

UNIVERSITÉ CATHOLIQUE DE LOUVAIN
Louvain School of Management

User Interface Generation from Task,
Domain and User Models: The DB-USE
approach

Thi Ai Vi TRAN

Thèse présentée en vue d'obtention
du grade de Docteur
en sciences économiques et de gestion

Membres du Jury :

Prof. Jean VANDERDONCKT (UCL), promoteur
Prof. Stéphane FAULKNER (FUNDP), Co-promoteur
Prof. Thierry VANDENBERGHE (ICHEC)
Dr. Costin PRIBEANU (ICI, Roumanie)
Prof. Christophe KOLSKI (Univ. de Valenciennes, France)
Prof. Loris PENSERINI (ITC-IRST)
Prof. Philippe CHEVALIER (UCL), président

Décembre 2010



User Interface Generation from Task, Domain and User Models: The DB-USE approach

Thi Ai Vi Tran

A thesis submitted in fulfillment of the requirements for the degree of
Doctor of Philosophy in
Economic and Management Sciences
of the Université catholique de Louvain
Louvain School of Management

Committee in charge:

Prof. Jean Vanderdonckt, Université catholique de Louvain, Belgium, Supervisor
Prof. Stéphane Faulkner, University of Namur, Belgium, Co-supervisor
Prof. Thierry Vandenberghe, ICHEC, Belgium, Examiner
Dr. Costin Pribeanu, National Institute for Research and Development in Informatics,
Romania, Examiner
Prof. Christophe Kolski, Université de Valenciennes, France, Examiner
Prof. Loris Penserini, ITC-IRST, Italy, Reader
Prof. Philippe Chevalier, Université catholique de Louvain, Belgium, President of jury

To my parents

Gửi tặng ba má

ACKNOWLEDGEMENTS

My PhD years at the Louvain School of Management, Université catholique de Louvain has had mentorship from numerous outstanding individuals both from within and outside the School. It is a pleasure to finally have the opportunity to thank here the many people who made this thesis possible.

I am first of all heartily thankful to my supervisors, Prof. Jean Vanderdonckt and Prof. Stéphane Faulkner.

It is difficult to overstate my gratitude to Jean. With his enthusiasm, inspiration, and great efforts to explain things clearly and simply, he has helped to make the discipline of human-computer interaction enjoying for me. Throughout my thesis-writing period, he/she provided encouragement, sound advice, good teaching, good company, and lots of interesting ideas. I would have been lost without him.

My gratitude also goes to Stéphane whose encouragement, guidance and support from the initial to the final level has enabled me to develop an understanding of the database and information systems aspects.

I would like to thank Prof. Philippe Chevalier for his help and guidance during my thesis.

I would also like to express my appreciation and regards to the members of my PhD jury, Profs. Thierry Vandenberghe, Costin Pribeanu, Christophe Kolski and Loris Penserini, for accepting to participate in the jury, for reviewing my thesis and for suggesting many useful and stimulating improvements.

Of course, my gratefulness also goes to my research companions in the ISYS unit, BCHI lab, and Translogistic project for their helpful discussions and support: Kenia Sousa, Juan Manuel Gonzales Calleros, Josephina Guerrero Garcia, Prof. Yves Wautelet, Sodany Kiv, Prof. Marco Saerens, Tai Nguyen, Thao Le, Hang Hoang, Youssef Achbany, Tung Do, Jean-Charles Lange, Prof. Xavier Brusset, and Prof. Pierre Semal.

Also thank to my vietnamese friends in Belgium for all of their help and support.

Last but not least, my thanks go to Marie-Claire Wadin and Lucy Cardols. I thank my friend Van Anh for her encouragement. I finally thank Jean-Pierre, and his wife My Anh, for reviewing the manuscript and helping me with the English difficulties.

I'm trying to think of a special way to thank my husband for the love and support he gave me while I was working on my thesis. I finally dedicate this research journey to my little boy Lucas who makes us laugh every day.

Je remercie tout spécialement mes beaux-parents pour leur support et leur engagement.

Đây là món quà con gửi tặng ba má vì những vất vả mà ba má đã trải qua để con có được ngày hôm nay. Ba má luôn là tấm gương và là động lực để con phấn đấu trong cuộc sống. Con luôn biết ơn ba má!

Cảm ơn anh Quang và chị Hiền đã luôn hỗ trợ và động viên em trong quá trình em làm luận án tiến sĩ này. Đồng thời, chị cũng gửi lời cảm ơn đến gia đình Tiên & Thụy Vy và Hòa & Thiện Vy vì sự quan tâm của các em dành cho chị. Chị rất vui vì luôn nhận được sự hỏi thăm của Tịnh Vy và Trịnh. Dì thương các công chúa Gia Quyên, Gia Hân, Gia Khanh, Vân Khánh của dì và dì cũng thương hoàng tử Khang lắm!

Cả nhà luôn là niềm vui và là chỗ dựa tinh thần của chị!

Publications

V. Tran, J. Vanderdonckt, M. Kolp, S. Faulkner, “Generating User Interface from Task, User and Domain Models”, Proc. of the Second International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services CENTRIC’09 (Porto, September 20-25, 2009), IEEE Computer Society Press, Los Alamitos, 2009, pp. 19-26.

V. Tran, J. Vanderdonckt, M. Kolp, Y. Wantelet, S. Faulkner, “Agent-Based User Interface Generation from Combined Task, Context and Domain models”, Proc. of 8th Int. Workshop on TAsk MOdels and DIAGrams TAMODIA’2009 (Brussels, 23-25 September 2009), Tamodia’2009 (Brussels, 23-25 September 2009), LNCS, Vol. 5963, Springer-Verlag, 2010, pp. 146-161.

