasses representing dynamic-states and stars are associated to methods (Figure 3-8).

Please note that it is possible that a dynamic state, which is appeared in an object life cycle diagram but not appeared in the system life cycle diagram. These dynamic states will be the “internal” dynamic states that are “invisible” at the system level.

For example, the Figure 3-8 introduces the dynamic diagram representing the Object life cycle of the Sale order tuple class. There are the methods such as Receiving-demand(), Checking-stock(), Getting-Customer-Info(), Decreasing-Quantity-in-stock(), Receiving-payment(), and Receiving-Returned-Items().

![Figure 3-8: Object life cycle.](image-url)
In particular, the *SO_demand_received* dynamic state is an “internal” dynamic state. It is existed in the Object life-cycle but not existed in the System life-cycle.

### 3.4. The Integrity rule aspect

The **Integrity rule aspect** concerns with the *coherence of information*, including the IS concepts such as *Integrity Rule* and *Primitive* as well as their interrelationships with other concepts such as *Scope* and *Risk* [LePham 90; Luu 02; Léonard 03].

The IS concepts of the Integrity rule aspect can be represented using the corresponding categories of conceptual specifications such as *Integrity rule*, *Scope*, and *Risk*.

#### 3.4.1. Integrity rule concepts

The IS concepts of the Integrity Rule aspect (in short: integrity rule concepts) include *Integrity rule* and *Primitive* concepts:

- An *integrity rule* (IR) is a logical condition defined over tuple-classes.
- *Primitives* are basic operations on tuple-classes (such as create, delete and update) whose executions may lead to any risk situations.

#### 3.4.1.1. Integrity rule

**Integrity rules** (IR) of an IS often represent the business rules of an enterprise. An IR actually is a logical condition defined over tuple-classes that can be specified formally and verified by processes or methods.

**Example 3-13: Some typical integrity rules.**

In order to guarantee the coherence of the IS for MATIS enterprise, we may need a large number of integrity rules. Hereafter are some typical integrity rules (Table 3-8):

<table>
<thead>
<tr>
<th>Name Rule</th>
<th>Expression Rule</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR#1</td>
<td>The sale order items must be the items that currently sold by the store.</td>
</tr>
<tr>
<td>IR#2</td>
<td>The amount of each sale order item is equal to unit price * quantity.</td>
</tr>
<tr>
<td>IR#3</td>
<td>The total amount of the sale order is the total of its entire sale order item amount.</td>
</tr>
<tr>
<td>IR#4</td>
<td>The total amount of the payments of a sale order must not exceed the total amount of that sale order.</td>
</tr>
<tr>
<td>IR#5</td>
<td>A sale order must have valid customer info before delivering items.</td>
</tr>
</tbody>
</table>

*Table 3-8: Some typical integrity rules.*
3.4.1.2. **Primitive**

A **primitive** is a basic operation on a tuple-class such as such *create, update* and *delete*. The execution of a primitive may violate the validation of an IR.

3.4.2. **Interrelationships between the Integrity rule concepts**

There are two concepts representing the interrelationships between the integrity rule concepts:

- The *scopes* of an IR denote a set of tuple-classes to which the IR has been defined;
- A *risk* is defined on a scope and a primitive to represent a possibility of suffering the incoherence of information.

3.4.2.1. **Scope**

Scopes of an IR represent the context of an IR including a set of tuple-classes on which the IR has been defined.

*Example 3-14: The scopes.*

The scopes of the IRs mentioned above are as the followings:

<table>
<thead>
<tr>
<th>Scope</th>
<th>Scope of rule</th>
<th>On tuple-class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>IR#1</td>
<td>Sale order item</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>Inventory item</td>
</tr>
<tr>
<td>3</td>
<td>IR#2</td>
<td>Sale order item</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>Inventory item</td>
</tr>
<tr>
<td>5</td>
<td>IR#3</td>
<td>Sale order</td>
</tr>
<tr>
<td>6</td>
<td></td>
<td>Sale order item</td>
</tr>
<tr>
<td>7</td>
<td>IR#4</td>
<td>Sale order</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>Payment</td>
</tr>
<tr>
<td>9</td>
<td>IR#5</td>
<td>Sale order</td>
</tr>
</tbody>
</table>

*Table 3-9: List of scopes.*

3.4.2.2. **Risk**

*Risks* are the possibilities of suffering the incoherence of information. In principal, a risk is defined on a scope and a primitive.

In particular, especially in the case of the Update primitive, it is indispensable to specify the related attribute of a risk.
Besides, a method performing several basic operations (i.e. primitives) may lead into certain risks. Therefore, a method may concern to a set of risks that it may lead into.

**Example 3-15: The risks.**

Hereafter is the specification of the risks related to our integrity rules (Table 3-10).

<table>
<thead>
<tr>
<th>Risk</th>
<th>Risk of scope</th>
<th>Risks</th>
<th>Defined On primitive</th>
<th>Risk on attribute</th>
<th>Concerned Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>01 IR#1</td>
<td>Sale order item</td>
<td>- Creating a new sale order item</td>
<td>Create</td>
<td></td>
<td>ReceivingDemand</td>
</tr>
<tr>
<td>02</td>
<td>Inventory item</td>
<td>- Deleting an existing inventory item</td>
<td>Delete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>04 IR#2</td>
<td>Sale order item</td>
<td>- Creating a new sale order item</td>
<td>Create</td>
<td></td>
<td>ReceivingDemand</td>
</tr>
<tr>
<td>05</td>
<td>Sale order item</td>
<td>- Updating the SO_Item.Amount attribute of the Sale order item tuple-class</td>
<td>Update</td>
<td>SO Item Amount</td>
<td>Checking Stock</td>
</tr>
<tr>
<td>06</td>
<td>Sale order item</td>
<td>- Updating the Quantity Ordered attribute of the Sale order item tuple-class</td>
<td>Update</td>
<td>Quantity Ordered</td>
<td>ReceivingDemand</td>
</tr>
<tr>
<td>07</td>
<td>Inventory item</td>
<td>- Updating the ItemUnitPrice attribute of the Inventory item tuple-class</td>
<td>Update</td>
<td>Item Unit Price</td>
<td></td>
</tr>
<tr>
<td>08 IR#3</td>
<td>Sale order</td>
<td>- Creating a new sale order</td>
<td>Create</td>
<td></td>
<td>ReceivingDemand</td>
</tr>
<tr>
<td>09</td>
<td>Sale order</td>
<td>- Updating the SaleOrder.Amount attribute</td>
<td>Update</td>
<td>Sale Order Amount</td>
<td>Checking stock</td>
</tr>
<tr>
<td>10</td>
<td>Sale order item</td>
<td>- Creating a new sale order item</td>
<td>Create</td>
<td></td>
<td>ReceivingDemand</td>
</tr>
<tr>
<td>11</td>
<td>Sale order item</td>
<td>- Updating the SO_Item.Amount attribute of the Sale order item tuple-class</td>
<td>Update</td>
<td>SO Item Amount</td>
<td>Checking stock</td>
</tr>
<tr>
<td>12 IR#4</td>
<td>Sale order</td>
<td>- Updating the SaleOrder.Amount of the Sale order tuple-class</td>
<td>Update</td>
<td>Sale Order Amount</td>
<td>Checking stock</td>
</tr>
<tr>
<td>13</td>
<td>Payment</td>
<td>- Creating a new payment</td>
<td>Create</td>
<td></td>
<td>Approving payment</td>
</tr>
</tbody>
</table>
### Table 3-10: List of risks.

**Notes:** It is supposed that the identifiers of a tuple-class cannot be modified.

### 3.5. BNF representation

The M7 method is developed for research purpose and therefore has been continuously evolving.

Hereafter is the formal specification at the time being, based on the Backus Naur Form (BNF), for specifying conceptual specifications.

#### 3.5.1. BNF representation for the Static aspect

```plaintext
-- IS specification
IS-spec := IS-aspect-spec {, IS-aspect-spec}
IS-aspect-spec := static-aspect-spec | dynamic-aspect-spec |
                  integrity-rule-aspect-spec

-- Static aspect specification
static-aspect-spec := category-spec {, category-spec}
category-spec := atomic-class-spec | hyperclass-spec

-- Atomic-class specification
atomic-class-spec := ATOMIC-CLASS atomic-class-name
                    SIZE [basic-type : length [, decimal]]
basic-type := Integer|Real|Character|Date|Boolean

-- Hyperclass specification
hyperclass-spec := HYPERCLASS hyperclass-name
                 [IS TUPLE-CLASS]
                 [IS SUB-HYPERCLASS OF hyperclass-name
                  [WHEN specialisation-condition]]
                 [attribute-spec {, attribute-spec}]
                 [key-spec {, key-spec}]

-- Attribute specification
```
attribute-spec := ATTRIBUTE attribute-name
    TERMINATION category-name
    ([[mincard],[maxcard]])

-- Key specification
key-spec := KEY key-name
    [ATTRIBUTES attribute-name {, attribute-name}]

### 3.5.2. BNF representation for the Dynamic aspect

-- Dynamic aspect specification
Dynamic-aspect-spec := Dynamic - spec {, Dynamic - spec }
Dynamic-spec := event-spec| process-spec |
    dynamic-state-spec| method-spec

-- Dynamic state specification
dynamic-state-spec := DYNAMIC STATE hyperclass-id

-- Method specification
method-spec := METHOD method-id
    OF HYPERCLASS hyperclass-id
    [FROM STATES hyperclass-id {, hyperclass-id}]
    [TO STATES hyperclass-id {, hyperclass-id}]
    [INVOLVED ATTRIBUTES attribute-id {,attribute-id}]

-- Event specification
event-spec := EVENT event-id

-- Process specification
process-spec := PROCESS process-id
    IN HYPERCLASS hyperclass-id
    [PRE-CONDITION condition]
    [POST-CONDITION condition]
    [PRE-HYPERCLASS hyperclass-id {, hyperclass-id }]
    [POST-HYPERCLASS hyperclass-id {, hyperclass-id}]
    [TRIGGERED BY event-id]
    EXECUTE method-id {, method-id}

### 3.5.3. BNF representation for the Integrity rule aspect

-- Integrity rule aspect specification
integrity-rule-aspect-spec := integrity-rule-spec
    {, integrity-rule-spec}

-- Integrity rule specification
integrity-rule-spec := INTEGRITY RULE rule-id
    EXPRESSION rule-expression
    [scope-spec {, scope-spec}]

-- Scope specification
scope-spec := ON TUPLE-CLASS hyperclass-id
    [risk-spec {, risk-spec} ]

-- Risk specification
risk-spec := RISK risk-id
    [DEFINED ON primitive-type
    [ON ATTRIBUTE Attribute-id]]
    [CONCERNED BY method-id {, method-id}]
primitive-type := CREATE | UPDATE | DELETE
3.5.4. An example about conceptual specifications of an IS

The complete conceptual specifications of an IS, including the Static, Dynamic, and Integrity rule aspects of the MATIS-IS example, will be illustrated in Annex 1.

3.6. Conclusion

This chapter presents the principle of the M7 reference model that may use as the framework for identifying and representing different categories of traditional conceptual specifications. The purpose of the reference model is to provide a uniform framework to study the identification, representation, coordination and validation of conceptual specification of information systems.

The M7 reference model includes a set of IS concepts belonged to the traditional aspects of information system such as the Static, Dynamic and Integrity rule aspects. In general, those concepts are invariant and shared by the majority of the object-oriented conceptual models and methods.

Those IS concepts can be represented by the corresponding categories of conceptual specification such as:

- Atomic class, Hyperclass, Sub-Hyperclass, Attribute and Key specifications concerning the Static aspect;
- Event, Process, Method, and Dynamic State specifications concerning the Dynamic aspect; and
- Integrity rule, Scope and Risk specifications concerning the Integrity rule aspect.
4.1. Introduction

This chapter tries to answer the question: How are we going to modularise our development?

In certain enterprise-wide information systems, designers may have to work with thousands of classes. In this case, it is difficult to work efficiently and to control the coherence of a large number of conceptual specifications. Therefore, there is a need to work with a part of an IS, in particular a unique and coherent set of conceptual specifications.

On the other hand, it is obvious that the Component-Based Development (CBD) approaches have been considered as a common solution for the implementation and deployment of information systems. Consequently, many research and industrial works have tried to evolve the component paradigm for enterprise-wide information system development.

Last but not least, enterprise integration approaches need information systems are ready to coordinate. Therefore, it is the time to think about a new generation of information systems: coordination-ready information systems [LeDinh 04]. This generation will take into account the current challenges of IS engineering such as components for enterprise-wide IS and enterprise integration.

For this reason, we enrich the M7 reference model with a new aspect: Component aspect that concerns with larger-scale components of enterprise-wide information systems. The purpose of the Component aspect is to enable to work with a part of an IS as a component.

The Component aspect of the M7 reference model is based on the concept of Information System component (ISC) [Léonard 01; LeDinh 02]. An IS component is actually an autonomous information system, which has been developed from analysis to deployment levels.

The remainder of this chapter is as follows. Section 4.2 concerns with the definition and description of an IS component. Section 4.3 introduces how an IS component cooperates with the external environment, in particular information overlaps between
IS components and how to manage them. Section 4.4 shows the meta-model of the Component aspect and its applications. Section 4.5 presents with how to develop information systems with IS components.

4.2. Information System component

4.2.1. Context

One of the main challenges for IS community today is how to manage the developments of different information systems in parallel, while at the same time being able to adapt rapidly to the evolutions of these systems caused by technology, business and organizational environment changes. Historically, there was a well-known “Divide and Conquer” solution based on modularization and separation of concerns, which authorized working in parallel, decentralizing, and making decision [Parnas 72; Arsanjani 02].

In our opinion, components for enterprise-wide information systems require to add a further dimension of separation of concerns. This new dimension could allow data and schemata (such as structural and behavioural specifications) to be used to express the contextual behaviour of components and their implementations at design-time, build-time and run-time.

4.2.2. Proposition

For this reason, we propose the concept of Information System component. An Information System component is a reusable artefact of an IS part that represents adequate characteristics of an autonomous information system.

Indeed, an IS component is a representation of the structure, behaviour, and coherence of information within an enterprise or a part of enterprise. In other words, an IS component is a representation of an information model.

An IS component may represent either the present structure of business and information systems, or a desired future structure. It has a scope indicating the exact boundaries of an IS component. The perspective of an IS component indicates which stakeholders are included.

The intended purpose of an IS component influences its focus. There are some possible purposes such as: i) To develop coordination-ready information systems; ii) To coordinate information systems in development; iii) To coordinate information systems (including legacy systems) in exploitation.
Each IS component has a particular scope, purpose and perspective. Since even the smallest enterprise has multiple purposes, with different areas of interest, it will potentially have many different IS components. Consequently, an information system is constructed by a set of IS components (Figure 4-1).

![Figure 4-1: Information System components](image)

Some approaches argue against the multiplicity of information models representing information system components. Those approaches try to work towards a single global information model for the entire enterprise.

In our opinion, the total convergence of the global information model may not always be feasible. At the time being, enterprises have not yet reached this goal. Therefore, they need interim techniques for the management of several information models, and of the differences and links between those information models.

### 4.2.3. Overview of an Information System component

From external view, an IS component can be considered as an autonomous system that can be developed and delivered independently with the others. However, autonomy does not mean isolation. An IS component can be easily cooperated with other IS components to provide more valuable functionality. The process of developing IS components is often seen as a process of discovery of pre-existing facts, but it is more useful to see it as a process of negotiation: developing a consensus between different stakeholders within the area of concerns.

An IS component is cooperative if it shares goals with other elements in its environment such as other IS components, organizational roles and the enterprise itself in order to contribute towards the fulfilment of common goals (Figure 4-2).
From the internal view, IS components can be viewed using the three facets of the Cooperative information system framework [Michelis 98a, 98b]:

- The Organizational facet concerns with managing work from a formal organizational perspective;
- The Group collaboration facet concerns with organizational roles/responsibilities working on common business process; and
- The Systems facet concerns with computerized systems that support business activities.

Figure 4-3 gives a brief view of the different elements of an IS component in accordance with the three facets of IS.

In this figure, there are three standpoints for viewing the work within an enterprise: the Operational perspective as executions of operations on a system; the Practical perspective as events in the history of group practice; or the Managerial perspective as performances intended to fulfil organizational objectives in accordance with organizational rules.

In our approach, the Practical perspective, which is reflected by the Group Collaboration facet, contains the Responsibility zone element. The Managerial perspective, which is reflected by the Organizational facet, contains the Business activities element. The Operational perspective, which is reflected by the System facet, contains the elements of the Static, Dynamic and Integrity rule aspects of IS.
**Group Collaboration facet.** Let’s discuss about the elements of the Group Collaboration facet. The “IS component” concept historically can be considered as the extension and amelioration of the term *module*, which was the key concept of modularization and separation of concerns approaches.

In this context, “IS component” is considered to be a *responsibility assignment* rather than a sub-system. The design decisions have to make up *before* the work on independent components can begin. It is the reason why we define *Responsibility zone* as an element of the Group Collaboration facet. This element could play an intermediary role between the System facet and the Group Collaboration facet.

**Organizational facet.** Concerning the Organizational facet, in a similar way, business processes can also be decomposed into business activities corresponding to the responsibility assignment process. Therefore, the main objective of an IS component is to fulfil organizational objectives of the business activities to which it supports. It is the reason why the Organizational facet contains the *Business activities* element.

**System facet.** Last but not least, the System facet contains the major parts of an IS component that can be represented by the elements of the three aspects of an IS: the Static, Dynamic and Integrity rule aspects. These elements are the fundamental elements of an IS component and will be developed throughout the IS life cycle.

### 4.2.4. Specification of an Information System component

An IS component can be specified following the traditional levels of abstraction such as analysis, design, implementation and deployment levels.

The specification of an IS component at different levels of abstraction can be summarized as the followings:

- At analysis level, the specification of an IS component includes the descriptions in detail of the elements of the three fundamental aspects of an IS: the Static, Dynamic and Rule aspects.
- At design level, the specification of an IS component concerns with the architecture of a computer-based solution that may include multiple tiers, and how the implementations of the elements correspond to each tier.
- At implementation level, actual coding of the implementations of the elements is developed.
- At deployment level, an IS component is appeared as a set of executables representing their corresponding tiers on various client machines, on server computers, and perhaps a set of corresponding database objects.
As a matter of fact, the specification of an IS component at design, implementation and deployment levels is very detailed and complex because the autonomy, heterogeneity and distribution of computerized solutions. In fact, they are beyond the scope of this thesis. In the following, we briefly introduce the specification of an IS component at analysis level.

**Specification of an Information System component at analysis level.**
To observe an object, we often use two complementary views: as a thing or as a process. Each view has been chosen depending on how we wish to view the event and what perspective we decide to focus on. Accordingly, the representation of an IS component must include these two views: static view (as a thing) and dynamic view (as a process). Moreover, it is also necessary to provide an additional view: integrity view to guarantee the coherence between these two complementary views.

At analysis level, the specification of an IS component using the M7 concepts is represented by:

- A unique hyperclass and its dependents (representing the Static aspect);
- A set of events and processes defined on that hyperclass and their dependents (representing the Dynamic aspect);
- A set of integrity rules, whose scopes defined on that hyperclass, and their dependents (representing the Integrity rule aspect).

The formal definition of the specification of an IS component at analysis level is as the following:

**Definition:** Specification of an IS component at analysis level is

a tuple $<Hcl, Cl_0, A, S_{kl}, M, E_m, P_m, Ir_n>$ where:

i) A unique hyperclass $Hcl$;
ii) A set of $n$ classes $Cl_0, Cl_1, \ldots Cl_n$ that constitutes the hyperclass $Hcl$;
iii) A set of attributes $\{A; attribute from class Cl_i to Cl_j and p,q \in [1,n]\}$;
iv) A set of dynamic states $\{S; state k of objects of class Cl_i and i \in [1,n]\}$;
v) A set of methods $\{M; method l of class Cl_i and i \in [1,n]\}$;
vi) A set of events $E_m$ that may occur to the component $IS_0$ and their corresponding processes $P_m$ that can access the set of classes of the hyperclass $Hcl$;

vii) A set of integrity rules $Ir_n$ defined on classes $Cl_0, Cl_1, \ldots Cl_n$.

**4.2.5. Example about IS components**

Let’s continue to discuss the information system for a retail-trading company (mentioned in Chapter 3): the IS for MATIS company (called MATIS-IS)

This IS aims to support traditional business processes of a business organization such as Sale/Collection, Purchase/Acquisition, and Conversion processes.
Therefore, this information system is built upon three IS components corresponding to its main business domains such as Sale management, Inventory management and Purchase management IS components (Figure 4-4).

![Diagram of IS components](image)

**Figure 4-4: The three IS components of the MATIS-IS.**

Let’s concentrate on the Sale management IS component and its information overlap with other IS components. The Sale management IS component aims to support the direct sale activities, which are responsible by the Sale section. This IS component contains the information overlapped with the Inventory management IS component, which is responsible by the Inventory section.

### 4.2.5.1. The Static aspect

The Static aspect of the Sale management IS component has been explained in Figure 4-5.a by the class schema of the Sale hyperclass. This hyperclass has the Sale order class as key class. From the Sale order class, one can navigate to other classes such as Customer, Sale order Items, Inventory Items, and Payment classes.

Similarly, the Inventory hyperclass represents the Static aspect of the Inventory management IS component (Figure 4-5.b). This hyperclass has the Inventory item tuple-class as its key class. From this class, we can navigate to other classes such as Sale order item, Sale order, Purchase order item, and Purchase order.

![Diagram of Static aspects](image)

**Figure 4-5: The Static aspects of the Sale management and the Inventory IS components.**
4.2.5.2. The Dynamic aspect

The Dynamic aspect of the Sale management IS component has been introduced in Figure 4-6.a by the global life-cycle diagram using Node-Star diagram [Léonard 03]. Nodes are associated to (pre- and post-) hyperclasses and stars are associated to processes.

Figure 4-6: The Dynamic aspects of the Sale management and Inventory management IS components.

There are four events that may affect the behaviour of the Sale management IS component (Section 3.3.2):

- The e1 vent: “When a customer demands for items” triggering the “p1. Receiving-and-Validating-demand” process;
- The e2 vent: “When a customer gives personal information and receives items” triggering the “p2. Getting-Customer-Info” process;
- The e3 vent: “When a customer pays for the sale order” triggering the “p4. Receiving-payment” process; and
- The e4 vent: “When a customer returns items” triggering the “p5. Receiving-Returned-Items” process.

Concerning the Inventory management IS component, there are the four events may happen (Figure 4-6.b):
• The \textit{e11} event, which is happened when creating a new inventory item, triggers the \textit{“p11. Creating new inventory item”} process;
• The \textit{e12} event, which is happened when receiving items ordered by a purchase order, triggers \textit{“p12. Increasing quantity in stock”} process;
• The \textit{e13} event, which is happened when delivering item sold by a sale order, triggers \textit{“p13. Decreasing quantity in stock”} process; and
• The \textit{e14} event, which is happened when receiving items returned by a sale order, triggers \textit{“p14. Undo Decreasing quantity in stock”} process.

4.2.5.3. The Integrity Rule aspect

The IR aspect of an IS component can be represented formally in detail with \textit{integrity rule, context, scope}, and \textit{risk}.

Table 4-1 illustrates the summary descriptions of the Integrity rule aspect of the \textit{Sale management IS} component.

Let’s focus on the IR#1, described as: “The sale order items must be the items that currently sold by the company”. The \textit{context} of this IR is the \textit{Sale} hyperclass. Its \textit{scopes} are the \textit{Sale order item} and \textit{Inventory item} classes. The IR can be verified by \textit{Checking stock} process. Its \textit{risks} are the methods of creation of \textit{Sale order Item} as well as deletion of \textit{Inventory item}.

<table>
<thead>
<tr>
<th>Integrity Rule</th>
<th>Expression</th>
<th>Scopes</th>
<th>Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR#1</td>
<td>The sale order items must be the items that currently sold by the company</td>
<td>Sale order item, Inventory item</td>
<td>- Creating a new sale order item;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Deleting an existing inventory item</td>
</tr>
<tr>
<td>IR#2</td>
<td>The amount of an sale order item is equal to its unit price * quantity ordered</td>
<td>Sale order, Sale order item, Inventory item</td>
<td>- Creating a new sale order item;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Updating the \textit{QuantityOrdered} attribute of the Sale order item tuple-class;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Updating the \textit{ItemUnitPrice} attribute of the Inventory item tuple-class;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Updating the \textit{SO_ItemAmount} attribute of the Sale order item tuple-class.</td>
</tr>
<tr>
<td>IR#3</td>
<td>The total amount of an sale order is equal the total of all its line item amount</td>
<td>Sale order, Sale order item</td>
<td>- Creating a new sale order;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Updating the \textit{SaleOrderAmount} attribute;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Creating a new sale order item;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Updating the \textit{SO_ItemAmount} attribute of the Sale order item tuple-class.</td>
</tr>
</tbody>
</table>
4.3. Coordinating with external environment

The external environment of an IS component includes the organizational environment corresponding to the Group Collaboration facet and the informatics environment corresponding to the System facet.

In our approach, coordination with the organizational environment is coordinating with the Responsibility zone element; and coordination with the computerized environment is represented by Overlap situation and Overlap protocol elements.

This section begins with the discussion about information overlap. It continues with the proposition of information coordination and ends with the example about information overlap and information coordination.

4.3.1. Information overlap

Information overlap is a fact of life despite the need for independence between components. Information overlap among components is indispensable when several components share a common subset of information.

Consequently, information coordination plays the decisive role of the interaction between IS components. In our opinion, information coordination is much more generic and conforms to the nature of IS.

4.3.1.1. Responsibility zone

In order to represent the coordination of an IS component with the Group Collaboration facet, we define Responsibility zone (RZ) element.

A responsibility zone is a part of organization-working environment to which IS components may correspond. An IS component may correspond to only a unique responsibility zone.
In other words, a responsibility zone of an IS component is the stakeholders of this IS component. RZs are responsible for making all human decisions concerning information overlap and coordination between IS components. Therefore, RZs will play the decisive role in information coordination of IS components.

Indeed, the interdependencies between IS components reflect the real interdependencies between / within the responsibility zones that manage and use these IS components. These differences demand coordination, which is the coordination of responsibility zones and their components (Figure 4-7).

![Diagram](image)

*Figure 4-7: Information coordination between IS components and their responsibility zones.*

### 4.3.1.2. Overlap situation

Information overlap among IS components is indispensable when several components share a common subset of information. An **overlap situation** occurs when there is at least one class or one process is common to several IS components.

There are possibly three types of overlap situations (Figure 4-8):

- **Distinct**: there is no common class and no common process/method between IS components;
- **With borders**: there are common classes; but no common process/method; and
- **With overlaps**: there are common classes; and common processes/methods performing operations on those common classes; and common integrity rules whose scopes are those common classes.