V. Tran, “Automatic User Interface Generation from Combining Task and Data models”. IAG/LSM Working Paper, June, 2009.

M. Kolp, Y. Wantelet, S. Kiv and V. Tran, “Engineering Software Systems with Social-Driven Templates” in Methodological Advancements in Intelligent Information Technologies: Evolutionary Trends, Advances in Intelligent Inf. Tech. Series, Information Science Publishing, 2010.

Y. Wantelet, Y. Achbany, J.C. Lange and V. Tran, “Developing a Collaborative Supply Chain Management Platform a Service Oriented Approach”, in Intelligent Systems in Operations: Models, Methods, Applications, Information Science Publishing, 2010.

Y. Wantelet, S. Kiv, J.C. Lange and V. Tran, “A Software Package for Actors E-Collaboration in Outbound Logistics using a Multi-Agent Architecture: The TransLogisTIC Project”, in Supply-Chain Management: Theories, Activities/Functions and Problems, Nova Science Publishers, New York, 2010.

V. Tran, “DB-USE: Generating User Interface from Task, Domain and User models”, Proc. of 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems EICS’2010 (Berlin, June 19-23, 2010), ACM Press, New York, 2010, pp. 353-356..

V. Tran, J. Vanderdonckt, M. Kolp, Y. Wantelet, “Generating User Interface for Information Applications from Task, Domain and User models with DB-USE”, Proc. of 1st Int. Workshop on User Interface eXtensible Markup Language UsiXML’2010 (Berlin, 20 June 2010), Thales Research and Technology France, Paris, 2010, pp. 183-194.

Y. Wautelet, S. Kiv, V. Tran M. Kolp, “Strategic Reasoning in Software Development”, 12th International Conference on Enterprise Information Systems (ICEIS’10), Portugal, 2010.

Y. Wautelet, S. Kiv, V. Tran, and M. Kolp “Round Tripping in Component Based Software Development”, 2010 IEEE/WIC/ACM International Conference on Intelligent Agent Technology, Toronto, Canada 2010.

V. Tran, J. Vanderdonckt, M. Kolp, Y. Wautelet, “Using Task and Data models for User Interface Declarative Generation”, 12th International Conference on Enterprise Information Systems (ICEIS’10), Portugal, 2010.

Table of Contents

1	INTRODUCTION.....	1
1.1	MOTIVATIONS IN USER INTERFACE GENERATION.....	1
1.1.1	USER INTERFACE GENERATION FOR DATABASE APPLICATIONS	1
1.1.2	ADVANTAGES OF DECLARATIVE MODELS IN UI GENERATION	4
1.1.3	ADVANTAGES AND PROBLEMS OF THE USER INTERFACE GENERATION.....	5
1.2	THE AGENT PARADIGM.....	7
1.3	AUTOMATIC USER INTERFACE GENERATION FROM COMBINING TASK, USER AND DOMAIN MODELS	8
1.4	READING MAP.....	10
2	STATE OF THE ART	13
2.1	MODEL-BASED USER INTERFACE DEVELOPMENT ENVIRONMENTS – AN OVERVIEW.....	13
2.2	APPROACHES TO AUTOMATIC USER INTERFACE GENERATION	15
2.2.1	TEALLACH SYSTEM: A TOOL FOR GENERATING THE USER INTERFACE.....	15
2.2.1.1	<i>Teallach's Models.....</i>	<i>15</i>
2.2.1.2	<i>Architecture</i>	<i>18</i>
2.2.1.3	<i>Teallach's Tool.....</i>	<i>19</i>
2.2.2	GOLIATH: A MODEL-AND-AGENT BASED ENVIRONMENT TO DESIGN USER INTERFACES.....	20
2.2.2.1	<i>Goliath's models.....</i>	<i>20</i>
2.2.2.2	<i>Goliath and its agents.....</i>	<i>21</i>
2.2.2.3	<i>Goliath's Tool.....</i>	<i>22</i>
2.2.3	WOLD: A MIXED-INITIATIVE WIZARD FOR PRODUCING MULTI-PLATFORM USER INTERFACES	23
2.2.3.1	<i>Wold's Domain Model.....</i>	<i>23</i>
2.2.3.2	<i>Architecture</i>	<i>24</i>
2.2.3.3	<i>Wold Tool</i>	<i>25</i>
2.2.4	ORACLE JDEVELOPER AND MS ACCESS.....	26
2.3	MODEL-BASED USER INTERFACE DEVELOPMENT ENVIRONMENT: A SURVEY.....	27
2.3.1	DECLARATIVE MODELS: GENERIC DEFINITIONS	27
2.3.2	APPLICATIONS AND THEIR DECLARATIVE MODELS	28
2.3.3	DESIGN TOOLS AND GENERATED UIs.....	31
2.3.4	DESIGN TOOLS AND GENERATED APPLICATION CODES	34

Table of Contents

2.4	CONCLUSION	34
2.5	CHAPTER SUMMARY.....	36
3	CONCEPTUAL MODELLING OF GRAPHICAL USER INTERFACES	37
3.1	MODELS	37
3.1.1	TASK MODEL.....	37
3.1.1.1	CTT: <i>ConcurTaskTrees</i>	38
3.1.1.2	DB-USE's Task Model: A Task Model for User Interface Design ...	43
3.1.2	DB-USE's DOMAIN MODEL	50
3.1.3	DB-USE's USER MODEL	52
3.2	USER INTERFACE	55
3.2.1	CONCRETE USER INTERFACE	56
3.2.2	FINAL USER INTERFACE	58
3.3	THE SKWYRL FRAMEWORK.....	59
3.3.1	THE SOCIAL DIMENSION.....	59
3.3.2	THE STRUCTURAL DIMENSION	59
3.3.3	THE DYNAMIC DIMENSION.....	61
3.4	CHAPTER SUMMARY.....	62
4	DB-USE: GENERATING UI FROM TASK, DOMAIN, USER MODELS COMBINED TOGETHER.....	63
4.1	ENGINEERING USER INTERFACES FROM COMBINING TASK, DOMAIN AND USER MODELS	63
4.1.1	WORKING PROCESS IN PRACTICE	63
4.1.2	DB-USE ARCHITECTURE	64
4.1.3	DB-USE WORKFLOW.....	67
4.1.4	DB-USE METHODOLOGY.....	69
4.2	DB-USE PROCESS	72
4.2.1	DB-USE - SOCIAL MODEL	72
4.2.2	DB-USE - AGENT STRUCTURE.....	74
4.2.2.1	Model Editor Agent	74
4.2.2.2	Developer Agent	82
4.2.2.3	UI Builder Agent.....	88
4.2.2.4	Functions Editor Agent.....	98
4.2.2.5	Code Generator Agent.....	101
4.2.3	DB-USE DYNAMIC DIAGRAM.....	105
4.4	CHAPTER SUMMARY.....	107

Table of Contents

5	DB-USE: SOFTWARE AND CASE STUDY	109
5.1	DB-USE TOOL	109
5.1.1	MODEL EDITORS	112
5.1.1.1	Task Model Editor	112
5.1.1.2	Domain Model Editor	120
5.1.2	USER INTERFACE CREATOR	122
5.1.3	CODE GENERATOR	124
5.1.3.1	User Interface Code Generation	126
5.1.3.2	Application Code Generation	131
5.1.3.3	How to generate code in C or Visual Basic	137
5.2	CASE STUDY	138
5.2.1	HUMAN RESOURCE MANAGEMENT	139
5.2.1.1	Edit Employees Task	139
5.2.1.2	Edit job information Task	142
5.2.2	TRANSLOGISTIC	145
5.2.2.1	Create a shipment order Task	145
5.2.2.2	Perform shipment order Task	147
5.2.3	CONCLUSION	149
5.3	VALIDATION	151
5.3.1	JUST-UI: USING PATTERN TO GENERATE USER INTERFACE	151
5.3.2	VAMPIRE: AGENTS AND GUIs FROM TASK MODEL	152
5.3.3	VERIFY THE COVERAGE OF CONCEPTUAL COAD & YOURDON PATTERNS BY DB-USE	153
5.4	CHAPTER SUMMARY	156
6	CONCLUSION	157
6.1	ORIGINAL CONTRIBUTION OF THIS THESIS	157
6.2	BENEFITS AND SHORTCOMINGS	159
6.2.1	POTENTIAL BENEFITS	159
6.2.2	SHORTCOMINGS	160
6.3	FUTURE WORK	162
7	REFERENCES	167

Table of Figures

FIGURE 1-1: DOMAIN-TASK-PRESENTATION MAPPINGS (FROM [PRIB07])	3
FIGURE 1-2: GENERIC FRAMEWORK FOR GENERATING THE UI BY USING DOMAIN MODEL	3
FIGURE 1-3: CAPABILITIES VS. RESOURCES FOR PRODUCING A USER INTERFACE.....	5
FIGURE 2-1: AUTOMATIC UI GENERATION PROCESS BASED ON DECLARATIVE MODELS.....	14
FIGURE 2-2: THE TEALLACH ARCHITECTURE.....	18
FIGURE 2-3: TEALLACH TOOL AND ITS MODELS (FROM [BARC99])	19
FIGURE 2-4: THE FIVE AGENTS OF GOLIATH (FROM [JULI04A])	22
FIGURE 2-5: GOLIATH'S DESIGN TOOL (FROM [JULI04B])	23
FIGURE 2-6: THE APPLICATION CODE GENERATION (FROM [STOC04A])	24
FIGURE 2-7: DOMAIN MODEL CONTAINED IN DATABASE STRUCTURE (FROM [STOC04B]).....	25
FIGURE 3-1: A TASK MODEL IN CONCURTASKTREES	39
FIGURE 3-2: DB-USE TASK MODEL	46
FIGURE 3-3: LOGIN TASK PATTERN	47
FIGURE 3-4: EMPLOYEE MANAGEMENT DOMAIN MODEL ILLUSTRATING THE DOMAIN OBJECTS AND ITS GENERIC FUNCTIONS	51
FIGURE 3-5: GROUPING USERS BASED ON USER CHARACTERISTICS	53
FIGURE 3-6: FUIs FOR VIEWING EMPLOYEES IN TWO DIFFERENT ENVIRONMENTS.....	58
FIGURE 3-7: SOCIAL DIAGRAM – MAIL SERVICES	59
FIGURE 3-8: AGENT – MAILSERVER.....	60
FIGURE 3-9: PLAN - OPENCONNECTION.	60
FIGURE 3-10: DYNAMIC DIAGRAM – EMAIL SERVICES	61
FIGURE 4-1: THE WORKING PROCESS IN PRACTICE	64
FIGURE 4-2: MAIN COMPONENTS OF UI AND CODE GENERATION ARCHITECTURE....	65
FIGURE 4-3: CODE GENERATION WORKFLOW: UIs ARE GENERATED FROM TASK, USER AND DOMAIN MODELS.....	68
FIGURE 4-4: DB-USE - METHODOLOGY	70
FIGURE 4-5: MAPPING RULES, ALGORITHMS, TEMPLATES USED IN DB-USE METHODOLOGY.....	71
FIGURE 4-6: I* SOCIAL DIAGRAM – PROCESSING CODE GENERATION FROM TASK, DOMAIN AND USER MODELS.	73
FIGURE 4-7: THE MODEL ANALYST AGENT STRUCTURE.....	74
FIGURE 4-8: THE LOADTASKMODEL PLAN STRUCTURE.	75
FIGURE 4-9: THE LOADUSERMODEL PLAN STRUCTURE	79
FIGURE 4-10: THE LOADDOMAINMODEL PLAN STRUCTURE	80
FIGURE 4-11: DEVELOPER AGENT STRUCTURE	82
FIGURE 4-12: DETERMINEATASK PLAN STRUCTURE	83
FIGURE 4-13: MAKELINKFOROPERATIONTASK PLAN STRUCTURE.....	85
FIGURE 4-14: MAKELINKFORACTIONTASK PLAN	87
FIGURE 4-15: UI CREATOR AGENT STRUCTURE	89