![Diagram](image)

*Figure 4-8: Overlap situations*
Depending to the levels of abstraction and the nature of information overlapped, there are two overlap layers (Figure 4-9): i) Schemata overlap layer concerns with overlaps of the specifications of IS components; and ii) Data overlap layer concerns with overlaps of the data of IS components.

![Figure 4-9: Schemata and Data overlap layers.](image)

### 4.3.2. Information coordination

This part introduces the overlap protocols, which can be used to coordinate information overlapped between IS components, and how to implement overlap protocols.

#### 4.3.2.1. Overlap protocol

There is a need for the coordination policies that manage the information overlap between IS components. For this reason, we define overlap protocol.

An overlap protocol is a protocol that allows each responsibility zone (RZ) to perform its own processes locally. Meanwhile, this protocol also helps the RZ to take into account the processes in other RZs that can influence its own processes.

Presently, we propose the following categories of overlap protocols: Ownership-based overlap protocol, Service-based overlap protocol, and Watch-based overlap protocol.

**Ownership-based overlap protocol** appoints which IS component would play the role of the owner for each common object. The owner of an object takes the responsibility for defining, developing and maintaining it. The other IS components (as the users) may communicate to the owner to obtain information about this object.
Service-based overlap protocol appoints which IS component would play the role of the provider (or the custodian) for each common object. The provider of an object takes the responsibility for providing services related to this object. This protocol allows other IS components (as the requesters) to send a request to perform a process to the provider. Normally, the provider will perform the requested process and return the result to the requested IS component.

Watch-based overlap protocol allows an IS component (as the co-owner) to monitor the consequences that causes by the other co-owners, which have performed a process overlapped between them.

Indeed, the Ownership-based overlap protocol conforms to With-borders overlap situations; meanwhile the Service-based and the Watch-based overlap protocols conform to With-overlaps overlap situations.

4.3.2.2. Implementation of overlap protocol

At first, in order to implement overlap protocols, there is a need to specify the way of working between IS components. Therefore, collaboration style needs to be determined for each layer of an overlap protocol.

Collaboration styles.
Collaboration styles indicate the way in which IS components work together and exchange their information.

There are three collaboration styles as Point-to-point collaboration, Bus-based collaboration, and Coordinated-based collaboration.
**Point-to-point collaboration** is a style in which one IS component can directly start the communications to other IS components.

**Bus-based collaboration** is a style in which all information exchanges between IS components is realized through an integration infrastructure called *bus*.

**Coordinator-based collaboration** is a style in which all information exchanges between IS components is controlled and/or realized through a *coordinator*.

**Integration technologies.**
Indeed, there are various integration technologies can be used to implement overlap protocols [Juric 01; Linthicum 03; Hohpe 03]. Nowadays, the widely known technologies of information coordination today are:

- *File transfer technology*: This technology allows each system produce files of shared information consumed by other systems and consume files produced by other system.
- *Database access technologies*: Database access technologies provide access to the database through an abstraction layer. We can use DBMS connectivity to access directly databases, such as JDBC (Java Database Connectivity), JDO (Java Data Objects) in Java platform; ODBC (Open Database Connectivity), ADO (Active Data Objects) in Microsoft platform, etc.
• **Message-based technologies:** Message-based technologies provide an infrastructure that enables communication between systems over distributed and heterogeneous platforms, and reduces complexity because it hides the communication details and the details of platforms and protocols involved. Those technologies use common communication protocol to exchange information over distributed and heterogeneous platforms, such as Message Oriented Middleware (MOM), Object Request Broker (with three major standards as CORBA compliant, Java RMI and MS COM/DCOM), and so on.

• **Service-based technologies:** Service-based technologies provide an infrastructure intended to enhance interoperability between systems over heterogeneous platforms using standard API (Application Program Interface) to invoke processes or services. The most popular technologies are Remote Procedure Calls (RPC), Transaction processing monitors, Application servers. Recently, the current fashion of services based technology is Web services, which uses the standards such as SOAP and XML [Champion 02].

The Data transfer and Database Access technologies can be used to implement Ownership based overlap protocol. Meanwhile, the Message based technologies can be used to implement Watch-based overlap protocol; and the Service based technologies can be used to implement Service based overlap protocol.

As the matter of fact, an enterprise may have to use several different integration technologies. Advantage of our approach is that those technologies can be coexisted and coordinated together.

### 4.3.3. Example about information overlap and coordination

Let’s continue with the MATIS-IS example. The MATIS-IS is constructed by three IS components: *Sale management*, *Inventory management* and *Purchase management* IS component.

#### 4.3.3.1. Responsibility zone

Correspondingly, there are three responsibility zones: *Sale management*, *Inventory management* and *Purchase management* RZs that correspond to these three IS components.

Let’s discuss more about the *Sale management* RZ, which corresponds to the *Sale management* IS component. This RZ is responsible to perform the business activities of the *Sale/Collection* process.
Depending on the organization structure of the MATIS enterprise, the Sale management RZ may involve several roles. There are two categories of involved roles:

- The first category of roles concerns with the usage and management of Sale management IS component such as sales clerk, deliverer, cashier, sale manager, etc.
- The second category of roles concerns with the development and evolution of Sale management IS component such as project leader, business analyst, system analyst, developer, etc.

4.3.3.2. Overlap situations

Let us continue with the information overlaps between the Sale management and Inventory management IS components (Figure 4-10).

There are three overlap situations: $os_1$, $os_2$, and $os_3$:

- $os_1$ is the overlap situation concerning the Sale order class, which is overlapped between these two IS components. It is a with-border overlap situation.
- $os_2$ is the overlap situation concerning the Sale order item class, which is also overlapped between two IS components. It is a with-border overlap situation.
- $os_3$ is the overlap situation concerning the Inventory item class, which is overlapped between the two IS components and both of them may need to perform processes on it.

![Figure 4-10: Overlap situations between Sale management and Inventory management IS components](image-url)
4.3.3.3. **Overlap protocols**

Let’s continue with the overlap protocols proposed for the $os_1$, $os_2$, and $os_3$ overlap situations as well as the implementation of these overlap protocols.

For $os_1$ and $os_2$, we propose to apply the *Ownership-based overlap protocol* for the both overlap layers (i.e. the schemata overlap layer and the data overlap layer). It is assumed that the *Sale management IS component* is the *owner* of the *Sale order* and *Sale order Item* classes. In consequences, the *Sale management responsibility zone (RZ)* of this IS component is responsible for creation, modification and deletion of the specification (i.e. schemata) as well as the objects (i.e. data) of this class.

For the schemata overlap layer of $os_3$, we propose to apply the *Watch-based overlap protocol*. Consequently, both the RZs of the *Sale management* and *Inventory management IS components* can perform the processes to create, modify or delete the specifications of the *Inventory item* class and its dependents in their own workspaces. Each RZ has the facility to monitor the consequences when the other RZs perform a process belonged to the overlap zone and then can update its workspace correspondingly.

For $os_3$, the data overlap layer is handled by the *Service-based overlap protocol*. It is assumed that the RZ of the *Inventory management IS component* is also the *custodian* (or the provider) of the *Inventory item* class. Therefore, only the RZ of the *Inventory management IS component* can perform the creation, modification and deletion of objects of the *Inventory item* class. The RZ of the *Sale management IS component* can request a service provided by the custodian to perform an operation on objects of the *Inventory item* class.

4.3.3.4. **Implementation of overlap protocols**

**Collaboration styles.**

In our opinion, for a small size company as MATIS, it is not necessary to establish a bus or a coordinator at the time being. Therefore, we propose the Point-to-point collaboration style for all layers of the $os_1$, $os_2$ and $os_3$ overlap situations.

Figure 4-11 presents the collaboration between the *Sale management* and *Inventory management IS components*. 
**Integration technologies.**

It is supposed that the MATIS-IS, including the Sale management and Inventory management IS component, is built upon the client-server architecture with the J2EE standard, Java platform as front-end and Relational DBMS as back-end. Besides, each IS component stores specifications in its own repository.

The Database Access technologies can be used to implement the Ownership-based overlap protocols for the Data overlap layer of the os₁ and os₂ overlap situations. Therefore, the JDBC can be used to implement those overlap protocols.

Furthermore, the File transfer technology can be used to implement the Ownership-based overlap protocols for the Schemata overlap layer of the os₁ and os₂ overlap situations. It means that the RZ of the Sale management IS component can produce file of shared specifications to consume by the Inventory management IS component.

Concerning the Watch-based overlap protocols for the Schemata overlap layer of the os₃ overlap situation, each time an IS component modifies the specifications of the Inventory item class or its dependents in its own repository, the other IS component must have the facility to monitor the consequences. A Java based message broker support such as an API called JMS (Java Message Service) can be used in this case. JMS defines a loosely coupled system communication mechanism that enables J2EE components to send and receive messages asynchronously.

Finally, the Remote procedure Call (RPC) technology can be used to implement the Services-based overlap protocols for the Data overlap layer of the os₃ overlap situation. The RZ of the Inventory management IS component can defines the stored procedures corresponding to overlapped processes and provide necessary information about those procedures to the RZ of the Sale management IS component. The RZ of the Sale management IS component can request a service by calling a corresponding remote procedure.
4.4. The Component aspect

The Component aspect concerns with the information about IS components and the information overlaps between them. In the following, the meta-model and the applications of the Component aspect are introduced.

4.4.1. Meta-model of the Component aspect

Figure 4-12 presents the meta-model of the Component aspect.

An IS component includes a unique hyperclass $Hcl_1$, a set of events and corresponding processes whose context is in $Hcl_1$, a set of integrity rule defined on $Hcl_1$.

An IS component corresponds to a responsibility zone defined on the Group collaboration facet of the IS.

There are overlap situations between IS components. An overlap situation happens when there are common classes between two or several IS components.

“With overlap” overlap situation happens when there are common processes between these IS components. On the contrary, “With border” overlap situation happens when there is no common process.

An overlap situation may apply an overlap protocol to manage its common classes and processes.
4.4.2. Applications of the Component aspect

The main purposes of the Component aspect or of IS components are:

- To develop coordination-ready information systems;
- To coordinate information systems in development; and
- To coordinate information systems in exploitation.

**Developing coordination-ready information systems.**
The main purpose of the Information system components is to *enable* to work with a part of an IS as a component. Information System components can be used to build coordination-ready information systems, which are highly componential, ready to coordinate, and easy to evolve.

For this reason, we propose the *Information System component based development* (IS-CBD) to develop coordination-ready information systems. To achieve its purpose, the IS-CBD approach provides a component-based framework, which handles IS components at earlier phase of development process. Furthermore, the approach supports continuous integration, which takes into account the issue of information coordination at each phase of the development process. The next section will present more details about the IS-CBD.

**Coordinating information systems in development.**
Coordination of information systems in development is also the main objective of this thesis, which proposes a *conceptual framework for Information System upon Information Systems* (ISIS). An ISIS is an information system that aims at coordinating information resources used in IS development process. A conceptual framework for ISIS deals with the representation, management and coordination of different types of conceptual specifications within/between IS components. The next chapters of this thesis will continue to clarify the conceptual framework for ISIS.

**Coordinating information systems in exploitation.**
Coordination of information systems in exploitation addresses the issue of enterprise integration, which is always the hot topic of the decade. Our approach for coordinating is based on the *Information kernel*. The Information kernel is a backbone to coordinate different applications and to adapt or integrate new applications included ERP packages in order to construct the global information system. More details about the Information kernel can be found in [Léonard 01].
4.5. Information System Component Based Development approach

This section introduces the Information System Component Based Development (IS-CBD) approach and presents how to develop information systems with IS components.

4.5.1. Motivation

The ISC approach aims at providing a component based development approach that is able to develop “coordination-ready” information systems.

To achieve its purpose, the ISC approach must firstly provide a component-based framework, which handles components for enterprise-wide information systems at high level of abstraction. Secondly, the approach should be well adapted to the nature of IS. Lastly, this approach must support continuous integration, which takes into account the issue of enterprise integration at each phase of development process.

Components for enterprise-wide information systems.
According to [Hopkins 00], there are two engineering drivers in the development of a component-based system: Reuse and Evolution. The ISC has focused on the second engineering driver. The purpose of this approach is to build large-scale and enterprise-wide information systems that are highly componential, ready to integrate, and easy to evolve. Furthermore, the ISC approach also aims to support larger-grained components and to handle them throughout all phases of the development process. An IS component can be considered an autonomous unit that will be developed from analysis to deployment levels.

Conformation to the nature of information system.
It is now quite clear that cost-effective IS solution building is not only the matter of technology but also the matters of business and organization. For this reason, the ISC approach adopts the framework for cooperative information systems [Michelis 98a, 98b] to define IS components. Each IS component is represented by three facets and their interactions: the Organizational facet, Group collaboration facet; and Systems facet. Moreover, the collaboration between IS components is fundamentally based on the information overlaps among them.

Continuous integration.
Continuous integration in the ISC approach allows IS professionals continuously integrate their works using a common integration infrastructure. The approach aims to support continuous integration with two categories of coordination: i) Inter-IS Coordination is the coordination of elements of different components; and ii) Intra-IS Coordination is the coordination of elements of the same component at different levels of abstraction.
4.5.2. Related works

According to our point of view, there are three categories of the related works that have tried to support reuse in many different ways and at different levels of abstraction. They are Pattern, Business Object and Component.

Pattern.
A pattern is defined as a generic micro-architecture that supplies a common solution of a recurring problem in a particular context. There are different types of patterns depending on the level of abstraction to which they address: i) Analysis patterns [Coad 96; Fowler 97] help IS professionals constructing conceptual models that are best suited to requirements; ii) Design patterns [Gamma 95] capture the experience and knowledge used for the design phase; and iii) Implementation patterns [Coad 92; Larman 98] handle components at the low-level of abstraction such as classes, objects and their interactions.

Business Object.
According to the Object Management Group, a Business Object (BO) [OMG 97] captures “information about the real world’s concept, operations on that concept, constraints on those operations, and relationships between that concept and other business concepts”.

Component.
In general, the traditional Component-Based Development (CBD) approaches [Booch 98; Szyperski 98; Souza 99; Hopkins 00; Arsanjani 02] have more concentrated on the implementation and deployment levels: i) Implementation components are defined as “a physical and replaceable part of a system that conforms to and provides the realization of a set of interfaces”; and ii) Deployment components are the components at deployment level such as DCOM, CORBA, JavaBean, etc.

Recently, the Business component approach [Herzum 00] applies and extends component thinking to all aspects of development, deployment, run-time, and evolution. A Business component (BC) is defined as a component that implements a single autonomous business concept. There are several types of components depending on their granularity levels such as Distributed component, Business component, Business component system and Federation of system-level component.

Differences between the ISC approach with the other approaches.
The main differences between the ISC approach and the other approaches can be deduced from the main characteristics of an IS component:

- Supporting components throughout all levels of abstraction, from analysis to deployment levels;
- Focusing on the development and evolution of enterprise-wide IS as well as the issue of enterprise integration;
Concerning with the nature of IS by covering all the facets of IS such as the System facet, the Group Collaboration facet and the Organizational facet; and

Guaranteeing the completeness and the coherence of an IS component by fulfilling the three aspects of an IS such as the Static, the Dynamic and the Integrity rule aspects.

Accordingly, the differences between the ISC approach and other approaches are as the followings:

- **Concerning the levels of abstraction**: the Pattern approaches have often concerned with one specific level of abstraction. So do the Component approaches (except the Business Component approach). On the contrary, the ISC approach has focused on all levels of abstraction.

- **Concerning the three aspects of an IS**: almost related approaches have more or less concentrated on one aspect of an IS, normally the Static aspect represented with classes, meanwhile the ISC approach always focuses on all the three aspects of an IS.

- **Concerning the three facets of IS**: as we know, there is no other approach that entirely focuses on the three facets of IS in a coherent manner as the ISC approach does.

It is obvious that the ISC approach is very similar to the Business component approach, which aims to support components across the whole development life cycle. Let’s compare the ISC approach with the BC approach:

The key concept of the ISC approach is “Information System Component” that aims to represent an artefact of IS part, while the “Business component” aims to represent an implementation of an artefact of the business world;

The second difference comes from the component granularity and recursion techniques. The ISC approach employs the continuous recursion technique. This technique allows dividing the domain space into very coarse-grained components. Thus these very coarse-gained components continue to be divided into components of a lower granularity and so on. On the contrary, the BC approach employs the discrete recursion technique that defines specific granularity levels and each level identifies a unique category of components;

Lastly, in ISC approach, the cooperation between IS components is based on information sharing, meanwhile the BC approach is based on the interaction between components.
4.5.3. IS development with IS components

This section gives an overview of IS development with IS components by concentrating on the fundamental questions of Design by reuse and Design for reuse.

According to [Fugini 93; Chabane 99], in the point of view of development process, it is necessary to distinguish two types of complementary perspectives of reuse: Design by reuse and Design for reuse. The design for reuse concerns with the production of reusable components including the tasks of extraction, representation and organization of components. Meanwhile, the design by reuse concerns with selection, adaptation and integration of components to develop new information systems.

Figure 4-13 illustrates the two principal processes: design for reuse and design for reuse. The purpose of design for reuse is to construct a library of generic IS components to reuse later. Then, the design by reuse process continues to construct the specification of the IS structure at global level (in short: global IS structure) that provides a unified view of most of information resources.

Actually, the global IS structure can be considered as a unique and coherent set of information resources used to support integration and coordination of information systems. In the ISC approach, the global IS structure is constructed by IS components and the information overlaps between them.

![Figure 4-13: IS development with IS components.](image-url)
4.5.3.1. Design by reuse

In order to support design by reuse, we define the concept of a generic IS component. A generic IS component can be considered as an IS component that has been constructed from the potential of IS and would be technology-independent and organization-independent in order to be reused later.

In the other words, a generic IS component has been constructed only from the fundamental elements of the System facet, concerning the Static, dynamic, and Integrity rule aspects.

The principal question of design by reuse in our IS component-based approach is ‘how to integrate an IS component into the global IS structure’. In the context of the IS component-based development, the integration of an IS component includes two steps: i) Adaptation of a generic IS component; and ii) Integration of an IS component into the global IS structure.

Adaptation of a generic IS component.
Adaptation happens when there are changes in the existing structure of the global conceptual model of IS to adapt to the new IS component. Depending on the scope of adaptation, we propose two types of adaptation: local adaptation and global adaptation.

Local adaptation is situated in the scope of an IS component. This type of adaptation may bring to the changes of the structure of this component such as:

- Condensations or expansions of the generic IS component in accordance with the requirements;
- Specifications or generalizations of the generic IS component (for instance, to be conformed to the organizational environment);
- Changes in the elements of the generic IS component as additions, deletions and modifications of certain classes, states, events, processes and integrity rules of the generic IS component.

Global adaptation performs the transformation of the IS structure at global level. This kind of adaptation may lead to the transformations of several IS components and their interactions to accommodate themselves to the new situation:

- The most important of the global adaptation is to define the strategy to handle conflicts caused by information sharing among IS components. The common solution is to apply overlap protocols in the System facet combined with coordination policies in the other facets of IS;
- Besides, the solution to overcome the side effects caused by the integrity rules of the new IS component needs to take into account the ability to guarantee the global integrity of the IS.
**Integration of the new IS component into the global conceptual model.**
Let us continue to integrate the adapted component into the global IS structure of information systems.

One of the most important difficulties of this work is to solve the problem of *semantic heterogeneity*. In the context of the IS component-based development, the problem of semantic heterogeneity is considered as the identification of semantically common elements in different IS components and the resolution of schematic differences among them.

At the time being, we employ the techniques for database integration (proposed by [Parent 98]) to identify the semantic correspondences between the elements of the new IS component and the elements of existing IS components and then to solve the inter-components conflicts and unifies corresponding elements afterward [Léonard 01].

4.5.3.2. **Design for reuse**

Let us continue with the extraction of a generic IS component and propose some strategies to carry out it.

**Extraction of a generic IS component.**
To extract a generic IS component whose elementary characteristics are adaptability, flexibility and evolution, it is necessary to focus on the potential of IS instead of the solution for current situation/project of IS. In our point of view, the potential of IS can be represented by the three facets of IS and the different aspects of each facet.

The techniques to extract components, called “Divide and Conquer”, are essentially heuristics. The purpose of these techniques is to group together elements, which could work on as a unique and coherent component. Each technique has different algorithms to calculate the *affinities* and *semantic relations* between the elements of IS components.

For extracting IS components, we propose two steps as the following:

- *Extracting the elements of a generic IS component.* At present, we use some techniques derived from the useful work about *information clustering* [Richard 94] with the main focus on extracting the *key elements* of each aspect of IS, for example extracting *classes* of the static aspect, *activities / processes* of the dynamic aspect and/or *integrity rules* of the integrity rule aspect. After that, from the existing elements, we try to find more elements that have semantic relations to the existing elements;

- *Verifying the completeness and the validity of a generic IS component.* Once a generic IS component is extracted, it is indispensable to verify the completeness and the validity properties of its elements. These properties
Information System Upon Information Systems

can be achieved by fulfilling all aspects of a generic IS component such as the static, the dynamic and the integrity rule aspects.

**Strategies to extract a generic IS component.**
A generic IS component can be extracted directly from the real world, indirectly from the design artefacts of the other approaches, or from the existing information systems. For this reason, we propose three strategies (derived from the idea of [Batini 92]) to extract a generic IS component such as **top-down, bottom-up and inside out.**

The *top-down strategy* begins with the knowledge about the real world domain of IS professionals, including the three main activities:

- From the real world domain, extracting the elements of a generic IS component;
- From the obtained elements, trying to find other elements to complete the specification of the generic IS component; and
- Verifying the completeness and the coherence between the elements of the IS component. If these properties have not been satisfied, returning to step 2.

On the contrary, the *bottom-up strategy* begins with the technology artefacts in the computerized environment, including:

- From the computerized environment, reverse-engineering the elements of a generic IS component (for example, reverse-engineering from legacy applications or databases);
- Verifying the completeness and the coherence between the elements of the specification of the generic IS component; and
- If these properties have not been satisfied, completing the specification by continuing to reverse or specifying new elements with the help of IS professionals.

Finally, the *inside-out strategy* (or *outside-in strategy*) that aims to reuse the design artefacts of the other CBD approaches. This strategy includes the following main activities:

- From the working environment of the other CBD approaches, extracting the elements of an IS component (for example, extracting generic IS components from the repositories storing specifications of *Business objects* or *Analysis patterns*);
- Verifying the completeness and the coherence between the elements of the specification of the IS component; and
- If these properties have not been satisfied, continuing to find new elements to complete the specification of the IS component with the help of IS professionals.
4.6. Conclusion

In this chapter, the M7 reference model is enriched with the new aspect: Component aspect, including the Information system component, Overlap situation and Overlap protocol concepts.

With this new aspect, the reference model can be also used as the framework for specifying conceptual specifications of large scale and component based information systems.

The purpose of Information System component (ISC) concept is to enable to work with a part of an IS as a component. An IS component can be considered as an autonomous information system that will be developed from analysis to deployment levels.

Consequently, traditional information system can be constructed by a set of IS components. Furthermore, the Overlap situation and Overlap protocol concepts aim to handle the information coordination between IS components.

Moreover, the Information System Component Based approach (IS-CBD) can be used to develop “coordination ready” information systems. The IS-CBD approach aims at handling large-scale components for enterprise-wide information systems at high level of abstraction; being well adapted to the nature of IS; and supporting continuous integration, which takes into account the issue of enterprise integration at each phase of development process.
This chapter aims at presenting how an ISIS stores, integrates, processes and manages specifications. The chapter begins with the overview of an ISIS, then continues with its architecture, and terminates with the discussion about IS project management with the ISIS.

5.1. Overview of an ISIS

5.1.1.1. Definition

An Information System upon Information System (ISIS) is defined as a typical information system supporting the development process of information systems.

Therefore, the first objective of an ISIS is to identify, represent, manage, integrate and coordinate specifications produced and consumed by activities of the IS development.

Furthermore, to reduce the complexity of IS development, an ISIS should also support working with parts of an IS. Consequently, the second objective is to allow working with a set of specifications as an Information System component.

5.1.1.2. Purposes of an ISIS

The perspective of building an ISIS is to develop a strategy for the enterprise. The ISIS may represent the important information resources used by principal activities of the enterprise. An information model of the whole enterprise can be clustered into sub-models, each of which are then the subject of detailed analysis and IS development projects.

For this reason, an ISIS may have several purposes such as:

- To establish the relation between the strategic business objectives and the IS development projects;
- To define the scope of business areas and of development projects;
- To provide a framework for integrating and coordinating IS development projects;
• To provide a framework for planning and implementing the necessary informatics infrastructures.

5.1.1.3. Benefits of an ISIS

An ISIS significantly reduces both the time and cost of analysis and development by documenting and managing conceptual specifications such as information structures, behaviours, integrity rules as well as transformation rules.

Managing conceptual specifications is extremely important because without the ISIS, those specifications may reside partially or totally in IS experts’ collective memory.

Once the results of the analysis and development are captured and managed by the ISIS, the enterprise can control and manage the development process and gains the benefits of long-lasting investment returns. The ISIS also helps to reduce the costs of future releases and to avoid various developmental errors.

Furthermore, an ISIS can help to overcome the gaps between the reality world and the artificial world by proving an architecture, which supports the coordination between the two worlds. With an ISIS, the reality world can understand and coordinate better with the artificial world and vice versa.

5.1.1.4. Facilities of an ISIS

In order to gain its objectives, an ISIS takes into account the issues of how to work with specifications and with IS components.

Therefore, an ISIS must have at least the following facilities: Representing specifications and IS components, Managing specifications and Coordinating specifications.

Representing specifications and IS components.

An ISIS must be able to handle all the specifications used in IS development. Representing different category of specifications could be difficult because there may exist wide variety of specifications’ categories that exist within an enterprise. Besides, an ISIS also provides the facility to work with IS components, which are represented by subsets of specifications.

Managing specifications.

ISIS must incorporate administrative facilities to allow business users and IS experts to manage specifications such as:

• Integrity validation: the first facility is to guarantee the completeness, validity, and coherence of specifications as well as their interdependencies.
• Concurrent access: If ISIS users need to concurrently access the specifications, the ISIS must provide the means to manage conflicts that
may arise when two or more users attempt to manipulate the same specifications. Moreover, conflicts occur when multiple users attempt to update same specifications; or when two responsibility zones differ in their interpretation of a common specification.

- **Security**: Security is extremely important for specifications access and manipulation, particularly when various responsible persons in a distributed environment use the ISIS. Security is often managed by granting various privileges to different users of RZs, depending on the type of operations they need to perform on the specifications.
- **Evolution support**: Specifications evolve frequently throughout the development lifecycle. The ISIS should be sufficiently flexible to effectively handle the evolution of specifications.

**Coordinating specifications.**

An ISIS must be able to coordinate the common specifications among various IS components. Consequently, it must also guarantee a consistent view of specifications throughout the development life cycle and across the enterprise.

This chapter continues with the discussion about managing specifications. The issues about representing and coordinating specifications will be discussed in the next chapters.

### 5.2. Management of specifications

A challenge of the ISIS approach is how an ISIS can be employed for an enterprise, which has already employed a quality approach, a standard or a particular development process.

For this reason, this section introduces the *Organizational aspect* of the M7 reference model, which includes a set of generic concepts representing the development world. Thus it continues to give an example about those generic concepts.

#### 5.2.1. Organizational aspect of an ISIS

This section aims to extend the M7 reference model with the new concepts concerning the *Organizational aspect*.

These new concepts are indispensable to manage specifications of information systems stored in the ISIS. They must be comprehensive and able to adapt to particular IS development processes as well as organization structures.

Indeed, most of those new concepts are the basic and common concepts of industrial standards for IS development such as *ISO standard* [ISO 9000-3], *ISO/IEC standard*
[ISO/IEC 12207], Rational Unified Process (RUP) [Kruchten 00; RUP 03], UN/CEFACT Modelling Methodology (UMM) [UMM 03], and Capability Maturity Model (CMM) [Paulk 93].

The new concepts of the Organizational aspect are Contributor, Team, Roles, Activity, Information resources, Specification, Conceptual specification, Technical specification, Common specification and Private specification (Figure 5-1).

A development team is a number of persons associated together to carry out an IS development project. A team includes a set of contributors. A team may assume one or several responsibility zones.

A contributor is a person that participates in the IS development process. A contributor may take on several roles.

Roles represent a set of necessary responsibilities, authorities and capabilities to perform the execution of activities of the development process or to watch (monitor) the execution of activities performed by the other roles.

An activity is defined as a unit of work that may produce or consume certain information resources. Activities can be nested: one activity can expand into several activities at a lower level. An activity can be performed by a set of roles and can be watched by another set of roles.

In the context of IS development, an information resource are packages of specifications consumed or produced by the activities of the development process. Information resources are close to the design artefacts in other approaches. An information resource is constructed by a set of specifications.

Figure 5-1: Meta-model of the Organizational aspect.
A **specification** specifies the structure, behaviour or integrity rule of a specific IS concept and its dependent concepts. Concerning the levels of abstraction, there are two types of specifications:

- **Conceptual specifications**, which are products of the analysis stage, represent the business requirements of users.
- **Technical specifications**, which are products of the implementation stage and platform-dependent, represent the internal design of the information system, turned to achieve reasonable performance on the target platform.

A conceptual specification can be implemented in one or several technical specifications.

Concerning the interest of development teams, there are two specialisations of specification: Private specification and Common specification. A **private specification** is belonged to only one team. Meanwhile, a **common specification** is overlapped between several teams. Consequently, a common specification can be referred by several private specifications.

### 5.2.2. An example about the Organizational aspect

The example in this chapter is extracted from our experiment in studying an ISIS for Computer Science Centre (CSC) of University of Natural Sciences - Ho Chi Minh city- Viet Nam that has been mentioned in Chapter 2. In short, this ISIS is called **CSC-ISIS**.

**Teams.**

It is supposed that the CSC has signed a contract for building an IS for one of its customers; the required information system is called the *IS*#1 information system. The customer, which is a commercial enterprise, would like to computerize its business activities. The concerning activities are the activities in the domains of Sale, Inventories control and Purchase.

For this reason, three teams has been created to develop the *IS*#1. They are *IAS*#1, *IAS*#2, and *IAS*#3 teams. The *IAS*#1 team is responsible for the Sale activities; the *IAS*#2 team is responsible with the Inventory control activities; and the *IAS*#3 team is responsible with the Purchase activities.

Let’s focus on the Sale activities and the *IAS*#1 team. This team includes two contributors: $\text{contrib}_1$ and $\text{contrib}_2$ (Figure 5-2). The name of the $\text{Contrib}_1$ contributor is “System analyst B”, and the name of $\text{Contrib}_2$ contributor is “Developer C”.
Information resources.

In the IS#1 information system, the document titled *Top-level-design-for-Database (IS#1 Sale activities)* is an information resource of the IS#1 information system concerning its Sale activities. Indeed, this is a document representing the conceptual specifications of the Static aspect of this part of the IS#1. This document may contain the graphical diagrams as well as the formal descriptions of the hyperclasses, atomic-classes, attributes and keys related to the Sale activities.

Figure 5-3 introduces two information resources. They are: *source1* information resource, titled “*Data definition and database requirements (IS#1 Sale activities)*”, and *source2* information resource, titled “*Top-level-design-for-Database (IS#1 Sale activities)*”.

![Figure 5-2: The IAS#1 team and its contributors.](image)

![Figure 5-3: Information resources of the CSC-ISIS.](image)
Activity.
The CSC intends to employ the ISO/IEC standard for its IS development projects. This standard process, which is proposed based on the ISO/IEC 12207 [ISO/IEC 12207; ISO 9000-3], includes the following main activities such as System requirement analysis, Software require analysis, System architecture design, Software architecture design, Code & Test, Integrate software, Software Qualification Test, Integrate system, and System Qualification Test (Figure 5-4).

![CSC IS development process diagram]

Figure 5-4: Activities of the standard development process of the CSC.

In fact, the term “Software” expresses the set of computer programs, procedures associated with data and documentation that can be developed independently.

Let’s focus on the act1 activity titled “Software architecture design” that can be decomposed into the following sub-activities:

- **Act11: Requirement transforming:** transforming the requirements for the IS into an architecture describes its top level architecture and identifies its components as well as their behaviours;
- **Act12: Interfaces external design:** developing and documenting a top level design for the interfaces external to the IS and between its components;
- **Act13: Database design:** developing and documenting a top level design for the database of the IS;
- **Act14: User document design:** developing and documenting preliminary versions of user document;
- **Act15: Test requirement:** defining and documenting preliminary test requirements;
- **Act16: Evaluation:** evaluating the architecture of the IS and the interface and database design considering the criteria listed in the requirement analysis documents.

For instance, the purpose of the act13 activity, titled “Database-design”, is “to develop and document a top-level design for the database of the IS”. It is a sub-activity of “Software-architecture-design”. Furthermore, this activity consumes the source1 information resource “Data definition and database requirements”, and produces the source2 information resource “Top level design for Database” (Figure 5-3). 
Conceptual specifications.
Let’s discuss about the specifications (of the IS#1 information system) related to the Customer IS concept (Figure 5-5).

Figure 5-5: Some conceptual specifications of the IS#1.

At first, there are the following categories of conceptual specifications:
- **Atomic-class specifications**: include the specifications of the Code and Name atomic-classes;
- **Hyperclass specifications**: include the specification of the Customer hyperclass;
- **Attribute specifications**: include the specifications of the Customer Code and Customer Name attributes;
- **Key specifications**: include the specification of the Customer-key key.

The specifications of the Name and Code atomic-classes and specification of the Customer hyperclass are used to construct the source₁ and source₂ information resources. Moreover, the specifications of the Customer Code and Customer Name attributes, and the Customer Key key are used to construct the source₂ information resource.

Technical specifications.
Besides, there are the following technical specifications such as the Customer table, Customer-Code column, Customer-Name column, and Customer-Key primary key specifications (Figure 5-6). Indeed, the Customer hyperclass conceptual specification is implemented by the Customer table technical specification; the Customer-Code and Customer-Name attribute conceptual specifications are implemented by the Customer-Code and Customer-Name columns technical specifications; and the Customer-Key key conceptual specification is implemented by the Customer-Key primary key technical specification.
Roles.
Figure 5-7 introduces the roles in our example such as: Role1: “Responsible-for-Database-requirements”; Role2: “Author-of-Database-design”; and Role3: “Review-of-Database-design”.

Contributors.
In Figure 5-7, there are two contributors: contrib1 and contrib2. The contrib1 contributor is a “Business analyst B” that takes on the roles of “Responsible-for-Database-requirements”, and “Review-of-Database-design”. The contrib2 contributor is a “System Analyst C” that takes on the role “Author-of-Database-design”.

For example, the Role3 role titled “Review of database design” does not perform any activity, but watches the “act13: Database-design” activity. Meanwhile, the Role2 role titled “Author-of-Database design” performs the “act13: Database-design” activity.
For example, the name of the contributor is “Business analyst B”. Its email address is sa_B@mattis.ch. This contributor takes on the following roles: “Role1: Responsible-for-Database-requirements”, and “Role2: Review-of-Database-design”.

**Private specifications.**
The conceptual specifications and technical specifications as mentioned above are the private specifications, which are belonged to the IAS#1 team.

### 5.3. Architecture of an ISIS

This section concerns with how an ISIS stores, processes and manages specifications. The section begins with the different layers of an ISIS, continues with the overall architecture, and ends with the discussion about how an ISIS coordinates with development tools.

#### 5.3.1. Three layers of an ISIS

An ISIS aims at producing specifications that can be used to support activities of the stakeholders in IS development process.

For this reason, the architecture of an ISIS can be considered as: “a group of interrelated service components and human responsibilities that work collectively to carry out communication, storage, processing and control processes in order to convert specifications into information products that can be used to support coordination, control, validation, forecasting, planning and operational activities of information system development in an enterprise”.

In order to obtain a unified view of specifications and to achieve cross-system consistency, ISIS needs to provide at least the fundamental capacities of such as:

- *Specification representation*: representing, validating, and managing specifications;
- *Specification integration*: various objects representing specifications in development tools and systems need to be integrated with specifications in the ISIS;
- *Specification coordination*: the development tools and systems need to coordinate and exchange their objects. Coordination requires agreements on overlap situations, overlap protocols and responsibilities of each participant.

Consequently, the ISIS architecture should include three layers:

- The *Representation layer* aims at representing different categories of specifications of information systems, including IS components and information overlaps between these IS components.
• The Integration layer integrates specifications of development tools with specifications in the ISIS.
• The Coordination layer deals with coordinating specifications within/between development projects.

Each layer may have several service components to fulfil its objectives. In the following paragraph, the principles of each layer will be illustrated.

5.3.1.1. Representation layer

The purposes of the Representation layer are to represent, validate and manage the objects of an ISIS, which represent specifications used by development teams.

In the ISIS, an object represents a specification. There are two types of objects: Private object and Common object. Private objects represent specifications owned by the RZ of an IS component. Common objects are overlapped between several RZs.

The Representation layer has two workspaces: Private workspace and Common workspace. The Private workspace contains the private objects. Each RZ has its own workspace. The Common workspace contains the common objects shared by several RZs of IS components.

The Representation layer has two service components:
• The Navigator service component provides the interface to allow the ISIS users to specify and complete their own specifications as well as to work with common specifications;
• The Administrator service component provides facilities to support the ISIS administrator in defining and enforcing coordination policies and maintaining the ISIS repository.

5.3.1.2. Integration layer

The Integration layer aims to integrate specifications and services of development tools with specifications and services of the ISIS.

In particular, this layer focuses on the common specifications and how to integrate them to construct a unique view of important information resources of an enterprise. Consequently, an important challenge of this layer is to solve the problem of semantic heterogeneity.

In the context of the ISIS approach, the problem of semantic heterogeneity is considered as the identification of semantically related specifications in development tools and the ISIS and the resolution of schematic differences among them. There are various approaches focusing on database and schema integration such as [Batini 86; Hammer 93; Hull 97; Kashyap 98; Parent 98]. These approaches can be employed to
identify the semantic correspondences between those common specifications and then to solve the conflicts and unifies corresponding specifications afterward.

This layer has at least two service components:
- The *Specification-integration service component* provides facilities to integrate specifications stored in development tools with specifications stored in the ISIS;
- The *Service-integration service component* includes facilities to integrate services provided by development tools and services provided by the ISIS to coordinate specifications in IS development process.

### 5.3.1.3. Coordination layer

The Coordination layer deals with how to coordinate specifications in IS development process. In the context of the ISIS approach, coordination can be considered as managing dependencies between IS components. In fact, the dependencies between IS components are overlap situations, including the specifications corresponding to common classes or common processes.

There are two categories of coordination:
- *Intra-IS coordination* is the coordination of private specifications of the same information system component;
- *Inter-IS coordination* is the coordination of common specifications of different information system components.

In consequences, there are two service components of this layer: *Intra-IS coordination* and *Inter-IS coordination* services components that provides facilities for two categories of coordination.

### 5.3.2. Overall architecture of an ISIS

![Diagram](Figure 5-8: Overall architecture of an ISIS.)
Figure 5-8 introduces the overall architecture of an ISIS. Specifications of the ISIS are stored in the ISIS repository. The repository includes two workspaces: private workspace and common workspace. *Private workspace* stores private specifications of development team. Meanwhile, *common workspace* stores common specifications.

There are three layers that provide facilities to represent, validate, manage, integrate and coordinate specifications of the ISIS such as the Integration, Coordination and Representation layers.

Integration layer provides the facilities to integrate specifications stored in development tools and specifications stored in the ISIS. This layer includes the Specification integration and Middleware integration service components.

Representation layer provides the facilities so that ISIS users, developers and administrators can work with specifications stored in the ISIS repository. It includes two layers: Navigator and Administrator service components.

Finally, Coordination layer provides the facilities to coordination specifications within / between development projects. The layer includes the Intra-IS coordination and Inter-IS coordination service components.

### 5.3.3. Coordinating with development tools

This section firstly introduces the various systems and tools used in IS development. Thus it continues to discuss about the level of integration and different modes of integration between the ISIS with development tools.

#### 5.3.3.1. Development tools

The commonly used systems and tools used in IS development (in short: development tools) are as the following: *CASE tools, Meta data repository tools, IDE tools, DBMSs and SCM tools* (Figure 5-9).

> Figure 5-9: The ISIS and development tools.


CASE tools.
Computer Aided Software Engineering (CASE) tools, which appeared in the 1970s, are one of the first commercial development tools to offer specifications services. The main purpose of CASE tools is to aid the process of designing information systems. Therefore, they need to store schemata (or specifications): data about the data they manage. However, it is not easy to link the specifications from various CASE tools together. Probably, CASE tools’ vendors were unwilling to build such interface because they didn’t want their competitors to be easily immigrating specifications from their tool.

Repository tools.
These tools integrate all specifications sources (form CASE tools, extraction/ transformation tools, and other repositories) into a repository. Indeed, most of the meta-models for these tools focus on the logical and physical level but relatively weak on the conceptual level.

IDE tools.
Integrated Development Environment (IDE) tools provide an environment to development applications, in particular the client side, with a fourth generation language (4GL). These tools concern with certain types of physical specifications such as forms, reports, procedures, etc.

DBMSs.
Database Management System (DBMS) aims to store, validate and manage data of application as well as their manipulations at the server side. These systems works with physical specifications types such as tables, views, stored procedures, transactions, triggers, constraints, etc..

Software Configuration Management.
Software Configuration Management (SCM) tools aim at providing facilities for identifying, monitoring and controlling configuration, managing artefacts of information systems.

5.3.3.2. Level of integration
Concerning the level of integration, there are two extremes (Figure 5-10):

- At one extreme is Full integration where the all the specifications, used by a development team, are stored as ISIS’ objects in the private workspace of that team. Those objects, providing maximum reuse benefits, can be accessed using the Navigator service component of the ISIS. Later on, the development team can decide to put the common objects, whose they are custodians or owners, into the common workspace so that the other teams can access them.
- At the other extreme is Partial integration, where the common specifications, shared by development teams, are stored in the common workspace. Those objects can be shared and coordinated among development teams.
5.3.3.3. Integration Mode

On the other side, concerning the integration modes, the ISIS needs to be flexible in how specifications of a variety of development tools and systems can integrate with the specifications of the ISIS. The generally used integration modes are Direct, Replication, Agent, Registration, and Loose integration modes.

Direct integration mode.
The development team uses directly ISIS’ objects through the Navigator service component of the ISIS. Members of that team can create and maintain those objects, which are stored in the workspace of the ISIS.
Replication integration mode.
In this case, the development team manages its own objects stored in its development tools. However, the copies of those objects will be created in the repository of the ISIS using replication techniques. The ISIS has to verify the coherence between the objects of that team and their copies in the ISIS.

![Diagram of Replication integration mode]

*Figure 5-12: Replication integration mode.*

Agent integration mode.
Development tools of development team do not work directly with the ISIS. Instead, an agent (or an interface) interacts with the ISIS. This agent acts as intermediaries for the tools. The agent recognizes the format requirements for specifications of the tools. The agent can retrieve specifications from the ISIS, and stores them in the tools. On the other hand, the agent can also retrieves the output specifications from the tools and stores them in the ISIS.

![Diagram of Agent integration mode]

*Figure 5-13: Agent integration mode.*

Registration integration mode.
In this case, the ISIS objects represent specifications that are input and output from development tools, which don’t have their own objects. When user invokes a supporting tool, this process causes the required specifications to be extracted from the ISIS and passed to the tool. When the process terminates, the processed specifications from the tool will be stored in the ISIS.
Loose integration mode.
Development tools work independently with the ISIS. The import and export specifications will be performed by scheduled modules. Specifications generated by the tools can be stored, shared, and controlled in the ISIS. Consequently, those tools do not require to be modified to integrate with the ISIS.

Each development team should need to select only one level of integration: full or partial integration. However, a team can use several development tools in development process. Therefore, it is possible that several integration modes can be selected for a team. However, each tool should be integrated with the ISIS using only one integration mode.

5.4. IS project management with the ISIS

This section clarifies the IS project management with the ISIS with the development process and the project organizations and responsibilities.
5.4.1. **ISIS Development process**

In our standpoint, information system is not a simple inert copy of the reality. In fact, information system is a transformation system that is able to improve by itself [Léonard 99b].

Therefore, IS development is considered as the learning process that has no start and no end. Roughly speaking IS development is an iterative process that refines the domain understanding, the computerized solution, and effective plan over several iterations. Each iteration will reach the current objectives of all stakeholders.

The next figure shows the iterative process.

![Diagram of the iterative process](image_url)

*Figure 5-16: The iterative process.*

We adopt the Piaget's theory of *Cognitive development* [Piaget 77] to define the iterative process. There are two cognitive processes that are crucial for progressing from iteration to iteration: *Assimilation* and *Accommodation*. Concerning to the IS domain, taken together, assimilation and accommodation make up the adaptation of information system, which refers to the ability of IS to adapt to its environment and vice versa.

Each iteration begins with the activity that defines its objectives and then continues with the processes of assimilation and accommodation.
Assimilation refers to the way in which responsible persons understand an experience in terms of their way of thinking and current knowledge about the IS. In the ISIS approach, the assimilation process includes two activities:

- Define/refine the IS components: Studying the essential philosophy of IS components in order to identify and extract IS components from the real world; and discovering the interdependences between them.
- Evolve the organization structure: Performing repercussion process such as the redefinition of tasks and the readjustments in organization environment to coordinate with the implementation of IS in informatics environment.

Accommodation happens when there are changes in the existing structure of IS in order to become adapted to the new situation. There are two activities in the accommodation process:

- Develop the information system: developing information system components in parallel or sequence with minimum interaction effort.
- Develop / evolve the IS infrastructure: developing or evolving the IS infrastructure, especially the ISIS, to support the development and exploitation of information systems; and putting the separated developed IS components together to archive a coherent solution with the help of the ISIS.

In the following, we continue to clarify the principles about project management concerning the development of the ISIS, which is the main focus of our framework.

5.4.2. **ISIS project management**

This section presents the organizations and responsibilities of the ISIS project as well as the principles about the development process.

5.4.2.1. **ISIS project organization**

With the existence of the ISIS, it is important to distinguish development teams in an enterprise into:

- **ISIS development team**: (in short: ISIS team) is responsible to carry out the IS development project. This team takes on the responsibilities to integrate, manage and coordinate specifications used by several IS development teams.

- **IS development team**: (in short: Development team) is responsible to carry out an IS development project. This team takes on a responsibility zone. Therefore, it is in charge of IS components assumed by that RZ.

In the following of this section, the responsibilities and the general structure of an ISIS develop team are presented.
5.4.2.2. Responsibilities of the ISIS team.

An ISIS team is an independent team and better dedicated directly to the head of IT staff. The team needs to reside at a high level because it has to work with other IS development teams and to define procedures that everyone must follow.

The ISIS team has the following main responsibilities:

- Identifying different categories of specifications, information resources constructed by those specifications, development activities consumed and produced those information resources;
- Building and maintaining the ISIS architecture;
- Selecting metadata tools and working with tool vendors;
- Administrating specifications managed by the ISIS;
- Integrating specifications in development tools with specifications in the ISIS;
- Working with other IS development teams, negotiating and defining overlap situations, overlap protocols and responsibilities of each development team, and
- Defining and enforcing coordination policies for development teams to follow and supporting coordination of specifications in IS development process.

5.4.2.3. Roles

Although ISIS may differ in scope and size, there are some key roles that must be filled for an ISIS development project. Figure 5-17 presents the key roles participated in the ISIS development projects.

![Figure 5-17: The key roles of the ISIS team.](image-url)
**IS Executive.**
IS executive is usually the role, who attains the initial funding for the ISIS development project. This role is responsible for articulating the value of building an ISIS, convincing the relevant departments and upper management that the ISIS project can help the company to achieve its major business and strategic objectives. Moreover, this role is also responsible to define the purposes and scopes of each development projects (such as IS components and responsibility zones involved in each project).

**ISIS User.**
An ISIS user can be a role who manages the IS development project (such as Project leader or Team leaders) or participates in the IS development project (such as Business Analyst, System Analyst, Developers, Testers, etc). An ISIS user can use the ISIS to store or integrate his own specifications in his private workspace. This role also share (i.e. publish) or integrate the common specifications in the common workspace in the ISIS.

**ISIS Project leader.**
The project manager is responsible for planning the ISIS project phases, configuring and assembling the staff. Besides, the project manager should be capable of working with external partners, both software tools and consulting services. In addition, the project manager should be able to propose the development strategy and to define coordination procedures for IS development teams to follow.

**Methodologist.**
The methodologist is responsible for designing information resources and activities of IS development process. At first, the methodologist identifies the necessary categories of specifications used in IS development process. Secondly, the methodologist also identifies the information resources, which are constructed by specifications, used in IS development process. Finally, the methodologist specifies the IS development process, in particular the development activities that consumed and produces those information resources.

Consequently, the methodologist should have adequate competences about *Software process* [ISO 9000-3; ISO/IEC 12207; Kruchten 00; RUP 03; UMM 03; Paulk 93] and *Method engineering* [Brinkkemper 96; Ralýté 01, 04].

**ISIS Team leader.**
The team leader is responsible for the technical architecture that the ISIS is based upon. This role is also responsible for defining and enforcing the standards that the ISIS contributors must follow during development process. Furthermore, the team leader must also have a deep understanding of the commercial meta data tools, such as meta data integration and access tools, and understand how each of them pluggs into the overall architecture.
**ISIS Meta-model modeller.**
The meta-model modeller will design and construct the meta-model, which will hold the conceptual and technical specifications. Even if an enterprise has selected its own meta-model, the meta-model modeller will analyze and extend this meta-model. The enhancement and adaptation of the meta-model are to ensure that it will meet the specific requirements proposed by the methodologist as well as adapted to the organization structure, development process and development environment of the enterprise.

The ISIS Meta-model modeller should have competences in *Meta data models and standards* such as [Bernstein 99; MDIS 97; CWM 03; MOF 02; XMI 00].

**ISIS Specification analyst:** the ISIS specification analyst is responsible for specifying the structure, behaviour and coherence of specifications. Furthermore, the specification analyst also needs to understand the database and programming technologies for specifying the references between conceptual and technical specifications.

Nevertheless, the ISIS Specification analyst should have competences in *Requirement engineering* [Rolland 94, 01; Salinesi 03] as well as *Standard framework for IS development* [Frankel 03].

**ISIS Specification integrator.**
The specification integration developer is responsible for extracting the specifications from their sources (i.e. development tools), programmatically integrating them, and then loading them into the ISIS; or vice versa, extracting the specifications from the ISIS, transforming them, and loading them into their sources depending on the integration modes.

**ISIS Service integrator.**
The service integration developer is responsible for using middleware to link multiple and diverse platforms. The middleware can be service-based or message-based middleware, which can be used to coordinate between development tools and the ISIS.

**ISIS Specification coordinator.**
The coordinator's primary responsibility is to meet with the ISIS users to define the necessary supports of the ISIS. This role is also responsible for negotiating and defining overlap situations between development teams and proposing overlap protocols to manage them. The specification coordinator must also be able to translate the requirements into technical solutions that the specification analyst, specification and service integrators can use to guide them during the implementation of the ISIS.

**ISIS Administrator.**
The responsibilities of the ISIS Administrator are to validate specifications stored in the ISIS, to verify the coherences between the objects in the ISIS and objects in
development tools, to manage roles and responsibilities of development teams and their contributors, and to perform administrative services such as security, performance management, etc.

5.4.2.4. **ISIS development process**

**Development process.**
This section illustrates the overview of the development process for the ISIS. In general, there are two strategies for ISIS development:

- *Waterfall development:* this strategy, which is also referred as *big bang*, intends to build a fully functional ISIS in one development effort. This strategy requires each step completing before moving on the next step.
- *Iterative development:* this strategy, which is also referred as *Spiral*, development ISIS functionality in incremental releases. At each iteration of the spiral, a decision about the next objectives has been made for the next iteration.

As discussed before, the iterative process is preferred because the major risks are addressed can be addressed early.

**Development phases.**
A typical ISIS development project should include the following phases: Analysis, Design, Implement and Deployment phases.

Let’s focus on the Analysis and Design phases, which are the main objectives of this thesis. Table 5-2 explains in details the Analysis and Design phases, which are the main focus of the thesis.

<table>
<thead>
<tr>
<th>Phase</th>
<th>Task</th>
<th>Explication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Preparation</td>
<td>Ensuring that the executives in the enterprise understand the important of specifications, and the added value of the ISIS.</td>
</tr>
<tr>
<td>Scope definition</td>
<td>Defining the objectives of the ISIS.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Identifying the specification categories, information resources and development activities.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Selecting development strategies.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Assigning the development teams, and development tools to the ISIS project.</td>
<td></td>
</tr>
<tr>
<td>Feasibility</td>
<td>Discovering the cost benefits of the enterprise to implement the ISIS and stating what the key factors will be important for determining the success of the ISIS project.</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>ISIS architecture</td>
<td>Determining the overall architecture for the ISIS. This task may include the evaluation and selection commercial meta data integration tools, which can be used to implement the ISIS.</td>
</tr>
<tr>
<td></td>
<td>ISIS meta-model</td>
<td>Building the meta-model to represent the structure, the behaviour and the coherence of specifications.</td>
</tr>
</tbody>
</table>
Table 5-1: Main tasks of the Analysis and Design phases.

**Development responsibilities.**
Continuously, Table 5-2 presents the tasks and involved human resources for the Analysis and Design phases.