Table of Figures

FIGURE 4-16: CREATECIO PLAN STRUCTURE	90
FIGURE 4-17: TRANSFORMCIOSTOFIOS PLAN STRUCTURE.....	96
FIGURE 4-18: THE FUNCTION EDITOR AGENT AND DEFINEFUNCTIONS PLAN STRUCTURE.....	98
FIGURE 4-19: FUNCTION STRUCTURES.	99
FIGURE 4-20: FUNCTION REPORT	100
FIGURE 4-21: THE CODE GENERATOR AGENT STRUCTURE	101
FIGURE 4-22: GENERATEUICODE PLAN STRUCTURE	102
FIGURE 4-23: GENERATEAPPCode PLAN STRUCTURE.....	103
FIGURE 5-1: DB-USE TOOL	111
FIGURE 5-2: TASK MODEL EDITOR.....	113
FIGURE 5-3: TASK MENU	114
FIGURE 5-4: TASK FORM	114
FIGURE 5-5: LOGIN DIALOGUE.....	120
FIGURE 5-6: DOMAIN'S OBJECT DIALOGUE.....	121
FIGURE 5-7: DOMAIN'S OBJECT DIALOGUE.....	121
FIGURE 5-8: THE FUNCTION MENU	122
FIGURE 5-9: MAKING THE ASSOCIATIONS.....	122
FIGURE 5-10: USER INTERFACE OBJECTS ARE DISPLAYED IN TERMS OF TEXT	123
FIGURE 5-11: THE PROCESS OF CODE GENERATION	125
FIGURE 5-12: GENERATING THE USER INTERFACE CODE	127
FIGURE 5-13: CODE GENERATION PROCESS FOR SETTING DATA OF A COMBO-BOX..	128
FIGURE 5-14: FOREIGN OBJECT TEMPLATE	128
FIGURE 5-15: DATA RECORD TEMPLATE.....	129
FIGURE 5-16: GETDATAFORCOMBOBOX TEMPLATE	130
FIGURE 5-17: ITEMSTATECHANGED TEMPLATE	131
FIGURE 5-18: APPLICATION CODE GENERATION PROCESS	131
FIGURE 5-19: INPUTED OBJECTS	132
FIGURE 5-20: TEMPLATE FOR STORING THE DATA RECORDS	133
FIGURE 5-21: DELETE FUNCTION TEMPLATE	134
FIGURE 5-22: ADDNew FUNCTION TEMPLATE.....	134
FIGURE 5-23: UPDATE FUNCTION TEMPLATE.....	135
FIGURE 5-24A: GETDATA TEMPLATE.....	136
FIGURE 5-24B: DISPLAYDATA TEMPLATE.....	136
FIGURE 5-25: DISPLAYDATA TEMPLATE	137
FIGURE 5-26: TREAT ORDERS SERVICES	138
FIGURE 5-27: <i>EDIT EMPLOYEES</i> TASK.....	140
FIGURE 5-28: CREATING THE UI FOR <i>EDIT EMPLOYEE INFORMATION</i> TASK.....	141
FIGURE 5-29: EDIT EMPLOYEE INFORMATION FORM	142
FIGURE 5-30: EDIT JOB INFORMATION TASK	143
FIGURE 5-31: CREATING THE UI FOR <i>EDIT JOB INFORMATION</i> TASK	144
FIGURE 5-32: EDIT JOB INFORMATION FORM.....	144
FIGURE 5-33: <i>CREATE A SHIPMENT ORDER</i> TASK.....	145
FIGURE 5-34: GENERATING THE UI FOR <i>CREATE A SHIPMENT ORDER</i> TASK	146
FIGURE 5-35: <i>CREATE A SHIPMENT ORDER</i> FORM.....	147
FIGURE 5-36: <i>PERFORM SHIPMENT ORDER</i> TASK	147
FIGURE 5-37: GENERATING THE UI FOR <i>PERFORM SHIPMENT ORDER</i> TASK.....	148