<table>
<thead>
<tr>
<th>Phases</th>
<th>Task name</th>
<th>ISIS Team</th>
<th>IS development teams</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analysis</td>
<td>Preparation</td>
<td>ISIS Project Leader</td>
<td>Executives, Project leader</td>
</tr>
<tr>
<td></td>
<td>Project Scope</td>
<td>ISIS Project leader, Methodologist</td>
<td>Project leader</td>
</tr>
<tr>
<td></td>
<td>Feasibility</td>
<td>ISIS Project leader</td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td>Architecture</td>
<td>ISIS Team Leader</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Meta model</td>
<td>ISIS Meta-model modeller</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Supports</td>
<td>ISIS Specification Coordinator, ISIS Specification Analyst, ISIS Administrator</td>
<td>Team Leader, Business analyst, System analyst</td>
</tr>
</tbody>
</table>

Table 5-2: The tasks and involved human resources for the Analysis and Design phases.

**5.5. Conclusion**

This chapter presents the principles of the *Information System upon Information Systems* (ISIS) and how it manages specifications of information systems.

An ISIS is a typical IS that supports information systems in development. Therefore, the main facilities of an ISIS are to identify, manage and coordinate specifications used in IS development.

Management of specification concerns with the enterprise of IS development process, in particular the development activities that consumed and produced information resources. An information resource is constructed by a set of specifications. Specification management also concerns with the responsibility assignments conforming to information resources used.

In general, an ISIS includes three layers: i) Representation layer for representing, validating and managing specifications; ii) Integration layer for integrating specifications and services of the ISIS with specifications and services of development tools; and iii) Coordination layer for coordinating specifications shared by development teams.
Moreover, the ISIS can integrate their specifications with the specifications stored in development tools such as CASE tools, Meta data repository tools, IDE tools, DBMSs and SCM tools. The possible integration modes are Direct integration, Replication integration, Agent integration, Registration integration, and Loose integration modes.

Finally, in order to build and manage an ISIS, there is a need for an ISIS team, including the following roles: IS Executive, ISIS User, ISIS Project leader, Methodologist, ISIS Team leader, ISIS Meta-model modeller, ISIS Specification analyst, ISIS Specification integrator, ISIS Service integrator, ISIS Specification coordinator, and ISIS Administrator.
Chapter 6.

SPECIFICATION REPRESENTATION

This chapter aims at presenting how to represent specifications in an ISIS. The structure of this chapter is as follows. Section 6.1 presents the principles of the Representation layer of the ISIS. Section 6.2 deals with the structure of conceptual specifications. Section 6.3 concerns with the behaviour of conceptual specifications. Section 6.4 discusses about the coherence of conceptual specifications.

6.1. Principles of the Representation layer

The purpose of the Representation layer is to represent, validate and manage the specifications stored as objects of an ISIS.

Specifications are defined as all physical data and knowledge consumed and produced by activities of IS development process. Specifications represent information about the structure, the behaviour and the coherence of business data used by an enterprise.

There are two types of specifications: Informational and technical specifications. 
Conceptual specifications provide information about information at conceptual level. Technical specifications provide information about information at the implementation level.

6.1.1. Objects of an ISIS

In the ISIS, an object represents a specification. There are two types of objects: Private object and Common object.

Private objects represents specifications of an IS component and are interested by a corresponding responsibility zone of a development team. Common objects are overlapped between IS components and above the interests of a responsibility zone.

The definition above implies a distinction between common objects and private objects. Private object is interested by a development team and can have an owner. Meanwhile, a common object can be interested by several development teams. A common object may have an owner, co-owners, or a custodian.
6.1.2. Private and Common workspaces

The Representation layer has two workspaces: Private workspace and Common workspace. The Private workspace contains the private objects of IS components. Each IS component has its own workspace. The Common workspace contains the common objects shared by several IS components.

A common object is concerned by an overlap situation between several IS components. Each overlap situation employs an overlap protocol.

Depending the selected overlap protocol, IS components concerned with an overlap situation can play different roles:

- Concerning the Ownership-based overlap protocol, there is a RZ of an IS component to act as the owner. The owner of a common object has unlimited rights over the object, including the ability the change the structure of the object without notifying anybody else.
- Concerning the Service-based overlap protocol, there is a RZ to act as the custodian (or the provider). The custodian of a common object has certain responsibilities towards other users of the object.
- Concerning the Watch-based overlap protocol, there are several co-owners of a common object. Each co-owner of the common object has the capacity to watch the consequences when the others perform an operation over this object.

6.1.3. Interpretation of the Representation layer

The Representation layer stores objects in its repository. The repository contains the Private workspace and Common workspace.

Figure 6-1 presents the interpretation of the Representation layer.

Figure 6-1: Example about the interpretation of the Representation layer.
In the Private workspace, each IS component has its own workspace. For example, there are workspace for the “A” IS component (contains the \(a1, a2, \) and \(a3\) private objects) and workspace for the “B” IS component (containing the \(b1, b2, b3\) private objects).

The Common workspace contains the \(x1\) and \(x2\) common objects. The \(x1\) object is in the overlap situation between the “A” and “C” IS components that employs the Ownership-based overlap protocol. The owner of the \(x1\) object is the RZ of the “C” IS component, the RZ of the “A” IS component is a user of this object. Moreover, the \(x1\) object is referenced by the two objects: the \(a3\) private object in the workspace of the “A” IS component and the \(c1\) object in a development tool used by the RZ of the “C” IS component.

On the other hand, the \(x2\) object is in the overlap situation between the “A” and “B” IS components that employs the Watch-based overlap protocol. Both RZs of the “A” and “B” IS components are the co-owners of this object. Consequently, the \(x2\) object is referenced by the two objects: the \(a2\) private object in the workspace of the “A” IS component and the \(b3\) private object in the workspace of the “B” IS component.

6.2. Structure of conceptual specifications

In an ISIS, a category of conceptual specifications, corresponding to an IS concept, is represented by a hyperclass of the ISIS. Correspondingly, a conceptual specification is represented by a hyper-object of the ISIS.

The hyperclasses of the ISIS, which represent different categories of conceptual specifications, are called generic hyperclasses.

Generic hyperclasses of ISIS can be classified according to the aspect to which they are belonged (Figure 6-2) such as the followings:

- **Atomic-class, Hyperclass, Attribute, and Key** generic hyperclasses (the Static aspect);
- **Event, Process, Dynamic state, and Method** generic hyperclasses (the Dynamic aspect);
- **Integrity rule, Scope, and Risk** generic hyperclasses (the Integrity rule aspect).
To represent the structure of generic hyperclasses, we can use Class diagram and Object diagram as mentioned in Chapter 3.

Besides, the complete description of all the tuple-classes and atomic-classes of an ISIS can be found in Annex 2, which represents the meta-model of the Static, Dynamic, and Integrity rule aspects of the M7 reference model.

6.2.1. Structure of specifications of the static concepts

Atomic-class specification.

Atomic-class hyperclass is a generic hyperclass of an ISIS that represents specifications of atomic-classes of IS.

This hyperclass has the Atomic-class tuple-class as its key class (Figure 6-3). The Atomic-class tuple-class has three attributes: Atomic-class Base type, Field size and Decimal.

The Atomic-class tuple-class is a specialisation of the Category tuple-class. Therefore, objects of Atomic-class tuple-class are also objects of Category tuple-class.
Hyperclass specification.

Hyperclass hyperclass is a generic hyperclass of an ISIS that represents the specifications of hyperclasses of IS.

Hyperclass hyperclass is defined over its key class: the Hyperclass tuple-class (Figure 6-4). Similar to Atomic-class tuple-class, objects of the Hyperclass tuple-class are objects of the Category tuple-class. In the same manner, objects of the Sub-hyperclass and Tuple-class tuple-classes are also objects of the Hyperclass tuple-class and therefore objects of the Category tuple-class.

From an object of the Hyperclass, one can navigate to objects of the Sub-Hyperclass tuple-class (using its sub-hyperclass-of attribute), a set of objects of the Attribute tuple-class (using the origin-Hyperclass attribute), and a set of objects of the Key tuple-class (using the key-of-Hyperclass attribute).

Attribute specification.

Attribute hyperclass is a generic hyperclass of an ISIS, which represents the specifications of attributes of IS.

Attribute hyperclass has the Attribute tuple-class is key class (Figure 6-5).

Attribute tuple-class has the attributes such as Attribute OID, Attribute Name, Origin hyperclass and Termination category.

From an object of the Attribute tuple-class, one can navigate to an object of Hyperclass tuple-class (using its origin-Hyperclass attribute) and an object of Category tuple-class (using its termination-Category attribute).

Key specification.

Key hyperclass is a generic hyperclass of an ISIS that represents the specifications of keys of IS.
**Key** hyperclass has the **Key** tuple-class is key class (Figure 6-6).

**Key** tuple-class has the attributes such as **Key OID**, and **Key Name**.

From an object of the **Key** tuple-class, one can navigate to an object of **Hyperclass** tuple-class (using its **key-of-Hyperclass** attribute), and a set of objects of **Attributes** tuple-class (using its **key-Attributes** attribute).

![Figure 6-6: Structure of Key specification.](image)

### 6.2.2. Structure of specifications of the dynamic concepts

**Dynamic-state specification.**

**Dynamic-state** hyperclass is a generic hyperclass of an ISIS that represents the specifications of dynamic states of IS.

**Dynamic-state** hyperclass has the **Dynamic-state** tuple-class as its key class (Figure 6-7).

Objects of the **Dynamic state** tuple-class are objects of the **Sub-hyperclass** tuple-class, and are objects of the **Hyperclass** tuple-class and are objects of the **Category** tuple-class. 

![Figure 6-7: Structure of Dynamic-state specification.](image)

**Method specification.**

**Method** hyperclass is a generic hyperclass of an ISIS that represents the specifications of methods of IS.
Method hyperclass has the Method tuple-class as its key class (Figure 6-8).

Method tuple-class has the Method OID and Method name attributes.

From an object of the Method tuple-class, one can navigate to an object of the Hyperclass tuple-class (using its method-of attribute), a set of objects of the Dynamic state tuple-class (using its From-States attribute), and another set of objects of the Dynamic state tuple-class (using its To-States attribute).

Event specification.
Event hyperclass is a generic hyperclass of an ISIS that represents the specifications of events of IS.

Event hyperclass is actually the Event tuple-class (Figure 6-9).

Key tuple-class has the attributes such as Event OID, and Event Name.

Process specification.
Process hyperclass is a generic hyperclass of an ISIS that represents the specifications of keys of IS.

Process hyperclass has the Process tuple-class as its key class (Figure 6-10). Process tuple-class has the attributes such as Process OID, Process name, Pre-condition and Post-condition.

From an object of the Process tuple-class, one can navigate to an object of Event tuple-class (using its Triggered-by attribute), an object of Hyperclass tuple-class (using its Process-of-hyperclass attribute), a set of objects of Hyperclass tuple-class (using its Pre-hyperclasses attribute), another set of objects of Hyperclass tuple-class (using its Post-hyperclasses attribute), and a set of objects...
of Method hyperclass (using its Execute-methods attribute).

6.2.3. Structure of specifications of the integrity rule concepts

Integrity rule specification.

*Integrity rule* hyperclass is a generic hyperclass of an ISIS that represents the specifications of integrity rules of IS.

*Integrity rule* hyperclass is actually the *Integrity rule* tuple-class (Figure 6-11).

*Integrity rule* tuple-class has the attributes such as *Rule OID*, *Rule name*, and *Rule expression*.

From an object of the *Integrity rule* tuple-class, one can navigate to a set of objects of the *Scope* tuple-class (using *Scope-of-Rule* attribute), and a set of objects of the *Risk* tuple-class (using the *Risk-of- Scope* attribute).

Scope specification.

*Scope* hyperclass is a generic hyperclass of an ISIS that represents the specifications of scopes of integrity rules of IS.

*Scope* hyperclass has the *Scope* tuple-class as its key class (Figure 6-10).

*Scope* tuple-class has one attribute: *Scope OID*.

From an object of the *Scope* tuple-class, one can navigate to an object of *Integrity rule* tuple-class (using its *Scope of Rule* attribute), and an object of *Tuple-class* tuple-class (using its *on tuple-class* attribute), and a set of objects of *Risk* tuple-class (using the *Risk-of- Scope* attribute).
**Risk specification.**  
*Risk* hyperclass is a generic hyperclass of an ISIS that represents the specifications of risks of scopes of integrity rules of IS.

*Risk* hyperclass has the *Risk* tuple-class is its key class (Figure 6-10).  
*Risk* tuple-class has one attribute: *Risk OID*.  
From an object of the *Risk* tuple-class, one can navigate to an object of the *Scope* tuple-class (using its *Risk-of-Scope* attribute), an object of the *Integrity rule* tuple-class (using its *Scope-of-Rule* attribute), an object of the *Primitive* tuple-class (using its *defined-on-Primitive* attribute), an object of the *Attribute* tuple-class (using its *Risk-On-Attribute* attribute), and a set of objects of the *Method* tuple-classes (using its *concerned-Methods* attributes).

![Figure 6-13: Structure of Risk specification.](image)

**6.2.4. An example about specification representation**

Let’s continue with our previous example considering the specification of *Customer* hyperclass of the IS for MATIS enterprise (in short: *MATIS-IS*).

This example illustrates how the ISIS stores its objects by presenting the object diagram for representation of the specification of the *Customer* tuple-class and the specifications of the tuple-classes, sub-hyperclass, attributes, and keys of this tuple-class.

Figure 6-14 clarifies the object level of the *Category, Atomic-class, Hyperclass, Tuple-class, Sub-hyperclass, Attribute*, and *Key* tuple-classes of the ISIS concerning the object diagram of the *Customer* object.

The *Customer* hyperclass of the MATIS-IS is represented by an object of the *Hyperclass* tuple-class of the ISIS. Its object diagram is presented in the next figure.
Let’s begin with the Customer object of the Hyperclass tuple-class.

This Customer object of the Hyperclass tuple-class is linked to the Customer object of the Tuple-class tuple-class by the “is-A-Tuple-class” link. It means that Customer hyperclass is also a tuple-class.

It is also linked to the Customer-Activated object of the Sub-hyperclass tuple-class by the “is-A-Sub-hyperclass” link. It means that the Customer hyperclass of the MATIS-IS is a tuple-class and has a sub-hyperclass named Customer-Activated.

Moreover, there are the objects of the Attribute tuple-class that have the “origin-Hyperclass” link to the Customer object of the Hyperclass tuple-class. Those objects represent the attributes of the Customer hyperclass of the MATIS-IS such as CustomerCode, CustomerName, CustomerAddress, CustomerTelephone and ActivatedTime.

In the same way, there is the CustomerKey object of the Key tuple-class has the “key-of-Hyperclass” link to the Customer object of the Hyperclass tuple-class. The link indicates that the Customer hyperclass of the MATIS-IS has a key named CustomerKey.
6.3. Behaviour of conceptual specifications

This section discusses about the local behaviour of a conceptual specification. In fact, the global behaviour of the ISIS is belonged to the organization structure and IS process of the enterprise. Therefore, the global behaviour is out of the scope of the conceptual framework proposed by this thesis.

The section begins with the definition of conformity rules and generic dynamic states of a conceptual specification. Thus, the section continues with the generic methods, and the object life cycle of generic hyperclasses of the ISIS. Finally, the section ends with the coordination between object life cycles of generic hyperclasses.

6.3.1. Conformity rules

An object of a generic hyperclass is said: “to be conformed” if it is satisfied all the conformity rules. In our approach, there are two types of conformity rules: validity and completeness rules [Pham 02].

The concept of validity rule is actually inherent to the concept of “integrity rule” at the meta-model level. They are the set of rules coordinating with the meta-model to control the validity of every object of generic hyperclasses.

When designer modifies an object of a generic hyperclass, this modification may violate the validity rules, which are concerned about this object. If the modification violates one of those rules, the object is brought into the Invalid state. On the contrary, it is in the Valid state. In general, the validity rules can be validated automatically.

For instance, there is a validity rule related to objects of the Hyperclass generic hyperclass such as: “The dependent constituents of a hyperclass such as attributes, keys, and sub-classes must be valid”.

The concept of completeness rule is related to the perception about the enterprise and the real world that designers have to realize. In fact, the decision of designers about the completeness of a conceptual specification depends on the finish of works and the stability of the constituents of the real world, which are modelled with that specification. Some of completeness rules cannot be validated automatically. Therefore, designers who are responsible for managing completeness rules will decide their completeness status.

For example, “a hyperclass must have all its attributes” is a completeness rule. In this case, the designer, who decides whether all the necessary attributes of that tuple class are already specified or not, will specify its completeness status.
6.3.2. **Generic dynamic states**

A generic dynamic state is a dynamic state that is common for all the object life cycles of conceptual specifications. In other words, these dynamic states exist in all the object life cycles of generic hyperclasses.

In the ISIS approach, we propose the following dynamic states: *Ready to initialize*, *Initialized*, *Valid*, *Invalid*, *Completed*, *Uncompleted*, *Implemented*, and *Unimplemented* (Figure 6-15).

Firstly, every object of a generic hyperclass, which represents a conceptual specification, has two dynamic states: *Ready to initialize* before its initiation and *Initialized* after its initiation.

Secondly, when an object is initialized, it is said: “to be conformed” if it is satisfied all the conformity rules, including validity and completeness rules.

Therefore, the next dynamic states are proposed:

i) **Valid** / **Invalid** to state that an object is satisfied /dissatisfied all the concerned validity rules;

ii) **Completed** / **Uncompleted** to state that an object has completely / uncompleted specified its constituents.

Finally, the dynamic states to indicate that an object of a generic hyperclass is implemented (or not) are also necessary. Consequently, other two generic dynamic states: *Implemented* and *Unimplemented* are also proposed.

In an object is in the *Implemented* dynamic state, it means that there are technical specifications referred to the conceptual specification that it represents.

![Generic dynamic state diagram](image)

*Figure 6-15: Generic dynamic states.*

6.3.2.1. **Generic object life-cycle**

In the ISIS approach, the development process of a conceptual specification can be represented by a generic object life cycle of the corresponding generic hyperclass of the ISIS.
Figure 6-16: A generic object life cycle of a generic hyperclass of the ISIS.

Figure 6-16 introduces the **generic life cycle** of an object of a generic hyperclass.

Each object of a generic hyperclass may have the generic dynamic states as mentioned above such as *Ready to initialize, Initialized, Invalid, Valid, Completed, Unimplemented, and Implemented.*

A method that may change the generic dynamic states of an object is called a **generic method** (Figure 6-16).

An object before its initiation is in the *Ready to initialize* state. After its initiation (using *Initialize()* method), the object is in the *Initialized, Invalid, Uncompleted, and Unimplemented* dynamic states. Consequently, when an object is in the *Initialized* dynamic state, it is ready for processing by other methods such as *Query(), Specify(), Create_dependent(), Delete_dependent(), Uncomplete-Implement(), and Complete-Implement().*

A set of *Specify()* generic methods can be used to bring an object from the *Invalid* or *Valid* states into the *Valid* or *Invalid* states.

Accordingly, a set of *Create_dependent() generic methods can be used to bring an object from the *Uncompleted* state into the *Completed* or *Uncompleted* dynamic states. On the other hand, the *Delete_dependent()* generic method may bring an object from *Uncompleted* or *Completed* dynamic states return to the *Uncompleted* state (Table 6-1).

An object can be implemented when it is in *Valid* state. When an object is in the *Unimplemented* state, there are two possibilities:

- This object is valid but has not yet completed, however, the responsible person decides to implement it. In that case, an *Uncompleted implement()*
method change the object states from Valid, Uncompleted, and Unimplemented states into Implemented;

- This object is Valid and Completed, and then a Completed implement() method changes the object states from Valid, Completed, Implemented states into Implemented.

Moreover, when an object is in the Implemented state, there are two possibilities:

- This object now becomes Completed, therefore a Completed implement() method can be executed. This method does not change its states;
- There is a need for an evolution. In that case, the Evolve() method can perform the evolution primitives. This type of methods does not change the state of object.

Finally, an object in the Initialized and Implemented states can be finalized by the Finalize () method. This method brings the object states return to the Ready to initialize state.

<table>
<thead>
<tr>
<th>Generic method</th>
<th>Result</th>
<th>From generic dynamic states</th>
<th>To generic dynamic states</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize ()</td>
<td>Initialize an object</td>
<td>Ready to initialize</td>
<td>Initialized</td>
</tr>
<tr>
<td>Query ()</td>
<td>Display the values of</td>
<td>Initialized</td>
<td>Initialized</td>
</tr>
<tr>
<td></td>
<td>attributes of an object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Specify ()</td>
<td>Specify an attribute of an</td>
<td>Initialized</td>
<td>Invalid, Valid</td>
</tr>
<tr>
<td></td>
<td>object</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create_dependent()</td>
<td>Create an object of a</td>
<td>Initialized</td>
<td>Uncompleted, Completed</td>
</tr>
<tr>
<td></td>
<td>dependent generic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hyperclass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Delete_dependent()</td>
<td>Delete an object of a</td>
<td>Initialized</td>
<td>Uncompleted</td>
</tr>
<tr>
<td></td>
<td>dependent generic</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hyperclass</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncompleted-Implement ()</td>
<td>Implement an object when</td>
<td>Valid, Uncompleted,</td>
<td>Implemented</td>
</tr>
<tr>
<td></td>
<td>it is in the Uncompleted</td>
<td>Initialized</td>
<td></td>
</tr>
<tr>
<td></td>
<td>state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completed - Implement ()</td>
<td>Implement an object when</td>
<td>Valid, Completed,</td>
<td>Valid, Completed,</td>
</tr>
<tr>
<td></td>
<td>it is in the Completed</td>
<td>Initialized</td>
<td>Implemented</td>
</tr>
<tr>
<td></td>
<td>state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evolve ()</td>
<td>Evolve an object when it</td>
<td>Implemented</td>
<td>Implemented</td>
</tr>
<tr>
<td></td>
<td>is completely Implemented</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Finalize ()</td>
<td>Finalize an object</td>
<td>Initialized, Implemented</td>
<td>Ready to initialize</td>
</tr>
</tbody>
</table>

Table 6-1: Generic methods of a generic hyperclass of the ISIS.

Notes: It is possible that an object does not change its states after executing a method (i.e. the set of dynamic states before execution and after execution of the method is the same). In that case, the object is changed into temporary states and then returns to its previous states.
For example, the Query() method (Figure 6-16) can be used to access the information concerning an object of a generic hyperclass. During executing this method, this object is in the “being queried” temporary state. This temporary state is not necessary to be persistent and therefore not presented in the local life cycle.

6.3.3. Adaptation of a generic object life cycle

For a particular context, the generic object life cycle can be adapted to a specific object life cycle of a generic hyperclass of ISIS.

The adaptation of an object life cycle is based on the adaptation of its methods. In general, the adaptation of generic methods is the transformation the generic methods into the specific methods of a generic hyperclass of ISIS.

Example 6-1: The object life cycle of the Hyperclass generic hyperclass.

This example concerns with the adaptation of the generic object life cycle of the Hyperclass generic hyperclass, an important generic hyperclass of the ISIS.

At first, the Specify() generic method can be refined into a set of methods according to the attributes of the Hyperclass generic hyperclass such as Specify-Name(), Specify-SuperHyperclass(), SpecifyIsATupleclass() and Specify-SpecialisationCondition().

Moreover, the Create-dependent() generic method is refined into the following methods: Create-SubHyperclass(), Create-Attribute(), and Create-key(). Accordingly, the Delete-dependent() generic method is refined into the Delete-SubHyperclass(), Delete-Attribute() and Delete-Key() methods.

Table 6-2 proposes the list of specific methods of the Hyperclass generic tuple-class.

<table>
<thead>
<tr>
<th>Primitive method</th>
<th>Specific method</th>
<th>Corresponding generic method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initialize a hyperclass</td>
<td>Initialize-Hyperclass()</td>
<td>Initialize()</td>
</tr>
<tr>
<td>Display a hyperclass</td>
<td>Query-Hyperclass()</td>
<td>Query()</td>
</tr>
<tr>
<td>Specify a name of a hyperclass</td>
<td>Specify-Name()</td>
<td>Specify()</td>
</tr>
<tr>
<td>Specify a super-hyperclass of a hyperclass</td>
<td>Specify-SuperHyperclass()</td>
<td>Specify()</td>
</tr>
<tr>
<td>Specify a specialization condition of a sub-hyperclass</td>
<td>Specify-SpecialisationCondition()</td>
<td>Specify()</td>
</tr>
<tr>
<td>Indicate whether the hyperclass is a tuple-class or not</td>
<td>Specify-IsATupleclass()</td>
<td>Specify()</td>
</tr>
<tr>
<td>Create a sub-hyperclass of a hyperclass</td>
<td>Create-SubHyperclass()</td>
<td>Create-dependent()</td>
</tr>
<tr>
<td>Create an attribute of a hyperclass</td>
<td>Create-Attribute()</td>
<td>Create-dependent()</td>
</tr>
<tr>
<td>Create a key of a hyperclass</td>
<td>Create-Key()</td>
<td>Create-dependent()</td>
</tr>
<tr>
<td>Delete a sub-hyperclass of a hyperclass</td>
<td>Delete-SubHyperclass()</td>
<td>Delete-dependent()</td>
</tr>
<tr>
<td>Delete an attribute of a hyperclass</td>
<td>Delete-Attribute()</td>
<td>Delete-dependent()</td>
</tr>
</tbody>
</table>
Delete a key of a hyperclass | Delete-Key() | Delete-dependent()
Implement a hyperclass | ImplementHyperclass() | Uncompleted-implement()
Re-Implement a hyperclass | Re-Implement-Hyperclass() | Completed-implement()
Finalize a hyperclass | Finalize-Hyperclass() | Finalize()

**Table 6-2: Specific methods of the Hyperclass generic hyperclass of the ISIS**

**Notes:** The Evolve() generic method, concerning with the evolution of specifications, is out of the scope of this thesis. Therefore, it is not discussed in detail in this example. More information about Evolve() method can be found in [Pham 02].

In summary, the object life cycle of the Hyperclass generic hyperclass is presented in Figure 6-17.

**Figure 6-17: Specific object life cycle of the Hyperclass generic hyperclass.**

### 6.4. Coherence of conceptual specifications

The coherence of conceptual specifications is guaranteed by integrity rules of the ISIS. For this reason, this section focuses on identifying, specifying and enforcing integrity rules of the ISIS.
6.4.1. Identifying integrity rules

In the ISIS, there are two main categories of integrity rules (IR): i) IRs originating from validity rules; and ii) IRs originating from completeness rules.

**IRs originating from validity rules.**
The validity rules, which are actually the “integrity rule” at the meta-model level, control the validity of every object of generic hyperclasses represented in this meta-model.

Every modification of an object of a generic hyperclasses (representing a conceptual specification) may violate the validity rules concerning this object. If the modification violates one of those rules, the object is brought into the *Invalid* state. On the contrary, it is in the *Valid* state.