Table of Figures

FIGURE 5-38: <i>PERFORM SHIPMENT ORDER FORM</i>	148
FIGURE 5-39: <i>SHIPMENT ORDER FORM – GRAPHIC OBJECTS</i>	149
FIGURE 5-40: <i>SHIPMENT ORDER FORM – TEXT OBJECT</i>	149
FIGURE 5-41: <i>SERVICE AND TRANSPORT FORM – GRAPHIC OBJECTS</i>	150
FIGURE 5-42: <i>SERVICE AND TRANSPORT FORM – TEXT OBJECT</i>	150
FIGURE 5-43: <i>VAMPIRE TOOL (FROM [EISE02])</i>	153
FIGURE 5-44: <i>DESIGN PATTERNS FOR TRANSACTION (FROM [COAD95])</i>	154
FIGURE 6-1: <i>LEVEL OF AUTOMATION VS. LEVEL OF STANDARDIZATION</i>	161
FIGURE 6-2: <i>LINKING PERFORM SHIPMENT ORDER FORMS</i>	162
FIGURE 6-3: <i>“ITEM - LINE ITEM” PATTERN</i>	163
FIGURE 6-4: <i>POSSIBLE INTERACTION STYLES DEPENDING ON USER PREFERENCES...</i>	164
FIGURE 6-5: <i>A UI COMBINING DIFFERENT INTERACTION STYLES: FORM FILLING, COMMAND LANGUAGE, AND FUNCTION KEYS.</i>	164
FIGURE 6-6: <i>FIRST ORDER RELATIONSHIP - EXAMPLE</i>	165
FIGURE 6-7: <i>“FROM DOMAIN TO TASK MODELS” TOOL</i>	166

Table of Tables

TABLE 2-1: MODEL'S NOTION	28
TABLE 2-2: THE DIFFERENT APPROACHES IN MB-UIDE AND THE USED DECLARATIVE MODELS	29
TABLE 2-3: THE DIFFERENT APPROACHES IN MB-UIDE AND THEIR TOOLS	32
TABLE 2-4: APPLICATION CODE GENERATION OF THE APPROACHES	34
TABLE 3-1: TYPES OF TASK.	41
TABLE 3-2: TEMPORAL OPERATORS USED IN CONCURTASKTREES	42
TABLE 3-3: RELATIONSHIPS TABLE.	43
TABLE 3-4: TASK ATTRIBUTES	44
TABLE 3-5: ACTION TYPES	44
TABLE 3-6: TASK TYPES	45
TABLE 3-7: GENERIC TASK PATTERNS FOR ELECTRONIC COMMERCE, EMANAGEMENT APPLICATIONS	48
TABLE 3-8: SPECIFIC TASK PATTERNS FOR ELECTRONIC APPLICATION	49
TABLE 3-9: DEFINED FUNCTIONS.	52
TABLE 3-10: CIO TYPE TABLE.	57
TABLE 4-1: TASK MAPPING.	77
TABLE 4-2: DATATYPE MAPPING TABLE.	81
TABLE 4-3: MAPPING THE DATA TYPE OF THE DOMAIN OBJECT TO THE CONTROL TYPE OF THE CIO	93
TABLE 4-4: CLASSIFY UI CONTROLS BASED ON ABILITY OF USER IN COMPUTER APPLICATIONS	94
TABLE 4-5: MAPPING THE CONCRETE CONTROL TYPE BASED ON THE LANGUAGE	97
TABLE 4-6: MESSAGE TABLE	104
TABLE 5-1: AGENTS OF DB-USE IN JAVA	110
TABLE 5-2: DATA TYPE MAPPING	128
TABLE 5-3: PATTERN'S FUNCTIONS AND PERFORMING THEM IN DB-USE	155
TABLE 5-4: FUNCTIONS OF DB-USE AND RELATED PATTERN'S FUNCTIONS	156

Table of Examples

EXAMPLE 4-1: AN EXAMPLE OF TASK <i>REVIEW EMPLOYEE</i> CREATED BY CTT.	76
EXAMPLE 4-2: MAPPING OF ACCESSSTUDENTDATA TASK	78
EXAMPLE 4-3: DOMAIN MODEL	82
EXAMPLE 4-4: DISPLAY A CHOSEN TASK FOR GENERATING THE UI.....	84
EXAMPLE 4-5: LINKING TASKS AND ATTRIBUTES OF DOMAIN'S OBJECTS	86
EXAMPLE 4-6: LINKING TASKS TO THE FUNCTIONS DEFINED IN DOMAIN MODEL.....	88
EXAMPLE 4-7: CREATING CUIs PROCESS.....	91
EXAMPLE 4-8: CIOs ARE DESCRIBED BY USING UsiXML	95
EXAMPLE 4-9: CIOs TO FIOs TRANSFORMATION PROCESS BASED ON EXAMPLE 4-7.98	
EXAMPLE 4-10: THE VALUE CHANGE OF THE DATA FIELDS	105
EXAMPLE 5-1: TASK <i>CHOOSE JOB TITLE</i> IS MODIFIED	115
EXAMPLE 5-2: SETTING TASK POSITION	117
EXAMPLE 5-3: ADDING A NEW TASK INTO THE TASK TREE	118
EXAMPLE 5-4: DELETING A TASK FROM THE TASK TREE	119
EXAMPLE 5-5: USER INTERFACE OBJECTS ARE DISPLAYED IN TERMS OF GRAPHICS .	124
EXAMPLE 5-6: JOBS – FOREIGN OBJECT	129
EXAMPLE 5-7: JOBS – DATA RECORD	130
EXAMPLE 5-8: JOBS – GETDATAFORCOMBOBOX	130
EXAMPLE 5-9: JOBS – ITEMSTATECHANGED	131
EXAMPLE 5-10: DOMAIN OBJECT- EXAMPLE	132
EXAMPLE 5-11: MESSAGE DIALOG BOXES (ERROR, CONFIRMATION, WARNING, AND INFORMATION)	137
EXAMPLE 5-12: TEMPLATES FOR GENERATING A TEXT-FIELD IN C++ AND VB	138

*Chapter 1***Introduction**

This chapter introduces and motivates the thesis. Section 1.1 overviews our research issue in the context of user interface generation and describes some advantages of using declarative models to specify the user interface. Section 1.2 presents some important points based on which we adopt the agent paradigm to represent our user interface generation process. Section 1.3 describes the context in which our framework - a methodology, model and computer software - is defined to generate the user interface but also the code behind. Finally, Section 1.4 concludes this introduction proposing a reading map of this manuscript.