Table 6-3 presents some integrity rules originating from the validity rules related to a *Hyperclass* generic hyperclass (and its dependent generic hyperclasses). The complete list of integrity rules can be found in Annex 2.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Generic hyperclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_V#1</td>
<td>Hyperclass</td>
<td>The Hyperclass-Name of a hyperclass must be a valid name (e.g. no special characters)</td>
</tr>
<tr>
<td>S_V#2</td>
<td>Sub-hyperclass</td>
<td>The Specialisation condition of a sub-hyperclass must be valid (e.g. a valid Boolean expression).</td>
</tr>
<tr>
<td>S_V#3</td>
<td>Sub-hyperclass</td>
<td>A hyperclass can be a sub-hyperclass of one and only one hyperclass.</td>
</tr>
<tr>
<td>S_V#4</td>
<td>Sub-hyperclass</td>
<td>Sub-hyperclass must be different to its super-hyperclass.</td>
</tr>
</tbody>
</table>

*Table 6-3: Integrity rules originating from the validity rules related to the Hyperclass generic hyperclass.*

**IRs originating from completeness rules.**
The completeness rules are related to the perception about the real world that designers have to realize. An object of a generic hyperclass, which is constructing, is in *Uncompleted* state. When the object is finished, its dynamic state will change to *Completed* state.

In general, the completeness of a specification is belonged to the decision of designers about the completeness of works. For this reason, some of completeness rules cannot be validated automatically. Indeed, designers who are responsible for managing these completeness rules will decide their completeness status.

Table 6-4 gives an example about the integrity rules originating from the completeness rules related to the *Hyperclass* generic hyperclass.
<table>
<thead>
<tr>
<th>Rule</th>
<th>Generic hyperclass</th>
<th>Description</th>
<th>Validated automatically?</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_C#1</td>
<td>Hyperclass</td>
<td>Each hyperclass must have at least one key.</td>
<td>Yes</td>
</tr>
<tr>
<td>S_C#2</td>
<td>Hyperclass</td>
<td>Each hyperclass must have all “required” attributes, keys and sub-hyperclasses.</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 6-4: Integrity rules originating from the completeness rules related to Hyperclass generic hyperclass.

### 6.4.2. Specifying integrity rules

An integrity rule of an ISIS actually is a logical condition defined over tuple-classes that can be specified formally and verified by processes or methods.

**Example 6-2: Specifications of some typical IRs.**

Table 6-5 presents the specification of some typical integrity rules such as the \( S_V\#1 \) and \( S_C\#1 \).

<table>
<thead>
<tr>
<th>Integrity Rule</th>
<th>Expression</th>
<th>Scopes</th>
<th>Risks</th>
</tr>
</thead>
<tbody>
<tr>
<td>( S_V#1 )</td>
<td>The Hyperclass-Name of a hyperclass must have a valid name.</td>
<td>Hyperclass</td>
<td>- Initializing a new hyperclass. - Modifying the Hyperclass Name attribute. - Changing the dynamic state of a hyperclass into Valid.</td>
</tr>
<tr>
<td>( S_C#1 )</td>
<td>If a hyperclass is completed, then it must have at least one key.</td>
<td>Hyperclass, Key</td>
<td>- Deleting a key of a hyperclass. - Changing the dynamic state of a hyperclass into Completed.</td>
</tr>
</tbody>
</table>

Table 6-5: Specifications of some typical integrity rules.

### 6.4.3. Enforcing integrity rules

In an ISIS, enforcing the IRs is based on the periods to which they are belonged. A period is determined by a set of integrity rules that must be enforced and a series of events that are authorized in that period [Estier 91; LeDinh 95].

In the context of the ISIS framework, a period can be considered as a set of enforced IRs and a set of authorized primitive methods. Indeed, there are the generic periods for enforcing the IRs in an ISIS such as:

- **Initiation**: an object of a generic hyperclass is initialized and not specified yet.
- **Conception**: the object is specified and modified. The verification of all the related IRs is not obligatory.
• **Pre Uncompleted-implementation**: the object is specified and modified, and then become **Valid** but not **Completed**. However, the designer has decided to implement it.

• **Pre Completed-implementation**: the designer continues to specify and modify the object until it is become **Valid and Completed**. Afterwards, it is ready to be (re) implemented.

• **Exploitation**: the implementation of the object is ready to be exploited and the modelling phase is finished. However, the evolution of the object (i.e. its structure and behaviour) is still available.

**Example 6-3: The periods for Hyperclass generic hyperclass.**

Table 6-6 presents the periods for the life cycle of an object of the Hyperclass generic hyperclass.

A specification of a hyperclass is in the **Pre Uncompleted-Implementation** period when the designer has specified its principal constituents. He would like to implement it so that other developers can use its implementation.

Later, this specification changes to **Pre completed-Implementation** period when the designer specifies all the missing constituents, decides that it is completed, and re-implement again.

<table>
<thead>
<tr>
<th>Period</th>
<th>Enforced integrity rules</th>
<th>Authorized methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiation</td>
<td></td>
<td>- Initialize, Query, and Finalize a Hyperclass</td>
</tr>
<tr>
<td>Conception</td>
<td></td>
<td>- Query, Finalize a Hyperclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specify a Name, SuperHyperclass,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SpecialisationCondition IsATupleClass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete a Sub-hyperclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete an Attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete a Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implement a Hyperclass</td>
</tr>
<tr>
<td>Pre Uncompleted-Implementation</td>
<td>S_V#1, S_V#2</td>
<td>- Query, Finalize a Hyperclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specify a Name, SuperHyperclass,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SpecialisationCondition IsATupleClass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete a Sub-hyperclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete an Attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete a Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implement a Hyperclass</td>
</tr>
<tr>
<td>Pre Completed Implementation</td>
<td>S_V#1, S_V#2 S_C#1</td>
<td>- Query, Finalize a Hyperclass.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specify a Name, SuperHyperclass,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SpecialisationCondition IsATupleClass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete a Sub-hyperclass</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete an Attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete a Key</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implement a Hyperclass</td>
</tr>
<tr>
<td>Exploitation</td>
<td>&lt;All&gt;</td>
<td>Evolve a Hyperclass, Query a Hyperclass.</td>
</tr>
</tbody>
</table>

*Table 6-6: The periods for the Hyperclass generic hyperclass.*
6.5. Conclusion

This chapter presents the principles of the Representation layer of an ISIS and how to represent a conceptual specification.

In an ISIS, a generic hyperclass represents a certain type of conceptual specification. Consequently, an object of a generic hyperclass represents a conceptual specification of information systems.

There are two types of objects: private and common objects. Private objects are stored in Private workspace; meanwhile common objects are stored in Common workspace of the ISIS.

On the other hand, the representation of a conceptual specification includes:

- The structure of each type of conceptual specifications is represented by a hyperclass defined on the meta-model of the M7 reference model.

- The behaviour of a conceptual specification is represented by an object life cycle, including the dynamic states and methods.

- The coherence of a conceptual specification is represented by a set of integrity rules of the ISIS, originating from the validity rules and completeness rules.
Chapter 7.

**SPECIFICATION COORDINATION**

This chapter introduces the general concepts of coordination and theory of coordination, and then proposes the models for coordination of conceptual specifications used in IS development.

The remaining of the chapter is structured as follows. Section 7.1 introduces the overview of information coordination. Section 7.2 continues to discuss about coordination of conceptual specifications in development with the support of an ISIS. Section 7.3 presents the coordination of conceptual specifications belonged to an IS component, called *Intra-IS coordination*. Section 7.4 presents the coordination of conceptual specifications shared by different IS components, called *Inter-IS coordination*.

### 7.1. Information coordination

#### 7.1.1. What is coordination

Coordination is defined as “the art of working together harmoniously” [Malone 94]. The conceptual framework for coordination proposed by Malone and Crowston includes four basic components: *actors* performing *activities* directed towards *goals*, with goal-relevant *interdependencies* between activities.

Consequently, coordination means identifying the interdependencies and determining the responsibilities of each contributor. Interdependencies can be analysed in terms of common objects that are involved in several activities.

An information resource or a specification can be a *common object*. For instance, a conceptual specification, which is communicated from the analysis activity to a design activity, can be a common object.

#### 7.1.2. Theory of coordination

In general, there are three main theories of coordination: *Hierarchical, Market* and *Network theories* [Richard 94].

---

122
Hierarchical theory of coordination.
A hierarchy is constructed by administration, command and control. Each part is exactly defined to perform a specific function. Ideally, each function is effectuated by a single part, without overlaps. Control functions are effectuated by other parts.

Market theory of coordination.
A market is a system of agents, in which an agent can provide products and services to other agents. A market is held together by product and service exchanges based on contracts.

Network theory of coordination.
A network is system that helps its members work together by communication, based on trust.

7.1.3. Information coordination

Entirely integrating of enterprise information resources is a major trend in information technology today and is an area both of intensive theoretical speculation and practical research and development. The need of integrating of enterprise information resources is growing quickly as a consequence of new technologies as well as organizational demands, especially the Web and Enterprise Resource Planning (ERP) packages.

Information coordination is a potential mean to reach entirely integrating of enterprise information assets. The benefits of information coordination are information sharing, reuse and coherence.

In the context of information system engineering, there are two situations of information coordination: coordination in development and coordination in production.

Coordination in development is information coordination within/between IS development projects, which are building or maintaining information systems in parallel. Coordination in development focuses on the information resources sharing between several development projects.

Coordination in exploitation is information coordination within/between information systems, which are in exploitation. Coordination in exploitation focuses on the information sharing between systems.

Information coordination is useful in the following situations: i) Developing and maintaining large-scale and enterprise-wide information systems; ii) Integrating information and services from different tools and systems used in IS development process; and iii) Integrating business activities and support enterprises with the help of information and systems.
7.1.4. Information coordination in development

An enterprise, at a point of time, may undertake several parallel development projects, and may also need to maintain and enhance existing systems. Each development project may have independent project leaders and schedules.

Moreover, there are some mechanisms for coordinating within/between projects. Intra-project coordination is the internal coordination, within the project itself. Inter-project coordination is external coordination, between the project with other projects or systems.

Consequently, there are three techniques that need to be taken into account [Richard 94]:

- Technique of planning projects, especially to reduce the costs and risks of coordination. In the context of new system development project, it is the technique to design new large systems as a set of coordinated units, so that these units can be easily commissioned or decommissioned in development stages. In the context of existing systems maintenance project, it is the technique to break an existing monolithic system into coordinated units that can be replaced at different times without endangering the global integrity.
- Technique for intra-project and inter-project coordination to reduce the costs and risks of projects. Indeed, it is the technique to coordinate information resources from different tools and systems, and to provide a coherent view of these information resources.
- Techniques of planning the coordination activities so that they can be managed and their gains and costs can be balanced.

7.2. Coordination in development supported by an ISIS

This section presents how to organize the development world with IS components and then introduces how to realize Intra–project and Inter-project coordination.

7.2.1. An ISIS supporting coordination in development

Organizing the development world with IS components.

As mentioned in previous chapters, the main purpose of an Information System upon Information Systems (ISIS) is to support the management and coordination of informational resources in development with the focus on the conceptual specifications.

In the ISIS approach, Information System components are the bricks for building and managing information systems. The objective of IS component approach is to enable to work with a part of an IS as a coordinated unit.
The differences between the IS components reflect the real differences between/within the responsibility zones (RZ) that manage and use the information system. Accordingly, they reflect real the differences between/within the projects that design and develop those components. The differences, which represent the interdependences between projects and components, may cause the conflict in development. Therefore, those differences demand coordination, which is the main focus of this chapter.

For this reason, the global IS development project should be decomposed into several development projects corresponding to different IS components, which can be developed in parallel or in sequence.

**Dealing with the differences between IS components.**
Experiments show that conflict in enterprises does not always have to be eliminated. However, it can be managed by a judicious application. An ISIS can play the role of a judicious application to manage the conflict in IS development and exploitation. An ISIS is neutral: it can be used to increase, decrease or preserve the level of complexity and tension in an enterprise.

In the ISIS approach, information system components can be used for development and maintenance projects. They are useful for establishing the requirements as well as designing and implementing information systems. Furthermore, IT executives can use IS components for planning and managing systems and projects.

Our current work focuses on the management and coordination of information system components. It is the reason why an enterprise should have many IS components. By having IS components, IT executives can be able to manage the interdependences within/between enterprises, projects and systems in terms of interdependences between IS components that support them.

Information coordination with IS components is a powerful technique of managing and coordinating information systems, or in other words, of managing the enterprise itself as an information processing system with the support of the ISIS.

### 7.2.2. Intra-IS coordination and Inter-IS coordination

At the analysis level, an IS component is constructed by a unique set of conceptual specifications. Concerning the IS development process, there are regular “bugs”, which are hard to find because the problem isn’t in one responsibility zone. In fact, the problem may happen in the vertical interaction or the horizontal interaction between IS components:

- **Horizontal interaction** is the interaction between specifications of several IS components at the same level of abstraction. For example, data schema integration approaches often address the problem happened at the informational level.
• **Vertical interaction** is the interaction between specifications of the same IS component. For example, the problem that may happen in the vertical interaction is the conflict between conceptual specifications at the information level and the corresponding technical specifications at the informatics level.

According to our approach, specification integration can be achieved through **coordination in development** with the support of an ISIS. In other words, the ISIS approach aims to support continuous specification integration with two categories of coordination (Figure 7-1):

- **Inter-IS coordination**, which addresses horizontal interaction, is the coordination of specifications of different information system components at the same level of abstraction;
- **Intra-IS coordination**, which addresses vertical interaction, is the coordination of specifications of the same information system component.

![Figure 7-1: Intra-IS and Inter-IS coordination.](image)

Indeed, in the context of the ISIS approach, Inter-IS coordination is the solution for inter-project coordination. Meanwhile, Intra-IS coordination is the solution for intra-project coordination.

### 7.2.3. Common objects and Private objects

In an ISIS, a conceptual specification is represented by an object. There are two types of ISIS objects: **Common objects** and **Private objects**.

A **common object** is elevated for strategic purposes above the interests of a single IS component. This implies a distinction between common objects and private objects.

A **private object** has an **owner** (the responsibility zone of the IS component to which it belonged); meanwhile a common object may have a **custodian** (or provider), an **owner** or **co-owners**. The owner of a private object has unlimited rights over the
object, including the ability the change the structure of the object without notifying anybody else. The custodian of a common object has certain responsibilities towards other users (or requesters) of the object.

Indeed, coordination in development is the coordination within and between information system components development projects. In IS development, the content of an information system component represents the state of knowledge of a project at a certain phase of development. A IS component can be developed through several stages in the development life cycle.

At a given time, there may exist several development projects, each at different stages of development. Coordination within a project depends on maintaining proper relationships between the several stages belonging to the project. Indeed, in the ISIS approach, Intra-IS coordination refers to the coordination between private objects belonged to the same project at different stages of the life cycle (Figure 7-2).

![Figure 7-2: Intra-IS coordination.](image)

On the contrary, Inter-IS coordination refers to the coordination between common objects belonging to different projects at the same stage in the development life cycle (Figure 7-3).

![Figure 7-3: Horizontal coordination between IS components](image)
7.3. **Intra-IS coordination**

This section continues to discuss about the Intra-IS coordination, presenting how to identify the interdependencies between specifications (representing by private objects) and how to enforce the coordination policies to manage those interdependencies.

7.3.1. **Interdependencies between specifications**

7.3.1.1. **Interdependences between dynamic states of objects of generic hyperclasses**

In the ISIS, the development process of a conceptual specification is represented by the object life cycle of the corresponding generic hyperclass.

Therefore, Intra-IS coordination can be considered as managing dependencies between the object life cycles of generic hyperclasses. Dependencies between those object life cycles can be expressed by their dynamic states. Consequently, coordination of object life cycles can be based on the interdependences between dynamic states of objects of generic hyperclasses of the ISIS.

In the ISIS, dynamic states of an object of a generic hyperclass should coordinate with dynamic states of other objects of generic hyperclasses in order to guarantee the coherence of the whole IS development process.

When an object of a generic hyperclass is in the dynamic state $s_1$, then it is compulsory that the states of objects of other generic hyperclasses, which are relative to this object, has to be in certain dynamic states conforming to the dynamic state $s_1$.

*Example 7-1: Coordination between the specifications of hyperclasses and specification of attributes.*

This example clarifies the coordination between specifications of hyperclasses and specifications of attributes. At first, Figure 7-4 reviews the structure of a hyperclass specification and an attribute specification.
Figure 7-4: Structures of Hyperclass and Attribute specifications.

The coordination between hyperclass and attribute specifications can be represented by the coordination of object life-cycles of the Hyperclass and Attribute generic hyperclasses as in Figure 7-5.

Figure 7-5: Coordination of the dynamic states of objects of Hyperclass and Attribute generic hyperclasses.

It is obvious that the Completed state of an object of Hyperclass generic hyperclass implies the Completed states of its related objects of Attribute generic hyperclass.

On the other hand, the Invalid state of an o-att object of Attribute generic hyperclass causes the Invalid state of the corresponding o-hcl object of Hyperclass generic
hyperclass, and the Uncompleted state of the o-att1 object causes the Uncompleted state of the o-hcl1 object.

Table 7-1 presents the impact-analysis table of changes of dynamic states of objects of Hyperclass and Attribute generic hyperclasses. The complete impact-analysis tables of changes of dynamic states of all the generic hyperclasses can be found in Annex 3.

<table>
<thead>
<tr>
<th>Hyper-class generic hyperclass</th>
<th>In dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Attribute generic hyperclass</th>
<th>Hyper-class generic hyperclass</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid</td>
<td>Implication</td>
<td>OriginHyperclass</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Invalid</td>
<td>Cause</td>
<td>IsACategory, terminationCategory</td>
<td>Invalid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completed</td>
<td>Implication</td>
<td>originHyperclass</td>
<td>Completed</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncompleted</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Attribute generic hyperclass</td>
<td>Valid</td>
<td>Implication</td>
<td>originHyperclass</td>
<td>Valid</td>
<td></td>
</tr>
<tr>
<td>Invalid</td>
<td>Cause</td>
<td>IsACategory, terminationCategory</td>
<td>Valid</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncompleted</td>
<td></td>
<td></td>
<td></td>
<td>originHyperclass</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

Table 7-1: Impact-analysis of changes of dynamic states of objects of Hyperclass and Attribute generic hyperclasses.

Let’s discuss about the first part of Table 7-1 that concerns with the impact-analysis of changes of dynamic states of Hyperclass generic hyperclass.

Firstly, when an object of Hyperclass generic hyperclass is in Valid dynamic state, there is one implication: the objects of Attribute generic hyperclass linked to this object via the originHyperclass attribute are in Valid states. The interpretation of that implication is that: “when a specification of a hyperclass is valid, then the specifications of the attributes, which are originated from this hyperclass, must be valid”.

Secondly, when an object of Hyperclass generic hyperclass becomes Invalid, this change causes the following effects: the objects of Attribute generic hyperclass, linked to this object via the originHyperclass attribute or linked to this object via the IsACategory and terminationCategory attributes become Invalid.

Thirdly, when an object of Hyperclass generic hyperclass is in Completed state, the objects of Attribute generic hyperclass linked to this object via the originHyperclass attribute must also be in Completed states.

Lastly, when an object of Hyperclass generic hyperclass becomes Uncompleted, there is no implication or effect on other objects of related generic hyperclasses.
7.3.1.2. Specifying Intra-IS coordination rules

To support the coordination of object life cycles of generic hyperclasses, the ISIS needs to guarantee the coordination rules, which represent the impact of the changes of dynamic states of objects of generic hyperclasses. Coordination rules can be extracted from the impact-analysis tables as presented above.

For instance, Table 7-2 presents the coordination rules extracted from the cross-analysis in previous example (Table 7-1). Those coordination rules concerns with the coordination of dynamic states of objects of Hyperclass generic hyperclass with objects of Attribute generic hyperclass.

<table>
<thead>
<tr>
<th>Coordination rule</th>
<th>Concerning generic hyperclass</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LCR_HCL#1</td>
<td>Hyperclass</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding objects of Attribute generic hyperclass, linked via originHyperclass attribute, must also be in Valid state.</td>
</tr>
<tr>
<td>LCR_HCL#2</td>
<td>Hyperclass</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding objects of Attribute generic hyperclass, linked via originHyperclass attribute, must also be in Invalid state.</td>
</tr>
<tr>
<td>LCR_HCL#3</td>
<td>Hyperclass</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding objects of Attribute generic hyperclass, linked via IsACategory and terminationCategory attributes, must also be in Invalid state.</td>
</tr>
<tr>
<td>LCR_HCL#4</td>
<td>Hyperclass</td>
<td>If an object of Hyperclass generic hyperclass is in Completed state then the corresponding objects of Attribute generic hyperclass, linked via originHyperclass attribute, must also be in Completed state.</td>
</tr>
<tr>
<td>LCR_ATTR#1</td>
<td>Attribute</td>
<td>If an object of Attribute generic hyperclass is in Valid state then the corresponding object of Hyperclass generic hyperclass, linked via originHyperclass attribute, must also be in Valid state.</td>
</tr>
<tr>
<td>LCR_ATTR#2</td>
<td>Attribute</td>
<td>If an object of Attribute generic hyperclass is in Valid state then the corresponding object of Hyperclass generic hyperclass, linked via IsACategory and terminationCategory attributes, must also be in Valid state.</td>
</tr>
<tr>
<td>LCR_ATTR#3</td>
<td>Attribute</td>
<td>If an object of Attribute generic hyperclass is in Invalid state then the corresponding object of Hyperclass generic hyperclass, linked via originHyperclass attribute, must also be in Invalid state.</td>
</tr>
<tr>
<td>Integrity Rule</td>
<td>Expression</td>
<td>Scopes</td>
</tr>
<tr>
<td>---------------</td>
<td>----------------------------------------------------------------------------</td>
<td>------------</td>
</tr>
<tr>
<td>LCR_HCL#1</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the</td>
<td>Hyperclass</td>
</tr>
<tr>
<td></td>
<td>corresponding objects of Attribute generic hyperclass, linked via origin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hyperclass attribute, must also be in Valid state.</td>
<td>Attribute</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LCR_HCL#2</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the</td>
<td>Hyperclass</td>
</tr>
<tr>
<td></td>
<td>corresponding objects of Attribute generic hyperclass, linked via origin</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Hyperclass attribute, must also be in Invalid state.</td>
<td></td>
</tr>
</tbody>
</table>
### Validating integrity rules originating from coordination rules.

As mentioned in Chapter 6, there are generic periods for validating integrity rules in an ISIS such as Initiation, Conception, Pre Uncompleted-implementation, Pre Completed-implementation, and Exploitation.

Table 7-4 presents how to validate integrity rules originating from coordination rules.

<table>
<thead>
<tr>
<th>Period</th>
<th>Enforced integrity rules originating from coordination rules</th>
<th>Authorized methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initiation</td>
<td></td>
<td>- Create, Query, and Finalize a specification</td>
</tr>
<tr>
<td>Conception</td>
<td></td>
<td>- Query, Finalize a specification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Specify attributes of a specification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete dependent specifications of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specifications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implement a specification</td>
</tr>
<tr>
<td>Pre Uncompleted-</td>
<td>- Integrity rules concerning validity dynamic states.</td>
<td>- Query, Finalize a specification</td>
</tr>
<tr>
<td>Implementation</td>
<td></td>
<td>- Specify attributes of a specification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete dependent specifications of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specifications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implement a specification</td>
</tr>
<tr>
<td>Pre Completed</td>
<td>- Integrity rules concerning validity dynamic states</td>
<td>- Query, Finalize a specification</td>
</tr>
<tr>
<td>Implementation</td>
<td>- Integrity rules concerning completeness dynamic states</td>
<td>- Specify attributes of a specification</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Create, Delete dependent specifications of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td>specifications</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Implement a specification</td>
</tr>
<tr>
<td>Exploitation</td>
<td>&lt;All integrity rules&gt;</td>
<td>Evolve, Query a specification.</td>
</tr>
</tbody>
</table>

Table 7-4: Periods for validating integrity rules originating from Intra-IS coordination rules.

Indeed, integrity rules originating from coordination rules need to be taken into at the moment when designers are going to complete the modelling phase and ready to implement conceptual specifications. Therefore, it is indispensable to validate those rules in the periods such as Pre Uncompleted-implementation and Pre Completed-implementation periods.
The coordination rules concerning the validity of specifications (i.e. the Valid and Invalid dynamic states) need to be validated in both Pre Uncompleted-implementation and Pre Completed-implementation periods. Meanwhile, the coordination rules concerning the completeness of specifications (i.e. the Completed and Uncompleted dynamic states) need to be validated in the Pre Completed-implementation period.

7.3.2.2. **Enforcing Intra-IS coordination rules**

Intra-IS coordination is the coordination between activities of the development process. Therefore, enforcing Intra-IS coordination rules is to guarantee the coherence of the coordination rules in the activities that specify specifications.

For this reason, we propose a *generic activity*, called “Specifying specifications”, whose purpose is to specify specifications. This activity must guarantee the Intra-IS and Inter-IS coordination rules. Specifying specifications includes the creation, modification and deletion of specifications stored in the development tools and in the ISIS.

The diagram representing the sub-activities of the “Specify specifications” activity is presented in Figure 7-6. There are two participators: the *environment* including the users of the ISIS and development tools and the *ISIS* including the service components of the ISIS.

**Specify specifications.**

![Diagram](image)

Figure 7-6: "Specify a specification" activity.

The environment can invoke a request for specifying a specification. The ISIS firstly verifies the request. Afterward, depending on the type of the specification, the ISIS continues to specify a private specification or specify a common specification. Private specifications can be published later to become common specifications.

In the following, we will continue to clarify the sub-activities of the “Specifying specifications” such as the “Invoke a request for specifying a specification”, “Verify the request”, and “Specify a private specification” activities.
The “Specify a common specification” and “Publish private specifications related to an overlap situation” activities will be discussed in the next section.

**“Invoke a request for specifying a specification” activity.**

![Diagram](image)

*Figure 7-7: “Invoke a request for specifying a specification” activity.*

Figure 7-7 presents the “Invoke a request for specifying a specification” activity. The *environment* now is refined to the *users* and the *development tools*. The *ISIS* is refined to its service components such as *Navigator, Specification Integrator, Service Integrator, Navigator* and *Administrator*.

At first, there are two possibilities to specify a specification in the ISIS. The first possibility is *direct request* using Navigator service component. Users of the ISIS can request to perform processes using Navigator service component. Thus, Navigator service component transfers the requested processes to the Administrator service component.

The second possibility is *indirect request* via development tools, Specification Integrator, and Service Integrator service components. Users of develop tools can request to perform processes in their own development tools. Development tools firstly perform the requested processes. Afterward, development tools transfer the requested processes to the ISIS via Service Integrator service component, or transfer the results (i.e. modified specification) via Specification Integrator service components.
Concerning different integration modes to integrate the development tools with the ISIS, direct requests can be used for Direct integration mode. Indirect requests via Service Integrator service component can be used for Replication and Registration integration modes. Indirect requests via Specification Integrator service component can be used for Replication, Agent and Loose integration modes.

“Verify the request” activity.

The “Verify the request” activity is performed by the Administrator service component. This activity firstly verifies the authorisation of the requestor.

Thus it classifies the requested processes according to two types of specifications: private and common specifications. Depending on the types of the specification, the request then is transferred to Intra-IS coordinator or Inter-IS coordinator service components to fulfil.

“Specify a private specification” activity.

Figure 7-8: "Verify the request" activity.

Figure 7-9: Intra-IS coordination.
After receiving all the necessary information, the Administrator service component verifies the privileges of the requestor. The requestor must be the owner or having sufficient privileges granted by the owner.

If the requestor has sufficient privileges, Administrator service component performs the requested processes. Consequently, the execution may change dynamic states of the specification.

Accordingly, Intra-IS coordinator service component identifies the consequences on dynamic states of related specifications, which have relationships with the modified specification. Thus, Intra-IS coordinator service component requests for changing dynamic states of related specifications to the Administrator service component. Moreover, Intra-IS coordinator service component also asks Navigator service component to inform the consequences to the owners of those specifications.

7.4. Inter-IS coordination

In certain enterprise-wide information systems, designers may have to work with thousands of classes. In this case, it is difficult to work efficiently and to control the coherence of a large number of conceptual specifications. Therefore, there is a need to work with a part of an IS, in particular a unique and coherent set of conceptual specifications.

For the same reason, we propose the Information System Component concept, which enables to work with a part of an IS. Consequently, Inter-IS coordination now becomes the coordination of IS component development projects, or in short: “coordination of IS components”.

In this following, we continue with the interdependences between IS components and how to coordinate those interdependencies.

7.4.1. Interdependences between IS components

In the ISIS approach, an information system is constructed by a set of IS components. For this reason, the IS development project is decomposed into several development projects of IS components in parallel.

The interdependences between IS components reflect the real interdependences between / within the responsibility zones that manage and use those IS components. Moreover, they reflect the real interdependences between / within the projects that design and develop those components.
The interdependences between IS components are represented by overlap situations and overlap situations are operated by overlap protocols.

### 7.4.1.1. Overlap situations

Information overlap among IS components is indispensable when several IS components share a common subset of information. Therefore, the interdependences between IS components can be represented by overlap situations between responsibility zones of IS components.

A responsibility zone (RZ) is a part of organization-working environment to which IS components may correspond. An IS component may correspond to only a unique responsibility zone. A development team is responsible to carry out a development project. Each team takes on a responsibility zone; and therefore, is in charge of the IS components assumed by the RZ.

An **overlap situation** occurs when there is at least one class or one process/method is common to several IS components.

There are possibly three types of overlap situations: i) *Distinct* (no common class and no common process/method); ii) *With borders* (common classes but no common process/method); and iii) *With overlaps* (common classes and common processes/methods).

### 7.4.1.2. Specifying overlap situations

The overlap area of several IS components is a subset of classes, which are common to those IS components. A class cross-analysis table can be used to present an overlap area, including all the concerned overlap situations, between a set of IS components.

**Example 7-2: Overlap area and overlap situations.**

Let’s continue with the example about the information overlap between the *Sale management* and *Inventory management* IS components in Chapter 4 (Figure 7-10).
Figure 7-10: Overlap area between the Sale management and Inventory management IS components.

The class cross-analysis table of the two IS components is presented in Table 7-5. The overlap area of them includes the three classes: Sale order, Sale order item, and Inventory item.

Accordingly, there are three overlap situations: i) $o_{S1}$ concerning the Sale order class; ii) $o_{S2}$ concerning the Sale order item class; and iii) $o_{S3}$ concerning the Inventory item class.

<table>
<thead>
<tr>
<th>Class</th>
<th>Sale management IS component</th>
<th>Inventory management IS component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Payment</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Sale order</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Sale order item</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Inventory Item</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Purchase order</td>
<td>*</td>
<td></td>
</tr>
<tr>
<td>Purchase order item</td>
<td>*</td>
<td></td>
</tr>
</tbody>
</table>

Table 7-5: Class cross-analysis table.

To identify the types of those overlap situations, we can use the pre- and post-hyperclasses cross-analysis table (as in Table 7-6), which represent how the processes of each IS component use the classes in the overlap area.

For instance, the row in Table 7-6 indicates that there are processes in the Sale management IS components that use the Sale order class as their pre-hyperclass and/or their post-hyperclass. Meanwhile, there is no process in Inventory management IS component that use Sale order class as its post-hyperclass.
It means that the RZ of the Sale management IS component needs to modify the specification of the Sale order class; meanwhile the RZ of the Inventory management IS component only needs to refer the specification of that class.

Consequently, according to the information in Table 7-6, we can identify that the $os_1$ (concerning the Sale order class) and $os_2$ (concerning the Sale order item class) are With border overlap situations. Meanwhile, the $os_3$ (concerning the Inventory item class) is With overlap situation.

Concerning the $os_1$ and $os_2$ With-border overlap situations, the RZ of the Sale management IS component is responsible for defining and maintaining the specifications of the Sale order and Sale order item classes and the specifications of their dependents such as their attributes, keys, and sub-hyperclasses. The RZ of the Inventory management IS component has the right to refer to those specifications.

<table>
<thead>
<tr>
<th>Class</th>
<th>Sale management IS component</th>
<th>Inventory management IS component</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pre-Hyperclasses</td>
<td>Post-Hyperclasses</td>
</tr>
<tr>
<td>Sale order</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Sale order item</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Inventory Item</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Table 7-6: Pre- & Post-hyperclass cross-analysis table.*

Concerning the $os_3$ With-overlap situations, we need to specify the process cross-analysis table to indicate the process overlap in details. Table 7-7 presents the process cross-analysis table concerning the $os_3$ overlap situation. This table introduces all the processes related to the Inventory Item class. The common processes of the two IS components are the Decreasing-quantity-in-stock and Undo-decreasing-quantity-in-stock processes.

<table>
<thead>
<tr>
<th>Process</th>
<th>Sale management IS component</th>
<th>Inventory management IS component</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creating new inventory item</td>
<td></td>
<td>*</td>
</tr>
<tr>
<td>Increasing quantity in stock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Decreasing quantity in stock</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>Undo Decreasing quantity in stock</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

*Table 7-7: Process cross-analysis table.*

Therefore, the both RZs of the Sale management and Inventory management IS components share the same responsibility for defining and maintaining the concerning specifications such as:

- The specification of the Inventory-Item class (and the specifications of its dependents such as attributes, keys, and sub-hyperclass);
• The specifications of the Decreasing-quantity-in-stock and Undo-decreasing-quantity-in-stock processes (and the specifications of their dependents such as methods); and

• The specification of the integrity rules (and the specifications of its dependents such as scopes and risks) defined on the Inventory-Item class and concerned by the two processes.

7.4.2. Inter-IS coordination supported by the ISIS

This part firstly begins with the overlap protocols to coordinate overlap situations between IS components. Thus it illustrates how the ISIS stores different types categories of specifications and the interrelationships between them. Finally, it presents how the ISIS enforces the overlap protocols as well as Inter-IS coordination rules.

7.4.2.1. Overlap protocols

Inter-IS coordination is actually the coordination of IS components that aims to manage the information overlap between IS components. In our approach, Inter-IS coordination can be realized using overlap protocols.

An overlap protocol is a protocol that allows each responsibility zone (RZ) to perform its own processes locally and to monitor the processes in other RZs, which can influence its own processes.

In fact, an overlap protocol includes a set of semantics, rules, and formats that conduct the cooperation between IS components. At the time being, we propose the following categories of overlap protocols, corresponding to the three theories of coordination as mentioned above:

- **Ownership-based overlap protocol**, corresponding to the Hierarchical theory of coordination, appoints which RZ would play the role of the owner for each common object. The owner of an object takes the responsibility for defining, developing and maintaining it. The other RZs may communicate to the owner to obtain information about this object.

- **Service-based overlap protocol**, corresponding to the Market theory of coordination, appoints which RZ would play the role of the provider for each common object. The provider of an object takes the responsibility for providing services related to this object. This protocol allows other RZs to send a request to perform a process to the provider. Normally, the provider will perform the requested process and return the result to the requested IS component.

- **Watch-based overlap protocol**, corresponding to Network theory of coordination, allows a RZ to monitor the consequences when other RZs
performed a process, which is overlapped between them. Indeed, each RZ plays the role of the co-owner for each common object.

**Example 7-3: Overlap protocols**

Let’s continue with the previous example about the $os_1$, $os_2$ and $os_3$ overlap situations.

At first, the Owner-ship based overlap protocol can apply for the $os_1$ and $os_3$ With border overlap situations. The RZ of the Sale management IS component will play the role of the owner of the specifications Sale order and Sale order item classes and the specifications of its dependents. The RZ of the Sale management IS component is appointed as the owner because the IS components of this RZ has the processes, which may define and modify those specifications. Consequently, the RZ of the Inventory management IS component can communicate with the owner to obtain those specifications.

On the other hand, both the Service-based and Watch-based overlap protocols can apply for the $os_3$ With-overlap overlap situations.

If the Service-based overlap protocol is selected, the RZ of the Inventory management IS component may play the role of the provider of the specifications of the Decreasing-quantity-in-stock and Undo-decreasing-quantity-in-stock processes. The RZ of the Inventory management IS component is appointed as the provider because this RZ is already responsible for the specifications of other processes, which concern the Inventory Item class, such as Creating-new-inventory-item, and Increasing-quantity-in-stock processes.

If the Watch-based overlap protocol is selected, both the RZs of the Sale management and Inventory management may play the role of the co-owner of the specifications of the two processes.

### 7.4.2.2. Private workspace and common workspace

In the ISIS, there are two types of objects: **Private object** and **Common object**. Private objects represents specifications owned by the RZ of an IS component. Common objects are overlapped between several RZs.

Specifications of the ISIS are stored in the ISIS repository, which includes two workspaces: **Private workspace** and **Common workspace**. Private workspace stores private objects of development teams. Meanwhile, common workspace stores common objects.

Private object is interested by a RZ and can have an owner, which can be a role in the development team. Meanwhile, a common object can be interested by several RZs.

A common object may have an owner RZ, a custodian (provider) RZ, or co-owners RZs depending on the overlap protocol apply for it. Similarly, a common object is referred to a private object or several private objects.
If a common object is operated by the Ownership-based overlap protocol, then a common object has a RZ as its owner. Thus the content of the common object is referred to the content of the original private object owned by that RZ.

If a common object is operated by the Service-based overlap protocol, then a common object has a RZ as its provider (or custodian). Thus the content of the common object can be referred using the services provided by the provider RZ.

If a common object is operated by the Watch-based overlap protocol, then a common object has several RZs as its co-owners. Thus the content of the common object can be referred from the content of the corresponding private object of one of its co-owners.

7.4.2.3. Publishing private objects

Publishing (or sharing) a private object means to make it become a common object. In other words, a private object is at the local level corresponding to particular development team. Meanwhile a common object is at the global level and is interested by several development teams.

At the global level, it is also necessary to guarantee the validity of common objects. Therefore, it is important to know that a common object is valid or not.

However, there is no need to concern about the completeness and implementation of common objects. Roughly speaking, there is no need to know that a common object is completed or not. It may be completed in a team but uncompleted in other teams. Similarly, a common object may be implemented in a team but unimplemented in other teams.

For this reason, the Intra-IS coordination rules concerning the validity of specifications need to be enforced at the global level. We called those rules: the Intra-IS coordination rules at the global level.

On the other hands, when publishing a private object, there is a decision to be made: what dependent objects of this object will be published. This decision must be approved by all the participators such as owner and referrers, provider and requestors, and co-owners.

For instance, when publishing a private specification of a hyperclass, we have to decide what specifications of its dependents such as attributes, keys, sub-hyperclasses will be published.

Once a specification of a dependent specification is published, it must be also valid at the global level. This may lead to further actions at the global level. For example, we decide to publish a specification of a dependent attribute. However, the specification
of the atomic-class on which that that attribute terminates has not yet publish. Therefore, we need to publish the specification of that atomic-class.

7.4.2.4. Enforcing overlap protocols and coordination rules

Inter-IS coordination is the coordination between responsibility zones of IS components. Therefore, enforcing overlap protocols and coordination rules is to ensure that the activities concerned in specifying common specifications must conform to them.

Let’s return to the workflow concerning the generic “Specifying a specification” activity that aims at specifying a specification, possibly a private or a common specification (Section 7.3.2). This section continues to clarify its two sub-activities: the “Publish a private specification” and “Specify a common specification” activities.

“Publish private specifications related to an overlap situation” activity.
The workflow representing the “Publish private specifications related to an overlap situation” activity is presented in Figure 7-11. There are two participators: User and Inter-IS Coordinator service component.

![Figure 7-11: "Publish private specifications related to an overlap protocol" activity.](image)

The user can invoke a request for publishing private specifications related to an overlap situation. Depending the overlap situations, private specifications to be published can be common hyperclasses or process classes.
Firstly, the ISIS publishes the specification of common hyperclass if it is not already published. Then the ISIS asks the user to select the specifications of its dependents to publish such as its attributes, keys and sub-hyperclasses.

Secondly, if the overlap situation is “With-overlap”, the ISIS continues to publish the specifications of the common processes. Accordingly, the ISIS also asks the user to select the specifications of the dependents of the common processes such as their methods and dynamic states to publish.

Thirdly, the ISIS continues with the specifications of the integrity rules defined on the common hyperclass and concerned by the common processes. The specifications of those integrity rules and their selected dependents (i.e. scopes and risks) will be published.

Finally, the ISIS verifies the validity of all the published specifications using the coordination rules. There may exist some missing elements and the specifications of those elements need to be published too.

“Specify a common specification” activity.
The “Specifying a common specification” activity is presented in Figure 7-12. It is refined into three sub-activities according to different overlap protocols: “Specify a common specification operated by Ownership-based overlap protocol”, “Specify a common specification operated by Service-based overlap protocol”, and “Specify a common specification operated by Watch-based overlap protocol” activities.

![Diagram](image-url)

Figure 7-12: “Specify a common specification” activity.

“Specify a common specification operated by Ownership-based overlap protocol” activity.
Figure 7-13 presents the “Specify a common specification operated by Ownership-based overlap protocol” activity.
When receiving a request for specifying a common specification operated by Ownership-based overlap protocol, the Inter-IS Coordinator firstly verifies whether the requestor participates in the Owner RZ.

If the requestor does not participate in the Owner RZ, the Inter-IS service component can transfer the request to the Owner RZ. Thus Inter-IS Coordinator service component waits for the notification. If the notification is favourable, then a request is transferred to Intra-IS Coordinator service component for specifying the private specification (corresponding to the common specification) in the private workspace of the requester. Otherwise, a refuse is informed by Navigator service component.

If the requester participates in the Owner RZ, the Inter-IS service component verifies whether the requestor is the owner of the common specification or has sufficient privileges granted by its owner. If the requestor does not have sufficient privileges, a refuse is informed by Navigator service component.

If the requestor has sufficient privileges, the Inter-IS Coordinator service component performs the requested processes on the common specification stored at the Common workspace. Thus the common specification may change its dynamic states. Therefore, it is necessary to perform the coordination of dynamic states of the common specification, which has interrelationships with this common specification. Afterward, a request to perform the corresponding processes on the original private specification of the common specification is transferred to the Intra-IS Coordinator.
“Specify a common specification operated by Service-based overlap protocol” activity.

The “Specify a common specification operated by Service-based overlap protocol” activity is presented in Figure 7-14. In general, this activity is the same scenario as the previous activity.

Figure 7-14: “Specify a common specification operated by Service-based overlap protocol” activity.

Nevertheless, there are two differences:

- When the requestor does not participate in the Provider RZ, the Inter-IS Coordinator service component will invoke the corresponding services to the Provider RZ (instead of sending a request).
- When the requestor participates in the Provider RZ and has sufficient privileges, the Inter-IS Coordinator will perform the requested services (instead of performing processes).

“Specify a common specification operated by Watch-based overlap protocol” activity.

The “Specify a common specification operated by Watch-based overlap protocol” activity is presented in Figure 7-15.
Figure 7-15: “Specify a common specification operated by Watch-based overlap protocol” activity.

When receiving a request for specifying a common specification operated by Watch-based overlap protocol, the Inter-IS Coordinator firstly verifies whether the requestor participates in the Co-owner RZ and has sufficient privileges on that RZ or not. If not, a refuse is informed by Navigator service component.

Thus the Inter-IS Coordinator service component continues to perform the requested processes on the common specification, perform the coordination of dynamic states of related common specifications, and to request for performing the corresponding processes on the private specification stored at the private workspace of the requestor.

Accordingly, the Inter-IS Coordinator also sends a message as an alert to inform the other co-owner RZs to update their related private specifications stored in their private workspace.

7.5. Conclusion

This chapter presents the part of the framework that supports with the coordination of conceptual specifications.

At first, coordination means identifying the interdependencies and determining the responsibilities of each contributor.

In the ISIS approach, the global IS development project can be decomposed into several development projects corresponding to different IS components. Consequently, the interdependencies within/between systems and projects can be managed in terms of the interdependencies within/between IS components supporting them.
**Intra-IS coordination** concerns with the interdependencies within an IS component. Indeed, those interdependencies can be considered as the dependencies of dynamic states of private objects of the generic hyperclasses of the ISIS. For this reason, intra-IS coordination becomes the coordination between object life cycles of the generic hyperclasses of the ISIS.

Therefore, an ISIS supports Intra-IS coordination by enforcing the coordination rules representing the impact of changes of dynamic states of objects of generic hyperclasses.

**Inter-IS coordination** concerns with the interdependencies between different IS components. Those interdependencies can be represented in terms of the overlap situations between the IS components. Consequently, Inter-IS coordination is the coordination of information overlaps between responsibility zones, which manage IS components.

Overlap protocols are proposed to manage overlap situations. There are three overlap protocols such as Ownership-based, Service-based, and Watch-based overlap protocols.

To enforce coordination rules and overlap protocols, the activities, involved in specifying specifications, must ensure those rules and protocols. For this reason, the ISIS approach proposes a generic activity: “Specifying a specification”, which ensures all the overlap protocol as well as coordination rules. This workflow can be customized and adapted to a particular development process of an enterprise.
Chapter 8.

CONCLUSION

8.1. Conclusion

Our research concerns with an approach for managing and coordinating information resources used in information system (IS) development. The proposed approach focuses on conceptual specifications, which are the specifications of IS concepts at the information level and independent with computing platforms.

The approach also considers that conceptual specifications of information systems are the fundamental constituents of information resources; and therefore, the first objective is to identify, manage and coordinate conceptual specifications during the earlier phases of development process.

Identifying and managing conceptual specifications early promise to improve the quality of information systems as well as reduce the costs of integration, adaptation, coordination, evolution and maintenance. In our knowledge, at the time being, there is still a little effort to manage and coordinate conceptual specifications in a coherent manner.

Our research domain concerns with an information system that supports traditional information systems, in particular supporting the management and coordination of information resources used in IS development with the focus on conceptual specifications. This information system is called the Information System upon Information Systems (ISIS).

For this reason, this thesis aims at presenting a conceptual framework for building and managing an ISIS.

The first part of the conceptual framework concerns with how to work with conceptual specifications. Despite the diversity of IS development processes, there are some categories of conceptual specifications that are invariant. Therefore, the thesis firstly presents how to identify these categories of conceptual specifications.

These categories can be classified according to the three aspects of information system such as: i) The Static aspect: Atomic-class, Hyperclass, Attribute, Key and Sub-Hyperclass specifications; ii) The Dynamic aspect: Event, Process, Method, and
Integrity rule specifications; and iii) The Integrity rule aspect: Integrity rule, Scope and Risk specifications.

The second part of the framework concerns with how to work efficiently with a part of an IS, in particular a unique and coherent set of conceptual specifications, called IS components. Modelling IS with IS components could be a powerful technique of managing and coordination information systems at the information level.

The third part conceptual framework concerns with the architecture of an ISIS, including the following objectives: i) how to manage conceptual specifications in the ISIS; ii) how to represent the structure, behaviour, and coherence of conceptual specifications; and iii) How to coordinate conceptual specifications within or between development projects, called Intra-IS coordination and Inter-IS coordination.

In conclusion, the contribution of our work is to provide a unique and coherent framework to identify, manage, represent, and coordinate conceptual specifications based on the new pioneers such as Information System Component and Information System upon Information Systems.

The perspective of this work is to provide an effective architecture that would be best suited for the management, representation and coordination of specifications of existing development tools and information systems.

### 8.2. Present works

At present, we are developing a first prototype (developed in the web-based environment) of a generic ISIS that supports working with conceptual specifications and the development with IS components.

The objects of this prototype are: i) To manage, represent and coordinate different categories of conceptual specifications; ii) To monitor and realize overlap protocols; and iii) To guarantee the validity and completeness of specification at different levels of abstraction.

Additionally, we have also foreseen to extend our prototype to cooperate with some systems and tools used in IS development, in particular the Computer Aided System Engineering (CASE), Software Configuration Management (SCM) and Database Management System (DBMS) tools and systems.

On the other hand, we have also foreseen to continue our work on IS component engineering. The next challenges are: i) to extract IS components from the living world and the artificial world; and ii) to reuse IS components at the earlier phases of IS development process.
8.3. Future works

Our future works concerns with providing multi-view of specification, supporting the evolution of information systems, coordinating with the Method engineering approaches, providing IS project management framework and training new competences for IS professionals.

Providing multi-view of specifications.
Furthermore, each development teams can have their own views on a conceptual specification. For instance, a conceptual specification may have two names corresponding to two different development teams; or a specification may have two names at two different points of time.

For this reason, we intend to extend our conceptual framework with two new dimensions: the Time dimension and the Intention dimension. Those two dimensions will help the ISIS to work with different points of view at different points of time.

Supporting the evolution of information systems.
In future work, we will focus our research on exploiting the benefit of ISIS in supporting the evolution of information systems. This evolution will also consist an extension of business activities supported by IS to coordinate with the responsibilities performed by organization actors, or the other case, the redesign the human responsibilities to be conformed to the implementation of an IS.

Co-ordinating with Method Engineering approaches.
Method engineering approaches concerns with the components of methods that include two parts: Product and Process. The categories of specifications proposed by the ISIS approach can be used to construct the generic products used in IS development. From the generic products, the corresponding processes can be defined accordingly to form the components of method.

Providing IS project management framework.
With the existence of the ISIS as a new IS infrastructure; there are new responsibilities for the IT staff. IS project management now includes the management of IS development as well as the management of the ISIS development and exploitation. Furthermore, the management of IS development becomes the management of several developments of IS components, which are developed in parallel or in sequence and at different stages of development process. Besides, there is the need to study the assimilation process in order to guarantee the coordination between the artificial world and reality world.

Training new competences for IS professional.
Nowadays, there is a tendency to make IS development becomes more independent with the informatics level. Consequently, IS professionals need to have new competences in order to work with the information level. There are the new roles
such as *Meta-model Modeller, Specification Integrator*, and *Specification Coordinator* requires certain knowledge on the ISIS framework as well as on Requirement engineering, Method engineering, industrial development frameworks and software processes.
Annex 1.
AN EXAMPLE ABOUT CONCEPTUAL
SPECIFICATIONS OF AN INFORMATION SYSTEM

This annex illustrates an example about the specification of a typical information system. This information system (IS) aims at supporting business activities of a small enterprise, named MATIS, which has been discussed in Chapter 3.

This example goes into details the specifications of the Static, the Dynamic and the Integrity rule aspects of the IS for MATIS, called the MATIS-IS.