1.1 Motivations in User Interface Generation

This section introduces the motivations of this research, especially the generation of user interfaces for database applications and the advantages and problems in using declarative models for user interface generation.

1.1.1 User Interface Generation for Database Applications

Database systems have always been a major component in business-oriented software applications such as Enterprise Resource Planning, Material Requirements Planning, EDI or data warehouse software [Lau09]. To be productive on a day-to-day basis and used by the organization's non-IT staff, these complex enterprise packages require all to support and provide efficient human-computer interaction (HCI) with the database management systems [Kroe07].

From the point of view of final business actors, these User Interfaces (UIs) are today managed through usable and powerful database tools that provide some support for using and manipulating data in queries that involve several components and heterogeneous component types of database such as tables, table spaces, indexes, views, queries, keys. By User Interface, we hereby refer to as the portion of an interactive application that handles the interaction with the end-user. An interactive application is usually decomposed into three parts: the UI, the business logic (also referred to as the functional core) and the control that ensures the connexion between the UI and the business logic.

As an example of such a complex UI, just consider a simple order processing application that an ERP typically has to offer: this UI requires to access and present data from different types of structures which reside in the sales, accounting, finance,

manufacturing and productions database modules [Lau09]. Beside of the advantages which come from the database, the CRUDS pattern (Create, Read, Update, Delete, and Search) also largely contributes in automatic UI generation.

Naturally database interfacing has been a technical and human interaction issue since the seventies [Grif01, Eise00]. But, with today's increasing use of very large databases and extended business applications such as ERP II, Customer Relationship Management, E-procurement, Reporting/Drill-Down or Data Mining systems, these UIs require dynamic automation and run-time generation to be properly dealt with on a large-scale.

UI researchers have richly discussed the capability and importance of automatic UI generation and propose them as the core of visual-based development environments [Nich05]. There are currently numerous and various approaches using different input materials: designs, patterns, architectures, declarative models... In this set of techniques, an emerging method is the automatic UI generation from declarative models [Puer94a, Silv00, Schl96, Jans93, Grif01, Juli04], inspired from Fourth Generation Languages code generation [Silv00]. In practice, these models are high-level abstractions such as goal or task [Boda95a], presentation, dialogue [Jans93] or interaction, domain [Puer94a, Puer96] models on which we will focus in this research. Examples of such model instantiations are *i** [Wood95], CTT (Concur Task Tree) [Pate01a] or UML. The high-level abstraction features provided by these declarative models typically reduce the semantic gap between the software and organizational concepts. As a matter of fact, the agent model, for instance, uses modeling constructs that are inspired from the organizational world (actors, goals, beliefs, plans, intentions, resources, ...) and typically serve well by essence as a human-computer interaction medium in software development including database and UI applications design. The task model, declarative model typically used in UI design, records the tasks that potential end-users of the system may need to perform in their jobs, independently of dealing with a particular computer [Hart92, Pate01a]. Much of the design of an interactive system is generated on the base of supporting these tasks.

Another technique related to declarative models, is to use the domain model of or engineered from the database itself. For instance, (object-)relational domain models can be effectively used to generate the database application interfaces [Puer94a, Jans93, Grif01, Baar92]. Unfortunately this method only generates static and inflexible layouts of the UI and fails to consider the dynamic features that operate on the database.

Moreover, automatically generated database applications UIs poorly manipulate data as a combination of simple widgets such as lists and forms. Typical examples are forms for search criteria and data detail with ok/cancel buttons to validate or cancel operations and transactions, list for table results with display and manipulation buttons for adding/editing/deleting/searching data [Moro06].

Typically, a strong motivation for database application UI generation is the unexploited potential of relationships between objects. This makes it possible to

create task-domain mapping rules that are in turn mapped onto containers in the UI model [Prib07].

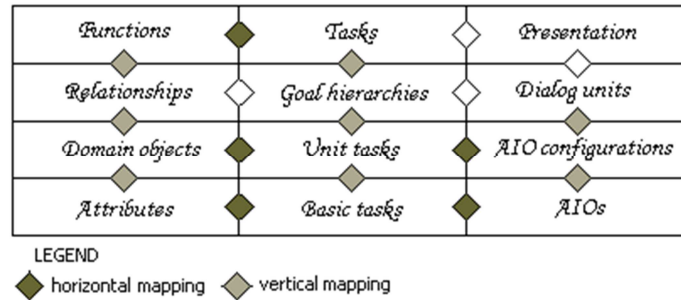


Figure 1-1: Domain-task-presentation mappings (From [Prib07])

Most of the approaches of the automatic UI generation of the database application have used the domain model. A domain model represents a high-level knowledge of the definitions and relationships of a given application domain; this establishes a vocabulary to access the objects in an application domain. Typically, this model should be used in combination with task, dialog, or presentation models, but this is not always the case.