Specification of the Static aspect

-- Atomic-classes

<table>
<thead>
<tr>
<th>ATOMIC-CLASS Code</th>
<th>SIZE Character:10</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOMIC-CLASS Name</td>
<td>SIZE Character:40</td>
</tr>
<tr>
<td>ATOMIC-CLASS Address</td>
<td>SIZE Character:80</td>
</tr>
<tr>
<td>ATOMIC-CLASS Telephone</td>
<td>SIZE Character:10</td>
</tr>
<tr>
<td>ATOMIC-CLASS Text</td>
<td>SIZE Character:80</td>
</tr>
<tr>
<td>ATOMIC-CLASS Date</td>
<td>SIZE Date</td>
</tr>
<tr>
<td>ATOMIC-CLASS Quantity</td>
<td>SIZE Integer:12:3</td>
</tr>
<tr>
<td>ATOMIC-CLASS Amount</td>
<td>SIZE Real:14:2</td>
</tr>
<tr>
<td>ATOMIC-CLASS DateTime</td>
<td>SIZE DateTime</td>
</tr>
</tbody>
</table>

-- Tuple-classes

<table>
<thead>
<tr>
<th>HYPERCLASS Customer</th>
<th>IS TUPLE-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE CustomerCode</td>
<td>TERMINATION Code (1,1)</td>
</tr>
<tr>
<td>ATTRIBUTE CustomerName</td>
<td>TERMINATION Name (0,1)</td>
</tr>
<tr>
<td>ATTRIBUTE CustomerAddress</td>
<td>TERMINATION Address (0,1)</td>
</tr>
<tr>
<td>ATTRIBUTE CustomerTelephone</td>
<td>TERMINATION Telephone (0,2)</td>
</tr>
<tr>
<td>ATTRIBUTE ActivatedTime</td>
<td>TERMINATION DateTime(0,1)</td>
</tr>
<tr>
<td>KEY CustomerKey</td>
<td>ATTRIBUTES CustomerCode</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>HYPERCLASS Sale order</th>
<th>IS TUPLE-CLASS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE SaleOrderNumber</td>
<td>TERMINATION Code (1,1)</td>
</tr>
<tr>
<td>ATTRIBUTE SaleOrderDate</td>
<td>TERMINATION Date (0,1)</td>
</tr>
<tr>
<td>ATTRIBUTE ReceivedTime</td>
<td>TERMINATION DateTime(0,1)</td>
</tr>
<tr>
<td>ATTRIBUTE AcceptedTime</td>
<td>TERMINATION DateTime(0,1)</td>
</tr>
<tr>
<td>ATTRIBUTE CancelledTime</td>
<td>TERMINATION DateTime(0,1)</td>
</tr>
<tr>
<td>ATTRIBUTE DeliveredTime</td>
<td>TERMINATION DateTime(0,1)</td>
</tr>
</tbody>
</table>
ATTRIBUTE PaidTime TERMINATION DateTime(0,1)
ATTRIBUTE ItemReturnedTime TERMINATION DateTime(0,1)
ATTRIBUTE SaleOrderAmount TERMINATION Amount (0,1)
ATTRIBUTE belongedToCustomer TERMINATION Customer(0,1)
KEY SaleOrderKey ATTRIBUTES SaleOrderNumber

HYPERCLASS Sale order item IS TUPLE-CLASS
ATTRIBUTE ofSaleOrder TERMINATION Sale order(1,1)
ATTRIBUTE SO_Item TERMINATION Inventory Item (0,1)
ATTRIBUTE Quantity ordered TERMINATION Quantity(0,1)
ATTRIBUTE SO_ItemAmount TERMINATION Amount (1,1)
KEY SaleOrderItemKey ATTRIBUTES ofSaleOrder,SO_Item

HYPERCLASS Inventory item IS TUPLE-CLASS
ATTRIBUTE ItemCode TERMINATION Code (1,1)
ATTRIBUTE ItemName TERMINATION Name (0,1)
ATTRIBUTE ItemDescription TERMINATION Text (0,1)
ATTRIBUTE ItemUnitPrice TERMINATION Amount (0,1)
ATTRIBUTE QuantityInStock TERMINATION Quantity(0,1)
ATTRIBUTE ItemActivatedTime TERMINATION DateTime(0,1)
KEY InventoryItemKey ATTRIBUTES ItemCode

HYPERCLASS Payment IS TUPLE-CLASS
ATTRIBUTE PaymentCode TERMINATION Code (1,1)
ATTRIBUTE paymentFor TERMINATION Sale order(0,1)
ATTRIBUTE PaymentDate TERMINATION Date (0,1)
ATTRIBUTE PaymentAmount TERMINATION Amount (0,1)
ATTRIBUTE ApprovedTime TERMINATION DateTime(0,1)
KEY PaymentKey ATTRIBUTES PaymentCode

HYPERCLASS LastYearSO IS TUPLE-CLASS
IS SUB-HYPERCLASS OF Sale order
WHEN year(SaleOrderDate) = this-year()-1

-- Hyperclasses

HYPERCLASS Sale
ATTRIBUTE hasCustomer TERMINATION Customer (0,1)
ATTRIBUTE hasSaleOrder TERMINATION SaleOrder(1,1)
ATTRIBUTE hasSaleOrderItems TERMINATION Sale order Item (0,n)
ATTRIBUTE hasItems TERMINATION Inventory Item (0,n)
ATTRIBUTE hasPayments TERMINATION Payment
KEY SaleKey ATTRIBUTES hasSaleOrder

Specification of the Dynamic aspect

-- Dynamic states

HYPERCLASS SO_demand_received
IS SUB-HYPERCLASS OF Sale order
WHEN ReceivedTime is not NULL
HYPERCLASS SO_demand_invalid
  IS SUB-HYPERCLASS OF SO_demand_received
  WHEN CancelledTime is not NULL

HYPERCLASS SO_demand_valid
  IS SUB-HYPERCLASS OF SO_demand_received
  WHEN AcceptedTime is not NULL

HYPERCLASS SO_customer_valid
  IS SUB-HYPERCLASS OF SO_demand_valid
  WHEN BelongedToCustomer is not NULL

HYPERCLASS SO_items_delivered
  IS SUB-HYPERCLASS OF SO_customer_valid
  WHEN DeliveredTime is not NULL

HYPERCLASS SO_amount_paid
  IS SUB-HYPERCLASS OF SO_items_delivered
  WHEN PaidTime is not NULL

HYPERCLASS SO_items_returned
  IS SUB-HYPERCLASS OF SO_amount_paid
  WHEN ItemReturnedTime is not NULL

HYPERCLASS Customer_activated
  IS SUB-HYPERCLASS OF Customer
  WHEN ActivatedTime is not NULL

HYPERCLASS Customer_deactivated
  IS SUB-HYPERCLASS OF Customer
  WHEN ActivatedTime is NULL

HYPERCLASS Inventory_item_activated
  IS SUB-HYPERCLASS OF Inventory_item
  WHEN ItemActivatedTime is not NULL

HYPERCLASS Inventory_item_deactivated
  IS SUB-HYPERCLASS OF Inventory_item
  WHEN ItemActivatedTime is NULL

HYPERCLASS Inventory_item_in_stock
  IS SUB-HYPERCLASS OF Inventory_item_activated
  WHEN QuantityInStock >= 0

HYPERCLASS Inventory_item_out_of_stock
  IS SUB-HYPERCLASS OF Inventory_item_activated
  WHEN QuantityInStock < 0

HYPERCLASS Payment_approved
  IS SUB-HYPERCLASS OF Payment
  WHEN ApprovedTime is not NULL

-- Dynamic states of the Sale order tuple-class
DYNAMIC STATE SO_demand_received
DYNAMIC STATE SO_demand_valid
DYNAMIC STATE SO_demand_invalid
DYNAMIC STATE SO_customer_valid
DYNAMIC STATE SO_items_delivered
DYNAMIC STATE SO_amount_paid
DYNAMIC STATE SO_items_returned

-- Dynamic states of the Customer tuple-class
DYNAMIC STATE Customer_activated
DYNAMIC STATE Customer_deactivated
-- Dynamic states of the InventoryItem tuple-class
DYNAMIC STATE Inventory_item_activated
DYNAMIC STATE Inventory_item_deactivated
DYNAMIC STATE Inventory_item_in_stock
DYNAMIC STATE Inventory_item_out_of_stock

-- Dynamic states of the Payment tuple-class
DYNAMIC STATE Payment_approved

-- Methods

-- Method of the Sale order tuple-class
METHOD ReceivingDemand OF HYPERCLASS Sale_order
FROM STATES Sale_Order
TO STATES SO_demand_received
INVOLVED ATTRIBUTES Sale_order_number, Sale_order_date,
of_Sale_order, SO_Items, Quantity ordered, ReceivedTime

METHOD CheckingStock OF HYPERCLASS Sale order
FROM STATES SO_demand_received
TO STATES SO_demand_valid, SO_demand_invalid
INVOLVED ATTRIBUTES AcceptedTime, CancelledTime

METHOD GettingCustomerInfo OF HYPERCLASS Sale_order
FROM STATES SO_demand_valid
TO STATES SO_customer_valid
INVOLVED ATTRIBUTES Belonged_to_customer

METHOD DecreasingQuantityInStock OF HYPERCLASS Sale_order
FROM STATES SO_customer_valid
TO STATES SO_items_delivered
INVOLVED ATTRIBUTES DeliveredTime

METHOD ReceivingPayment OF HYPERCLASS Sale_order
FROM STATES SO_items_delivered
TO STATES SO_amount_paid
INVOLVED ATTRIBUTES Sale Order Amount, PaidTime,
SO_ItemAmount

METHOD ReceivingReturnedItems OF HYPERCLASS Sale_order
FROM STATES SO_amount_paid
TO STATES SO_item_returned
INVOLVED ATTRIBUTES ItemReturnedTime

-- Method of the Customer tuple-class
METHOD CreatingNewcustomer OF HYPERCLASS Customer
FROM STATES Customer
TO STATES Customer_activated
INVOLVED ATTRIBUTES Customer_code, Customer_name,
Customer_address, Customer_telephone

-- Method of the Payment tuple-class
METHOD ApprovingPayment OF HYPERCLASS Payment
FROM STATES Payment
TO STATES Payment_approved
INVOLVED ATTRIBUTES Payment_code, Payment_date,
Payment_amount, ApprovedTime
Method of the Inventory item tuple-class

METHOD DecreasingQuantityInStock OF HYPERCLASS Inventory_item
FROM STATES Inventory_item_in_stock
TO STATES Inventory_item_activated
INVOLVED ATTRIBUTES QuantityInStock

METHOD UnDoDecreasingQuantityInStock OF HYPERCLASS Inventory_item
FROM STATES Inventory_item_activated
TO STATES Inventory_item_in_stock
INVOLVED ATTRIBUTES QuantityInStock

Events

EVENT DemandForItems
EVENT AcceptUnitPricesAndGivePersonalInformation
EVENT PayTheSaleOrder
EVENT ReturnItems

Processes

Processes of the Sale hyperclass

PROCESS ReceivingAndValidatingDemand
IN HYPERCLASS Sale
PRE-CONDITION -- A blank sale order
\exists so ∈ SALE ORDER | (so.valid is NULL) ∧ (so.invalid is NULL)
POST-CONDITION -- A valid sale order and a list of valid items;
-- or an invalid sale order
(\exists so.SO_demand_invalid is TRUE) ∨
(\exists so.SO_demand_valid is TRUE ∧
(\exists so_i, so_i,...,so_i), so_i ∈ SALE-ORDER-ITEM
(so_i.ofSaleOrder = so.SaleOrderNumber) ∧
(so_i.QuantityOrdered > 0) ∧
(\exists i ∈ INVENTORY-ITEM, i.ItemCode = so_i.SO-Item))
PRE-HYPERCLASS Sale order, Sale order Item, Inventory item
POST-HYPERCLASS SO_demand_valid, SO_demand_invalid,
Sale_order_item, Inventory_item_in_stock
TRIGGERED BY DemandForItems
EXECUTE ReceivingDemand, CheckingStock

PROCESS GettingCustomerInformation
IN HYPERCLASS Sale
PRE-CONDITION -- A sale order without customer info
\exists so ∈ SALE ORDER | (so.SO_demand_valid is TRUE) ∧
(so.belongedToCustomer is NULL)
POST-CONDITION -- A sale order with valid customer info
\exists c ∈ CUSTOMER ACTIVATED |
(so.belongedToCustomer = c.CustomerCode)
PRE-HYPERCLASS SO_demand_valid, Customer
POST-HYPERCLASS SO_customer_valid, Customer_activated
TRIGGERED BY AcceptUnitPricesAndGivePersonalInformation
EXECUTE GettingCustomerInfo, CreatingNewCustomer
PROCESS Decreasing_quantity_in_stock
IN HYPERCLASS Sale
PRE-CONDITION -- A sale order without delivered time
   ∃ so ∈ SALE ORDER | (so.belongedToCustomer is NOT NULL)
   ^ (so.deliveredTime is NULL)
POST-CONDITION -- A sale order with delivered time
   so.deliveredTime is NOT NULL

PRE-HYPERCLASS SO_customer_valid, Inventory_item_in_stock,
   Sale_order_item
POST-HYPERCLASS SO_items_delivered, Sale_order_item,
   Inventory_item_activated
EXECUTE DecreasingQuantityInStock

PROCESS Receiving_payment
IN HYPERCLASS Sale
PRE-CONDITION -- A sale order without payment info
   ∃ so ∈ SALE ORDER | (so.paidTime is NULL)
   ^ (so.deliveredTime is NOT NULL)
POST-CONDITION -- A sale order with valid payment info
   ∃ p ∈ PAYMENT APPROVED |
       (p.paymentFor = so.SaleOrderNumber)
   ^ (so.paidTime is NOT NULL)
PRE-HYPERCLASS SO_items_delivered, Payment
POST-HYPERCLASS SO_amount_paid, Payment_approved
TRIGGERED BY PayTheSaleOrder
EXECUTE ReceivingPayment, ApprovingPayment

PROCESS ReceivingReturnedItems
IN HYPERCLASS Sale
PRE-CONDITION -- A sale order without item returned info
   ∃ so ∈ SALE ORDER | (so.itemReturnedTime is NULL)
   ^ (so.paidTime is NOT NULL)
POST-CONDITION -- A sale order with item returned info
   (so.itemReturnedTime is NOT NULL)
PRE-HYPERCLASS SO_amount_paid, Sale_order_item,
   Inventory_item_activated
POST-HYPERCLASS SO_item_returned, Sale_order_item,
   Inventory_item_in_stock
TRIGGERED BY ReturnItems
EXECUTE ReceivingReturnedItems, UndoDecreasingQuantityInStock

Specification of the Integrity rule aspect

-- Integrity rules

INTEGRITY RULE IR#1
EXPRESSION
   -- The sale order items must be the items that
   -- currently sold by the store
   ∀ so_i ∈ SALE_ORDER_ITEM,
   ∃! itm ∈ INVENTORY_ITEM | so_i.SO_Item = itm.ItemCode
ON TUPLE-CLASS Sale order Item
RISK R#01 DEFINED ON Create.
CONCERNED BY ReceivingDemand
ON TUPLE-CLASS Inventory Item
RISK R#02 DEFINED ON Delete
INTEGRITY RULE IR#2
EXPRESSION
-- The amount of a sale order item is equal to
-- its unit price * quantity ordered
-- (i.e. SO Item amount=unit price * quantity)
∀ so_i ∈ SALE_ORDER_ITEM,
∃! itm ∈ INVENTORY_ITEM ∧
   (so_i.SO_Item = itm.ItemCode) ∧
   (so_i.SO_ItemAmount = so_i.QuantityOrdered *
    itm.ItemUnitPrice)
ON TUPLE-CLASS Sale order Item
RISK R#04 DEFINED ON Create
   CONCERNED BY ReceivingDemand
RISK R#05 DEFINED ON Update
   ON ATTRIBUTE SO Item Amount
   CONCERNED BY CheckingStock
RISK R#06 DEFINED ON #Update
   ON ATTRIBUTE Quantity Ordered
   CONCERNED BY ReceivingDemand

ON TUPLE-CLASS Inventory Item
RISK R#07 DEFINED ON Update
   ON ATTRIBUTE Item Unit Price

INTEGRITY RULE IR#3
EXPRESSION
-- The total amount of an sale order is equal to the
-- total of all its line item amount
∀ so ∈ SALE_ORDER,
∃! (so_i, so_i, ..., so_i), so_i, ∈ SALE ORDER ITEM|,
   (so.SaleOrderNumber = so_i.SO_Item) ∧
   (so.ToTotalAmount = Σ so_i.SO_ItemAmount)
ON TUPLE-CLASS Sale order
RISK R#08 DEFINED ON Create
   CONCERNED BY ReceivingDemand
RISK R#09 DEFINED ON #Update
   ON ATTRIBUTE Sale Order Amount
   CONCERNED BY CheckingStock
ON TUPLE-CLASS Sale order Item
RISK R#10 DEFINED ON #Create
   CONCERNED BY ReceivingDemand
RISK R#11 DEFINED ON #Update
   ON ATTRIBUTE SO Item Amount
   CONCERNED BY CheckingStock

INTEGRITY RULE IR#4
EXPRESSION
-- The total amount of the payments of a sale order
-- must not exceed the total amount of that sale order
∀ so ∈ SALE_ORDER,
∃! (py, py, ..., py), py, ∈ PAYMENT|
   (so.SaleOrderNumber = py.PaymentFor) ∧
   (so.ToTotalAmount >= Σ py.PaymentAmount)
ON TUPLE-CLASS Sale order
  RISK R#12 DEFINED ON Update
    ON ATTRIBUTE Sale Order Amount
    CONCERNED BY CheckingStock

ON TUPLE-CLASS Payment_approved
  RISK R#13 DEFINED ON Create
  CONCERNED BY ApprovingPayment
  RISK R#14 DEFINED ON Update
    ON ATTRIBUTE Payment Amount
    CONCERNED BY ApprovingPayment

INTEGRITY RULE IR#5
  EXPRESSION
    -- A sale order must have valid customer info before
    -- delivering items
    ∀ so ∈ SALE_ORDER |
      so.DeliveredTime is not NULL
      ⇒ so.belongedToCustomer is not NULL

SCOPE OF RULE IR#5 ON TUPLE-CLASS Sale order
  RISK R#15 DEFINED ON Update
    ON ATTRIBUTE Delivered time
    CONCERNED BY GettingCustomerInfo
  RISK R#16 DEFINED ON Create
    ON ATTRIBUTE Belong to customer
    CONCERNED BY GettingCustomerInfo

-- Method actions

-- Method of the Sale order tuple-class
METHOD ReceivingDemand OF HYPERCLASS Sale order
  ACTIONS create, update
  CONCERN R#01, R#04, R#06, R#08, R#10
METHOD CheckingStock OF HYPERCLASS Sale order
  ACTIONS update
  CONCERN R#05, R#09, R#11, R#12
METHOD GettingCustomerInfo OF HYPERCLASS Sale order
  ACTIONS update
  CONCERN R#15, R#16
METHOD ApprovingPayment OF HYPERCLASS Payment
  ACTIONS create
  CONCERN R#13, R#14
Annex 2.
META-MODEL OF THE M7 REFERENCE MODEL

This annex illustrates the meta-model of the Static, Dynamic, and Integrity rule aspects of the M7 reference model.
For each aspect, we present the meta-model and the conforming rules of this meta-model.

The Static aspect

Meta-model of the Static aspect

The meta-model of the Dynamic aspect has been represented in Figure 8-1.

![Figure 8-1: The meta-model of the Static aspect.]

There are the atomic-classes such as OID, Name, Integer and String:

- **OID** is a set of objects representing object identifiers;
- **NAME** is a set of objects representing object names;
- **Integer** and **String** are primitive classes representing base types supported by operational level.

Besides, the tuple-classes representing the specifications of the key concepts and their interrelationships of the Static aspect are as the followings:

- **Base type** is a tuple-class containing the set of objects, which represents the base types supported by the computerized environment. In general, objects of
that class are \textit{Integer}, \textit{Real}, \textit{Character}, \textit{Date} and \textit{Boolean}. There is only one attribute of \textit{Base type}: \textit{Base type OID}.

- \textbf{Category} is a tuple-class representing the generalisation of M7’s basic elements such as atomic-classes and hyperclasses. \textit{Category} has two attributes: \textit{Category OID} and \textit{Category Name}. \textit{Category Name} is also a logical identifier of that tuple-class.

- \textbf{Atomic-class} is a specialisation of the \textit{Category} tuple-class. Therefore, it inherits the attributes \textit{Category OID} and \textit{Category Name} of its super-hyperclass. In some cases, these attributes can be redefined as \textit{Atomic-class OID} and \textit{Atomic-class Name}. That tuple-class has three attributes: \textit{Atomic-class Base type}, \textit{Field size} and \textit{Decimal}.

- In a similar way, \textbf{Hyperclass} is a specialisation of the \textit{Category} tuple-class and inherits the attributes \textit{Category OID} and \textit{Category Name} of its super-hyperclass. These attributes can also be redefined as \textit{Hyperclass OID} and \textit{Hyperclass Name}.

- \textbf{Tuple-class} is a specialisation of the \textit{Hyperclass} tuple-class. At this moment, this class has no attribute. Afterward, it can be considered to expand to represent the information about implementation choice at operational level.

- \textbf{Sub-hyperclass} is also a specialisation of the \textit{Hyperclass} tuple-class. That tuple-class has two attributes:
  - \textit{Sub-hyperclass of} representing the identifier of its super-hyperclass;
  - \textit{Specialisation condition} representing the condition to select objects of the class from objects of its super-hyperclass.

- \textbf{Attribute} is a tuple-class that represents the correspondences of each object of the origin hyperclass to a set of objects of the termination category. Therefore, it has the attributes such as \textit{Origin hyperclass} and \textit{Termination category}. Furthermore, it also has attributes to represent its name and OID: \textit{Attribute OID} and \textit{Attribute Name}.

- \textbf{Key} is a tuple-class that represents the logical identifier of a hyper-object in its hyperclass. Its attributes are:
  - \textit{Key name} representing its name,
  - \textit{Key of hyperclass} representing the OID of its hyperclass, and
  - \textit{Key attributes} representing a set of attributes used to define this key.

\section*{Conformity rules}

The Table 8-1 presents the proposed validity rules of the Static aspect.

\begin{table}[h]
\begin{tabular}{|l|l|l|}
\hline
\textbf{Rule} & \textbf{Concept} & \textbf{Description} \\
\hline
S_V\#1 & Atomic-class & The \textit{Atomic-class-Name} of an atomic-class must be a valid name. \\
S_V\#2 & & The \textit{Base-type} of an atomic-class must be a valid base type. \\
S_V\#3 & & The \textit{Field-Size} of an atomic-class must be a positive integer. \\
S_V\#4 & & The \textit{Decimal} of an atomic-class must be a positive integer or zero. \\
\hline
\end{tabular}
\end{table}
Table 8-1: The validity rules of the Static aspect.

The Table 8-2 presents the proposed completeness rules of the Static aspect.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S_C#1</td>
<td>Atomic-class</td>
<td>For each atomic-class, there exists at least one attribute whose termination class is this atomic class.</td>
</tr>
<tr>
<td>S_C#2</td>
<td>Hyperclass</td>
<td>Each hyperclass, which is a tuple-class and not a sub-hyperclass of other hyperclass, must have at least one attribute whose termination class is an atomic class.</td>
</tr>
<tr>
<td>S_C#3</td>
<td>Hyperclass</td>
<td>Each hyperclass, which is not a sub-hyperclass of other hyperclass, must have at least one attribute whose termination class is a tuple class.</td>
</tr>
<tr>
<td>S_C#4</td>
<td>Hyperclass</td>
<td>Each hyperclass must have all “required” attributes, keys, sub-hyperclasses, dynamic states and methods.</td>
</tr>
<tr>
<td>S_C#5</td>
<td>Key</td>
<td>A key must have at least one attribute.</td>
</tr>
<tr>
<td>S_C#6</td>
<td>Key</td>
<td>A key must define all its “required” attributes.</td>
</tr>
</tbody>
</table>

Table 8-2: The completeness rules of the Static aspect.

The Dynamic aspect

Meta-model of the Dynamic aspect

The meta-model of the Dynamic aspect has been represented in Figure 8-2.
In *Figure 8-2*, there are the tuple-classes, which are representing the key concepts of the Dynamic aspect and their interrelationships such as:

- **Dynamic state** is a tuple-class containing the set of objects, which represents the dynamic states of IS. In the M7 method, a dynamic state is represented by a sub-hyperclass. Therefore, the *Dynamic state* tuple-class is a sub-hyperclass of the *Sub-hyperclass* tuple-class.

- **Method** is a tuple-class representing the methods used to transit between the dynamic states of a hyperclass. The *Method* tuple-class has the following attributes:
  - *Method OID* representing its object identifier,
  - *Method name* representing its name,
  - *Method Of Hyperclass* representing the hyperclass to which it is belonged,
  - *From States* representing a set of dynamic states before invoking the method,
  - *To States* representing a set of dynamic states after invoking the method, and
  - *Involve Attributes* representing a set of attributes in which the method may involve.

- **Event** is a tuple-class representing the business events that may be happened to the information system. That tuple-class has two attributes: *Event OID* and *Event name*.

- **Process** is a tuple-class containing the set of objects representing the processes, which may be triggered by business events. It has the following attributes:
  - *Process OID* representing its object identifier,
  - *Process name* representing its name,
  - *Triggered By* representing the event that may trigger it,
category of hyperclass representing the hyperclass to which the process belonged,
- Pre-hyperclasses representing a set of hyperclasses whose objects can be accessed by the process,
- Post-hyperclasses representing a set of hyperclasses whose objects are associated or disassociated by the process,
- Pre-condition and Post-condition representing the conditions must be true before / after performing the process,
- Execute methods representing a set of methods that may be invoked by the process.

Conformity rules

Similar to the meta-model of the Static aspect, the meta-model of the Dynamic aspects have also its own conformity rules, including the validity rules (Table 8.3) and the completeness rules (Table 8.4).

<table>
<thead>
<tr>
<th>Rule</th>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_V#1</td>
<td>Method</td>
<td>The Method-name of a method must be a valid name.</td>
</tr>
<tr>
<td>D_V#2</td>
<td>Each attribute involved in a method must have its origin hyperclass as the hyperclass to which the method belonged, or one of its descendants.</td>
<td></td>
</tr>
<tr>
<td>D_V#3</td>
<td>The sub-hyperclass representing a dynamic state participated in the FromStates or ToStates lists of a method must be the hyperclass to which the method is belonged, or one of its descendants, or one of its attributes.</td>
<td></td>
</tr>
<tr>
<td>D_V#4</td>
<td>Event</td>
<td>The name of an event must be a valid name.</td>
</tr>
<tr>
<td>D_V#5</td>
<td>Process</td>
<td>The Process-name of a process must be a valid name.</td>
</tr>
<tr>
<td>D_V#6</td>
<td>The Pre-condition and Post-condition of a process must be a valid Boolean condition.</td>
<td></td>
</tr>
<tr>
<td>D_V#7</td>
<td>A pre-hyperclass or a post-hyperclass of a process must belong to the hyperclass to which the process defined, or one of its attributes, or one its descendants.</td>
<td></td>
</tr>
<tr>
<td>D_V#8</td>
<td>A process must execute methods belonged to hyperclasses, which are the hyperclass to which the process belonged, or one of its descendants, or one of its attributes.</td>
<td></td>
</tr>
</tbody>
</table>

Table 8.3: The validity rules of the Dynamic aspect.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D_C#1</td>
<td>Dynamic state</td>
<td>A dynamic state is concerned by at least one method (in the fromStates or toStates lists of this method).</td>
</tr>
<tr>
<td>D_C#2</td>
<td>Method</td>
<td>Each method must have at least one dynamic state, participated in the fromStates list. Similarity, each method must also have at least one dynamic state, participated in the toStates list.</td>
</tr>
<tr>
<td>D_C#3</td>
<td>A method is executed by at least one process.</td>
<td></td>
</tr>
<tr>
<td>D_C#4</td>
<td>A method must have all its involved attributes, its related dynamic states (in the fromStates and toStates lists), and its concerned risks.</td>
<td></td>
</tr>
</tbody>
</table>
**The Integrity rule aspect**

**Meta-model of the Integrity rule aspect**

The meta-model of the Integrity rule aspect has been represented in Figure 8-3.

The tuple-classes representing the key concepts of the Integrity rule aspect are as the followings:

- **Integrity rule** is a tuple-class containing the set of objects, which represents integrity rules (IR) of the IS. In the M7 method, an IR is identified by its *OID*, specified by its *name* and *expression*.

- **Scope** is a tuple-class denoting scopes of IRs. Scopes of an IR indicates a set of tuple-classes to which the IR has been defined. That tuple-class have three attributes:
  - *Scope OID* represent its object identifier,
  - *Scope of Rule* representing the IR on which the scope is defined,
  - *On Tuple-class* representing the corresponding tuple-class.
- **Primitive** is a tuple-class including a set of objects representing the primitives, which are defined as basic operations on tuple-classes. That class has two attributes:
  - *Primitive OID* representing its identifier, and
  - *Primitive type* representing its type (such as create, delete and update). Besides, a method can perform several primitives such as creating, updating or deleting a set of objects. These actions are called *Actions* of method.
- **Risk** is a tuple-class containing the set of objects representing the risks of IRs. A risk is a situation in which an IR may be violated. Actually, a scope of an IR includes a set of risks that may violate this IR. That tuple-class has the following attributes:
  - *Risk OID* representing its identifier,
  - *Risk Of Scope* indicating its belonged scope,
  - *Defined On Primitive* indicating its primitive, and

### Conformity rules

Similar to the meta-model of the Static and the Dynamic aspects, the meta-model of the Integrity rule aspect have its own conformity rules, including the validity rules (Table 8-5) and the completeness rules Table 8-6).