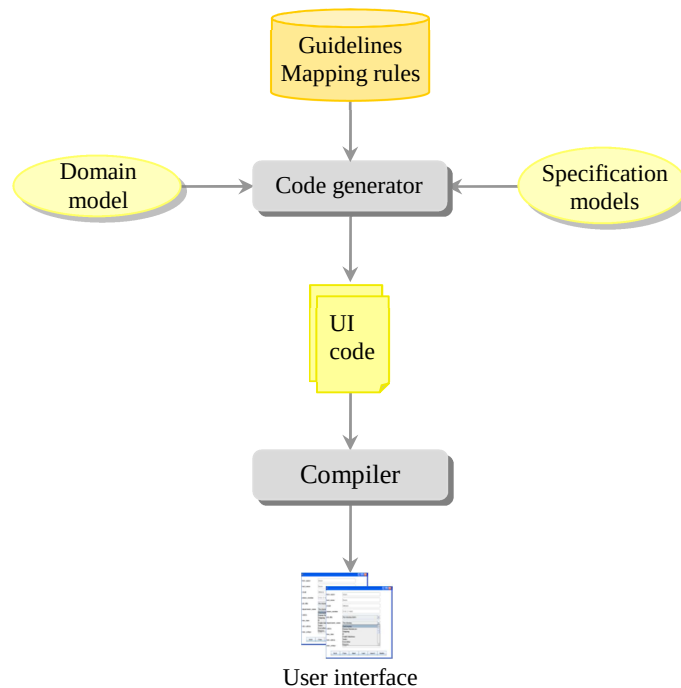


Figure 1-2: Generic framework for generating the UI by using domain model

Figure 1-2 depicts the generic framework for automated UI generation based on domain models. The code generator uses the guideline bases and mapping rules to generate the UI code based on the domain model and the specification models. The different approaches use the different specification models. For example in Genius [Jans93], it is a dialogue model; in Teallach [Silv00], it is a task model; in Goliath [Juli04a] it is a presentation model; and in Mecano [Puer94a], it is an interface model.

1.1.2 Advantages of Declarative Models in UI Generation

UI is that subset of a software system that interacts with the user. Since it is used to communicate between the end-users and the system's computer, making it a crucial element, the successfulness of a software system depends on, besides the design of its architecture, the UI design [Myer00]. End-users expect UIs to facilitate the use, understanding and give the most adequate result.

The development of a successful interactive system requires careful design of the UI. The UI affects the acceptance of the application by the users. A good UI design helps the users to work effectively and quickly. The first point is to take a step-wise approach by using a methodological UI derivation process based on the available resources such as task, domain, data, user, presentation, dialog models. This process is conducted by designing the UI manually or semi-automatically [Eise02] or automatically [Boda95a]. In this thesis, we will approach the semi-automatic UI generation.

Numerous research works in the area of automatic UI generation have focused mainly on UI generated from the declarative models. They offer the following advantages [Pinh00] in regard to both the automatic UI and code application generations:

- *Higher-level of abstraction*: They can provide a more abstract description of the UI than the ones provided by other UI development tools.
- *Systematic procedure*: They facilitate designing and implementing the UI in a systematic way since they offer capabilities such as modeling UIs using different levels of abstraction, incrementally refining the models, and re-using UI specifications.
- *Software support*: They provide the infrastructure required to automate tasks related to the UI design and implementation processes.

An important point in using declarative models for creating the UI is that these models are typically described at a conceptual level. So that, the UI created based on these models can be implemented in different ways on the different delivery platforms.

1.1.3 Advantages and Problems of the User Interface Generation

Potential benefits of automatic UI generation for designers and developers are:

- *Low-cost development*: it reduces work for designer in designing and for developer in implementing the UI.
- *Rapid implementation*: the automatic generation helps the designer/developer to implement the system quicker than by hand.
- *Effective development*: The code generated is consistent by construction. Fewer flaws and defects subsist in the application in principle (*error-free quality*).

The model-based approach has been the target of some major criticisms [Myer00, Puer96, Szek96]. The main shortcomings commonly cited are:

- (1) *High threshold*: the designers need to learn a new language in order to express the specifications of the UI.
- (2) *Low ceiling*: each model-based system has strict limitations on the kind of UIs they can produce and the generated UIs are generally not as good as those that could be created with conventional techniques.
- (3) *Wide walls*: model-based systems do not support a wide range of possible explorations [Shne06].

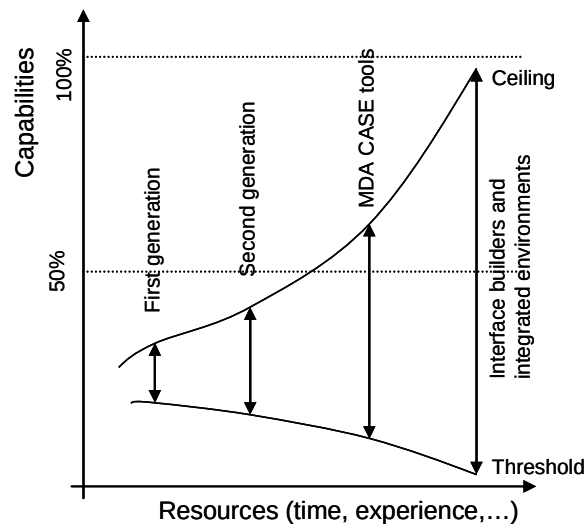


Figure 1-3: Capabilities vs. resources for producing a user interface

Typically, UIs produced in interface builders and IDEs require some significant amount of resources (in terms of time, experience, skills), probably more than model-based IDEs, but their coverage is maximum (Figure 1-3): they exhibit a low threshold and a high ceiling. In contrast, first-generation model-based IDEs suffered

from a high threshold and a low ceiling: they forced designers and developers to learn a new language (the one of the models), but once this effort is made, the resources required to produce the UI are low. However, only some limited UIs could be obtained. The second generation of model-based IDEs has expanded this coverage and the trend is now pursued by MDA-compliant softwares. It is worth to notice that such softwares are assumed to require less effort for learning the models since these models are already part of general purpose development methods like UML. We therefore hope that the coverage of such tools will progressively reach the coverage of traditional tools, but always with less resources involved.