<table>
<thead>
<tr>
<th>Rule</th>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_V#1</td>
<td>Integrity Rule</td>
<td>The <em>Rule-name</em> of an IR must be a valid name.</td>
</tr>
<tr>
<td>R_V#2</td>
<td>Integrity Rule</td>
<td>The <em>Rule-Expression</em> of an IR must be a valid expression.</td>
</tr>
<tr>
<td>R_V#3</td>
<td>Scope</td>
<td>A scope must be defined by to one and only pair, including a tuple class and an IR.</td>
</tr>
<tr>
<td>R_V#4</td>
<td>Risk</td>
<td>A risk must be related to one and only one pair, including a scope and a primitive.</td>
</tr>
<tr>
<td>R_V#5</td>
<td></td>
<td>If primitive of a risk is UPDATE, its <em>riskOnAttribute</em> attribute must be declared.</td>
</tr>
<tr>
<td>R_V#6</td>
<td></td>
<td>An attribute related to a risk must have its origin hyperclass as the tuple-class on which the risk is defined.</td>
</tr>
<tr>
<td>R_V#7</td>
<td>Method</td>
<td>A method of a hyperclass may concern to a risk of a scope defined on a tuple-class that is also this hyperclass.</td>
</tr>
<tr>
<td>R_V#8</td>
<td></td>
<td>For each risk concerned by a method, its defined primitive must be found in a list of <em>actions</em> primitives of the method.</td>
</tr>
<tr>
<td>R_V#9</td>
<td></td>
<td>An attribute related to a risk must be found in a list of involved attributes of a method, which concerns to this risk.</td>
</tr>
</tbody>
</table>

*Table 8-5: The validity rules of the Integrity rule aspect.*
<table>
<thead>
<tr>
<th>Rule</th>
<th>Concept</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_C#1</td>
<td>Rule</td>
<td>An IR must have at least one scope</td>
</tr>
<tr>
<td>R_C#2</td>
<td>Rule</td>
<td>An IR must declare all its scopes and risks</td>
</tr>
<tr>
<td>R_C#3</td>
<td>Scope</td>
<td>A scope must have at least one risk</td>
</tr>
<tr>
<td>R_C#4</td>
<td>Scope</td>
<td>A scope must have all its risks</td>
</tr>
<tr>
<td>R_C#5</td>
<td>Method</td>
<td>A method must declare all its concerned risks.</td>
</tr>
<tr>
<td>R_C#6</td>
<td>Method</td>
<td>A method must declare all its actions primitives.</td>
</tr>
</tbody>
</table>

*Table 8-6: The completeness rules of the Integrity rule aspect.*
Annex 3.
COORDINATION OF OBJECT LIFE CYCLES OF GENERIC HYPERCLASSES

This annex presents the impact-analysis tables and coordination rules concerning the impact of changes of dynamic states of objects of the generic hyperclass according to the Static, Dynamic and Integrity rule aspects.

**ATOMIC-CLASS generic hyperclass**

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOMIC-CLASS</td>
<td>Invalid</td>
<td>Cause</td>
<td>Isa-AtomicClass, terminationCategory</td>
<td>ATTRIBUTE</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td>UnCompleted</td>
<td>Cause</td>
<td>Isa-AtomicClass, terminationCategory</td>
<td>ATTRIBUTE</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

*Table 8-7: Impact-analysis table concerning the changes of dynamic states of objects of Atomic-class generic hyperclass.*

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic-class</td>
<td>Invalid</td>
<td>1</td>
<td>If an object of Atomic-class generic hyperclass is in <em>Invalid</em> state then the corresponding objects of Attribute generic hyperclass (linked via the Isa-Atomic-class and Termination category) must also be in <em>Invalid</em> state.</td>
</tr>
<tr>
<td></td>
<td>Uncompleted</td>
<td>2</td>
<td>If an object of Atomic-class generic hyperclass is in <em>Uncompleted</em> state then the corresponding objects of Attribute generic hyperclass must also be in <em>Uncompleted</em> state.</td>
</tr>
</tbody>
</table>

*Table 8-8: Coordination rules concerning the Atomic-class generic hyperclass.*
HYPERCLASS generic hyperclass

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>HYPERCLASS</td>
<td>Valid</td>
<td>Implication</td>
<td>IsSubHyperclassOf (the superHyperclass)</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IsSubHyperclassOf (the subHyperclass)</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>originHyperclass</td>
<td>ATTRIBUTE</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key-of-Hyperclass</td>
<td>KEY</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IsA-SubHyperclass, IsA-DynamicState</td>
<td>DYNAMIC STATE</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Method-of-Hyperclass</td>
<td>METHOD</td>
<td>Valid</td>
</tr>
<tr>
<td>HYPERCLASS</td>
<td>Invalid</td>
<td>Cause</td>
<td>IsSubHyperclassOf (the subHyperclass)</td>
<td>HYPERCLASS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>originHyperclass</td>
<td>ATTRIBUTE</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IsA-Hyperclass, terminationCategory</td>
<td>ATTRIBUTE</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key-of-Hyperclass</td>
<td>KEY</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Process-of-Hyperclass</td>
<td>PROCESS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PreHyperclass</td>
<td>PROCESS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>PostHyperclass</td>
<td>PROCESS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IsA-SubHyperclass, IsA-DynamicState</td>
<td>DYNAMIC STATE</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Method-of-Hyperclass</td>
<td>METHOD</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IsA-Tupleclass, OnTupleclass</td>
<td>SCOPE</td>
<td>Invalid</td>
</tr>
<tr>
<td>HYPERCLASS</td>
<td>Completed</td>
<td>Implication</td>
<td>IsSubHyperclassOf (the subHyperclass)</td>
<td>HYPERCLASS</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>originHyperclass</td>
<td>ATTRIBUTE</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Key-of-Hyperclass</td>
<td>KEY</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>IsA-SubHyperclass, IsA-DynamicState</td>
<td>DYNAMIC STATE</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Method-of-Hyperclass</td>
<td>METHOD</td>
<td>Completed</td>
</tr>
<tr>
<td>HYPERCLASS</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>IsSubHyperclassOf (the superHyperclass)</td>
<td>HYPERCLASS</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

Table 8-9: Impact-analysis table concerning the changes of dynamic states of objects of Hyperclass generic hyperclass.
<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hyperclass</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding object representing its superHyperclass (linked via the sub-hyperclass-of attribute) must also be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding objects representing its subHyperclasses (linked via the sub-hyperclass-of and IsA-SubHyperclass attributes) must also be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding objects of the Attribute generic hyperclass (linked via the origin-hyperclass) must also be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding objects of the Key generic hyperclass (linked via the key-of-Hyperclass attribute) must also be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding objects of the Dynamic State generic hyperclass (linked via the IsA-Dynamic-state and IsA-SubHyperclass-of attributes) must also be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>If an object of Hyperclass generic hyperclass is in Valid state then the corresponding objects of the Method generic hyperclass (linked via the method-of-Hyperclass attribute) must also be in Valid state.</td>
</tr>
<tr>
<td>Hyperclass</td>
<td>Invalid</td>
<td>7</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding object representing its superHyperclass (linked via the sub-hyperclass-of attribute) must also be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding objects representing its subHyperclasses (linked via the sub-hyperclass-of and IsA-SubHyperclass attributes) must also be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding objects of the Attribute generic hyperclass (linked via the origin-hyperclass attribute) must also be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>10</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding objects of the Attribute generic hyperclass (linked via the IsA-Hyperclass and terminationCategory attributes) must also be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>11</td>
<td>If an object of Hyperclass generic hyperclass is in Invalid state then the corresponding objects of the Key generic hyperclass (linked via the key-of-Hyperclass attribute) must also be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Process generic hyperclass (linked via the <em>process-of-Hyperclass</em> attribute) must also be in <em>Invalid</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Process generic hyperclass (linked via the <em>pre-Hyperclass</em> attribute) must also be in <em>Invalid</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Process generic hyperclass (linked via the <em>post-Hyperclass</em> attribute) must also be in <em>Invalid</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Dynamic State generic hyperclass (linked via the <em>Is-A-Dynamic-state</em> and <em>Is-Sub-Hyperclass-of</em> attributes) must also be in <em>Invalid</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Method generic hyperclass (linked via the <em>method-of-Hyperclass</em> attribute) must also be in <em>Invalid</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Scope generic hyperclass (linked via the <em>Is-A-Tupleclass</em> and <em>On-Tupleclass</em> attributes) must also be in <em>Invalid</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hyperclass</td>
<td>Completed</td>
<td>18</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Completed</em> state then the corresponding objects representing its subHyperclasses (linked via the <em>sub-Hyperclass-of</em> and <em>Is-A-Sub-Hyperclass</em> attributes) must also be in <em>Completed</em> state.</td>
</tr>
<tr>
<td>19</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Completed</em> state then the corresponding objects of the Attribute generic hyperclass (linked via the <em>origin-hyperclass</em> attribute) must also be in <em>Completed</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Completed</em> state then the corresponding objects of the Key generic hyperclass (linked via the <em>key-of-Hyperclass</em> attribute) must also be in <em>Completed</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Completed</em> state then the corresponding objects of the Dynamic State generic hyperclass (linked via the <em>Is-A-Dynamic-state</em> and <em>Is-Sub-Hyperclass-of</em> attributes) must also be in <em>Completed</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>If an object of Hyperclass generic hyperclass is in <em>Completed</em> state then the corresponding objects of the Method generic hyperclass (linked via the <em>method-of-Hyperclass</em> attribute) must also be in <em>Completed</em> state.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Hyperclass

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIBUTE</td>
<td>Valid</td>
<td>Implication</td>
<td>OriginHyperclass</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Isa-Atomic-class, terminationCategory</td>
<td>ATOMIC-CLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Isa-Hyperclass, terminationCategory</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>Invalid</td>
<td>Cause</td>
<td>OriginHyperclass</td>
<td>HYPERCLASS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KeyAttributes</td>
<td>KEY</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>InvolveAttributes</td>
<td>METHOD</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>RiskOnAttribute</td>
<td>RISK</td>
<td>Invalid</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>Completed</td>
<td>Implication</td>
<td>RiskOnAttribute</td>
<td>RISK</td>
<td>Completed</td>
</tr>
<tr>
<td>ATTRIBUTE</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>OriginHyperclass</td>
<td>HYPERCLASS</td>
<td>Uncompleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>KeyAttributes</td>
<td>KEY</td>
<td>Uncompleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>InvolveAttributes</td>
<td>METHOD</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

### Table 8-10: Coordination rules concerning the Hyperclass generic hyperclass.

### ATTRIBUTE generic hyperclass

### Table 8-11: Impact-analysis table concerning the changes of dynamic states of objects of Attribute generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Attribute</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Attribute generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass representing its originHypattribute must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>If an object of Attribute generic hyperclass is in Valid state then the corresponding object of the Atomic-class generic hyperclass representing its termination category (linked via the Isa-Atomic-class and terminationCategory attributes) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>If an object of Attribute generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass representing its termination category (linked via the Isa-Hyperclass and terminationCategory attributes) must be in Valid state.</td>
</tr>
<tr>
<td>Invalid</td>
<td>4</td>
<td>If an object of Attribute generic hyperclass is in <em>Invalid</em> state then the corresponding object of the Hyperclass generic hyperclass representing its originated hyperclass (linked via the <em>originHyperclass</em> attribute) must be in <em>Invalid</em> state.</td>
<td></td>
</tr>
<tr>
<td>---------</td>
<td>---</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5</td>
<td>If an object of Attribute generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Key generic hyperclass (linked via the <em>keyAttributes</em> attribute) must be in <em>Invalid</em> state.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>6</td>
<td>If an object of Attribute generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Method generic hyperclass (linked via the <em>involveAttributes</em> attribute) must be in <em>Invalid</em> state.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>If an object of Attribute generic hyperclass is in <em>Invalid</em> state then the corresponding objects of the Risk generic hyperclass (linked via the <em>riskOnAttribute</em> attribute) must be in <em>Invalid</em> state.</td>
<td></td>
</tr>
<tr>
<td>Completed</td>
<td>If an object of Attribute generic hyperclass is in <em>Completed</em> state then the corresponding objects of the Risk generic hyperclass (linked via the <em>riskOnAttribute</em> attribute) must be in <em>Completed</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Uncompleted</td>
<td>8</td>
<td>If an object of Attribute generic hyperclass is in <em>Uncompleted</em> state then the corresponding object of the Hyperclass generic hyperclass representing its originated hyperclass (linked via the <em>originHyperclass</em> attribute) must be in <em>Uncompleted</em> state.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>If an object of Attribute generic hyperclass is in <em>Uncompleted</em> state then the corresponding objects of the Key generic hyperclass (linked via the <em>keyAttributes</em> attribute) must be in <em>Uncompleted</em> state.</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>If an object of Attribute generic hyperclass is in <em>Uncompleted</em> state then the corresponding objects of the Method generic hyperclass (linked via the <em>involveAttributes</em> attribute) must be in <em>Uncompleted</em> state.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Table 8-12: Coordination rules concerning the Attribute generic hyperclass.*

**KEY generic hyperclass**

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>KEY</td>
<td><em>Valid</em></td>
<td>Implication</td>
<td>key-of-Hyperclass</td>
<td>HYPERCLASS</td>
<td><em>Valid</em></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>keyAttributes</td>
<td>ATTRIBUTE</td>
<td></td>
</tr>
<tr>
<td>KEY</td>
<td><em>Invalid</em></td>
<td>Cause</td>
<td>key-of-Hyperclass</td>
<td>HYPERCLASS</td>
<td><em>Invalid</em></td>
</tr>
<tr>
<td>KEY</td>
<td><em>Completed</em></td>
<td>Implication</td>
<td>keyAttributes</td>
<td>ATTRIBUTE</td>
<td><em>Completed</em></td>
</tr>
</tbody>
</table>
Information System Upon Information Systems

<table>
<thead>
<tr>
<th>KEY</th>
<th>Uncompleted</th>
<th>Cause</th>
<th>key-of-Hyperclass</th>
<th>HYPERCLASS</th>
<th>Uncompleted</th>
</tr>
</thead>
</table>

Table 8-13: Impact-analysis table concerning the changes of dynamic states of objects of Key generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Key generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass (linked via the key-of-Hyperclass attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>If an object of Key generic hyperclass is in Valid state then the corresponding objects of the Attribute generic hyperclass (linked via the key-Attributes attribute) must be in Valid state.</td>
</tr>
<tr>
<td>Invalid</td>
<td></td>
<td>3</td>
<td>If an object of Key generic hyperclass is in Invalid state then the corresponding object of the Hyperclass generic hyperclass (linked via the key-of-Hyperclass attribute) must be in Invalid state.</td>
</tr>
<tr>
<td>Completed</td>
<td></td>
<td>4</td>
<td>If an object of Key generic hyperclass is in Completed state then the corresponding objects of the Attribute generic hyperclass (linked via the key-Attributes attribute) must be in Completed state.</td>
</tr>
<tr>
<td>Uncompleted</td>
<td></td>
<td>5</td>
<td>If an object of Key generic hyperclass is in Uncompleted state then the corresponding object of the Hyperclass generic hyperclass (linked via the key-of-Hyperclass attribute) must be in Uncompleted state.</td>
</tr>
</tbody>
</table>

Table 8-14: Coordination rules concerning the Key generic hyperclass.

EVENT generic hyperclass

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>EVENT</td>
<td>Completed</td>
<td>Implication</td>
<td>triggeredBy</td>
<td>PROCESS</td>
<td>Completed</td>
</tr>
</tbody>
</table>

Table 8-15: Impact-analysis table concerning the changes of dynamic states of objects of Event generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Event</td>
<td>Completed</td>
<td>1</td>
<td>If an object of Event generic hyperclass is in Completed state then the corresponding object of the Process generic hyperclass (linked via the triggeredProcess attribute) must be in Completed state.</td>
</tr>
</tbody>
</table>

Table 8-16: Coordination rules concerning the Event generic hyperclass.
**PROCESS generic hyperclass**

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROCESS</td>
<td>Valid</td>
<td>Implication</td>
<td>process-of-Hyperclass</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>preHyperclass</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>postHyperclass</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>executeMethods</td>
<td>METHOD</td>
<td>Completed</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Completed</td>
<td>Implication</td>
<td>executeMethods</td>
<td>METHOD</td>
<td>Completed</td>
</tr>
<tr>
<td>PROCESS</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>triggeredProcess</td>
<td>EVENT</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

Table 8-17: Impact-analysis table concerning the changes of dynamic states of objects of Process generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Process</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Process generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass (linked via the process-of-Hyperclass attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>If an object of Process generic hyperclass is in Valid state then the corresponding objects of the Hyperclass generic hyperclass (linked via the preHyperclasses attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>If an object of Process generic hyperclass is in Valid state then the corresponding objects of the Hyperclass generic hyperclass (linked via the postHyperclasses attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>If an object of Process generic hyperclass is in Completed state then the corresponding objects of the Method generic hyperclass (linked via the executeMethods attribute) must be in Completed state.</td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td>4</td>
<td>If an object of Process generic hyperclass is in Completed state then the corresponding objects of the Method generic hyperclass (linked via the executeMethods attribute) must be in Completed state.</td>
</tr>
<tr>
<td></td>
<td>Uncompleted</td>
<td>5</td>
<td>If an object of Process generic hyperclass is in Uncompleted state then the corresponding object of the Event generic hyperclass (linked via the triggeredProcess attribute) must be in Uncompleted state.</td>
</tr>
</tbody>
</table>

Table 8-18: Coordination rules concerning the Process generic hyperclass.
DYNAMIC STATE generic hyperclass

<table>
<thead>
<tr>
<th>Generic</th>
<th>Dynamic</th>
<th>Impact</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>hyperclass</td>
<td>states</td>
<td>types</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DYNAMIC STATE</td>
<td>Valid</td>
<td>Implication</td>
<td>IsADynamicState, SubHyperclass-of</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td>DYNAMIC STATE</td>
<td>Invalid</td>
<td>Cause</td>
<td>IsADynamicState, SubHyperclass-of</td>
<td>HYPERCLASS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fromStates METHOD</td>
<td>Invalid</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>toStates METHOD</td>
<td>Uncompleted</td>
<td></td>
</tr>
<tr>
<td>DYNAMIC STATE</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>IsADynamicState, SubHyperclass-of</td>
<td>HYPERCLASS</td>
<td>Uncompleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fromStates METHOD</td>
<td>Uncompleted</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>toStates METHOD</td>
<td>Uncompleted</td>
<td></td>
</tr>
</tbody>
</table>

Table 8-19: Impact-analysis table concerning the changes of dynamic states of objects of Dynamic state generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic state</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Dynamic state generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass (linked via the IsADynamicState and subHyperclass-of attributes) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td>Invalid</td>
<td>2</td>
<td>If an object of Dynamic state generic hyperclass is in Invalid state then the corresponding object of the Hyperclass generic hyperclass (linked via the IsADynamicState and subHyperclass-of attributes) must be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>If an object of Dynamic state generic hyperclass is in Invalid state then the corresponding objects of the Method generic hyperclass (linked via the fromStates attribute) must be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
<td>If an object of Dynamic state generic hyperclass is in Invalid state then the corresponding objects of the Method generic hyperclass (linked via the toStates attribute) must be in Invalid state.</td>
</tr>
<tr>
<td>Uncompleted</td>
<td></td>
<td>5</td>
<td>If an object of Dynamic state generic hyperclass is in Uncompleted state then the corresponding object of the Hyperclass generic hyperclass (linked via the IsADynamicState and subHyperclass-of attributes) must be in Uncompleted state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6</td>
<td>If an object of Dynamic state generic hyperclass is in Uncompleted state then the corresponding objects of the Method generic hyperclass (linked via the fromStates attribute) must be in Uncompleted state.</td>
</tr>
</tbody>
</table>
Table 8-20: Coordination rules concerning the Dynamic state generic hyperclass.

### METHOD generic hyperclass

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>METHOD</td>
<td>Valid</td>
<td>Implication</td>
<td>Method-of-Hyperclass</td>
<td>HYPERCLASS</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>involveAttributes</td>
<td>ATTRIBUTE</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fromStates</td>
<td>DYNAMIC STATE</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>toStates</td>
<td>DYNAMIC STATE</td>
<td>Valid</td>
</tr>
<tr>
<td>METHOD</td>
<td>Invalid</td>
<td>Cause</td>
<td>Method-of-Hyperclass</td>
<td>HYPERCLASS</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>executeMethods</td>
<td>PROCESS</td>
<td>Invalid</td>
</tr>
<tr>
<td>METHOD</td>
<td>Completed</td>
<td>Implication</td>
<td>involveAttributes</td>
<td>ATTRIBUTE</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>fromStates</td>
<td>DYNAMIC STATE</td>
<td>Completed</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>toStates</td>
<td>DYNAMIC STATE</td>
<td>Completed</td>
</tr>
<tr>
<td>METHOD</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>executeMethods</td>
<td>PROCESS</td>
<td>Uncompleted</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Method-of-Hyperclass</td>
<td>HYPERCLASS</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

Table 8-21: Impact-analysis table concerning the changes of dynamic states of objects of Method generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Method</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Dynamic state generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass (linked via the method-of-Hyperclass attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>If an object of Dynamic state generic hyperclass is in Valid state then the corresponding objects of the Attribute generic hyperclass (linked via the involveAttributes attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td>Dynamic states</td>
<td>Impact attributes</td>
<td>Affected generic hyperclass</td>
</tr>
<tr>
<td>---</td>
<td>----------------</td>
<td>------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>3</td>
<td>Valid</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>4</td>
<td>Valid</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>5</td>
<td>Invalid</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>6</td>
<td>Invalid</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>7</td>
<td>Completed</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>8</td>
<td>Completed</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>9</td>
<td>Completed</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>10</td>
<td>Uncompleted</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
<tr>
<td>11</td>
<td>Uncompleted</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
</tr>
</tbody>
</table>

Table 8-22: Coordination rules concerning the Method generic hyperclass.

**INTEGRITY RULE generic hyperclass**
<table>
<thead>
<tr>
<th>Integrity RULE</th>
<th>Completed</th>
<th>Implication</th>
<th>Scope-of-Rule</th>
<th>SCOPE</th>
<th>Completed</th>
</tr>
</thead>
</table>

**Table 8-23: Impact-analysis table concerning the changes of dynamic states of objects of Integrity rule generic hyperclass.**

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity rule</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Dynamic state generic hyperclass is in Valid state then the corresponding objects of the Scope generic hyperclass (linked via the scope-of-Rule attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td>Invalid</td>
<td>2</td>
<td>If an object of Dynamic state generic hyperclass is in Invalid state then the corresponding objects of the Scope generic hyperclass (linked via the scope-of-Rule attribute) must be in Invalid state.</td>
</tr>
<tr>
<td></td>
<td>Completed</td>
<td>3</td>
<td>If an object of Dynamic state generic hyperclass is in Completed state then the corresponding objects of the Scope generic hyperclass (linked via the scope-of-Rule attribute) must be in Completed state.</td>
</tr>
</tbody>
</table>

**SCOPE generic hyperclass**

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>SCOPE</td>
<td>Valid</td>
<td>Implication</td>
<td>Scope-of-Rule</td>
<td>INTEGRITY RULE</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Risk-of-Scope</td>
<td>Valid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>IsA-Tupleclass, OnTupleclass</td>
<td>Valid</td>
</tr>
<tr>
<td>SCOPE</td>
<td>Invalid</td>
<td>Cause</td>
<td>Risk-of-Scope</td>
<td>RISK</td>
<td>Invalid</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Scope-of-Rule</td>
<td>INTEGRITY RULE</td>
<td>Invalid</td>
</tr>
<tr>
<td>SCOPE</td>
<td>Completed</td>
<td>Implication</td>
<td>Risk-of-Scope</td>
<td>RISK</td>
<td>Completed</td>
</tr>
<tr>
<td>SCOPE</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>Scope-of-Rule</td>
<td>INTEGRITY RULE</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

**Table 8-25: Impact-analysis table concerning the changes of dynamic states of objects of Scope generic hyperclass.**

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Scope generic hyperclass is in Valid state then the corresponding object of the Integrity rule generic hyperclass (linked via the scope-of-Rule attribute) must be in Valid state.</td>
</tr>
</tbody>
</table>
### Table 8-26: Coordination rules concerning the Scope generic hyperclass.

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>If an object of Scope generic hyperclass is in Valid state then the corresponding objects of the Risk generic hyperclass (linked via the risk-of-Scope attribute) must be in Valid state.</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>If an object of Scope generic hyperclass is in Valid state then the corresponding object of the Hyperclass generic hyperclass (linked via the Is-A-Tupleclass and OnTupleClass attributes) must be in Valid state.</td>
<td></td>
</tr>
<tr>
<td>Invalid</td>
<td>4</td>
<td>If an object of Scope generic hyperclass is in Invalid state then the corresponding object of the Integrity rule generic hyperclass (linked via the scope-of-Rule attribute) must be in Invalid state.</td>
</tr>
<tr>
<td>Completed</td>
<td>5</td>
<td>If an object of Scope generic hyperclass is in Completed state then the corresponding objects of the Risk generic hyperclass (linked via the risk-of-Scope attribute) must be in Completed state.</td>
</tr>
<tr>
<td>Uncompleted</td>
<td>7</td>
<td>If an object of Scope generic hyperclass is in Uncompleted state then the corresponding object of the Integrity rule generic hyperclass (linked via the scope-of-Rule attribute) must be in Uncompleted state.</td>
</tr>
</tbody>
</table>

### RISK generic hyperclass

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic states</th>
<th>Impact types</th>
<th>Via attributes</th>
<th>Affected generic hyperclass</th>
<th>Dynamic state</th>
</tr>
</thead>
<tbody>
<tr>
<td>RISK</td>
<td>Valid</td>
<td>Implication</td>
<td>riskOnAttribute</td>
<td>ATTRIBUTE</td>
<td>Valid</td>
</tr>
<tr>
<td>RISK</td>
<td>Invalid</td>
<td>Cause</td>
<td>Risk-of-Scope</td>
<td>SCOPE</td>
<td>Valid</td>
</tr>
<tr>
<td>RISK</td>
<td>Uncompleted</td>
<td>Cause</td>
<td>Risk-of-Scope</td>
<td>SCOPE</td>
<td>Uncompleted</td>
</tr>
</tbody>
</table>

### Table 8-27: Impact-analysis table concerning the changes of dynamic states of objects of Risk generic hyperclass.

<table>
<thead>
<tr>
<th>Generic hyperclass</th>
<th>Dynamic state</th>
<th>Rule #</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Risk</td>
<td>Valid</td>
<td>1</td>
<td>If an object of Risk generic hyperclass is in Valid state then the corresponding object of the Attribute generic hyperclass (linked via the riskOnAttribute attribute) must be in Valid state.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>If an object of Risk generic hyperclass is in Valid state then the corresponding object of the Scope generic hyperclass (linked via the <em>risk-of-Scope</em> attribute) must be in Valid state.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Invalid 3</td>
<td>If an object of Risk generic hyperclass is in Invalid state then the corresponding object of the Scope generic hyperclass (linked via the <em>risk-of-Scope</em> attribute) must be in Invalid state.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Uncompleted 4</td>
<td>If an object of Risk generic hyperclass is in Uncompleted state then the corresponding object of the Scope generic hyperclass (linked via the <em>risk-of-Scope</em> attribute) must be in Uncompleted state.</td>
<td></td>
</tr>
</tbody>
</table>

*Table 8-28: Coordination rules concerning the Risk generic hyperclass.*
REFERENCES