Automatic UI generation using declarative models is not a new research domain. Such models like the task, user, presentation or domain models can be used to describe the aspects or viewpoints of the UIs of an application. For instance, existing works such as Tood [Mahf05] use the task and user models to generate the UI. The UI is automatically generated from the domain and use case models [Rosa07], Mecano [Puer94a, Puer96] and from combining task, domain and presentation models [Brid05]. Trident [Boda95a] and Fuse [Lonc96] combine the task and domain models.

Viewing from over the current generated UI systems, we can state that automatic generation is very large and complex; the artifacts used in UI generation come from the different resources such as the data, domain, task and user models [Boda95a], data schema [Baja06]. Therefore, beside the above mentioned advantages, the following issues can also be advanced:

- *Lack of mapping rules*: how to relate the different models with each other, in this thesis, they are task model and domain model; and how to translate these models in order to support the design process [Prib07].
- *Lack of flexibility*: the UI generation systems do not allow the developer to change the UI before it is generated. The system should show the developer the UI which will be automatically generated; thus the developer could change it according to his/her expectations.
- *Lack of attention to the user*: the automatic generated UI is not user-centered. The user's preference is not considered in the automatic generation. The user's preference is reflected in the user's characteristics. These characteristics are the skill, experience, professional knowledge of the users.
- *Lack of adaptability*: almost no UIs generated automatically can be accepted immediately so they have to be modified by the developer in order to meet the requirements of the application.
- *Lack of techniques for combining models*: so far, the results produced from the combination of the task and domain models in UI generation are not considered rich enough to meet all of the requirements determining the UI [Schl96]. Further research works should be done to adequately exploit the advantages of combining these task and domain models. The same statement can be made for combining other models.

- *Lack of resource management*: the current UI generation systems do not support the developers in the process management and traceability of their resources. When the content of one of the models changes then the UIs generated from this model should change to match the new version of the model; likewise, when the UI changes then the models used to generate this UI should also integrate these changes.
- *Lack of techniques for generating application code*: most of the research works have focused on the UI generation only; they have forgotten the importance of application code generation. To make an accurate application at a low cost the developers expect that both the UI and the application code are automatically generated as generic functions of a database application that are not belonging to the domain model.

1.2 The Agent Paradigm

Our framework will be based on and supported by declarative technologies we have pointed out above. More specifically, we will adopt the agent paradigm (models, language, methods, ...), that we introduce below to analyze task, domain and user models and generate the database UI specifications and application code.

An agent is a declarative entity defined as “a computer system, situated in some environment that is capable of flexible autonomic action in order to meet its design objective” [Wood95, Wood96, Zamb00]. The declarative nature of an agent can be characterized by its autonomy, flexibility and situateness [Wood96]. The autonomy of agents reflects their social and decentralized nature. The flexible way in which agents operate to accomplish their goals is particularly suited to these business systems which are normally expected to function in an ever-changing environment. The situateness implicitly supposes that agents are not useful as stand-alone entities. As a matter of fact, they are most of the time structured in a multi-agent system (MAS).

The global behavior of a MAS derives from the interaction between the constituent agents: they cooperate, coordinate or negotiate with one another. A multi-agent system is then considered as a society of autonomous, collaborative, and goal-driven software components, the agents, like a social organization. The agents intelligently and automatically behave and collaborate to fulfill the objectives of a declarative structure [Zamb00]. Each role that an agent has the ability to play has a well-defined set of responsibilities (goals) achieved by means of an agent’s own abilities, as well as its interaction capabilities.

This sociality of MAS is well adapted for describing the complexity of the organization of software systems. It permits a better match between system architectures and its organizational operational environment, it reflects the social and decentralized nature of modern systems [BMO01] that are operated by different stakeholders [Parunuak97], it is suited to the dynamic and unpredictable situations in which a software is now expected to run [ZJOW00, ZJW00].

Automatic User Interface Generation from Combining Task, User and Domain models

In addition to properties discussed above, multi-agent systems also naturally fit services oriented architectures. Each model can indeed be managed through one agent that can provide the model itself and the SOA with its capabilities. Since, numerous models can be possible; factorization made possible by multi-agent architecture is as important and useful as modularity.

1.3 Automatic User Interface Generation from Combining Task, User and Domain models

The information used to build the UI usually comes from a large and complex context in which the users work to complete their tasks. This context includes the users' characteristics (user model), their current domain of application (domain model often linked in the database world to the data or conceptual model), the tasks they commonly perform (task model), the platform they work on (platform model), the device they are currently using (device model) and so on. This context is not fixed for all the UI generating processes since it depends on the different approaches. For example, in Trident [Boda95a] the UI is determined from the task and application models, in Mecano [Puer94a, Puer96] the UI is determined from the sole domain model.

The task, domain and user models are three important models based on which the UI is easily built for the following reasons:

- *The task model describes the abstract user interface.* The task model expresses how an end-user may want to interact with a system in order to reach a given goal. This expression is intended to be independent of any particular implementation or technology. This explains why a same set of models could initiate several different UIs [Wils96].
- *The domain model provides the special features needed for creating a UI and specifying the methods for performing the generic application functions.* These features are the attributes of the objects in domain model, the relationships between these objects and prototype of the generic application functions. The domain model is not used separately from other models to generate the UI, it is combined to the task, presentation, dialogue models.
- *The user model supports the creation of user interfaces which consider the preferences of the users.* Like the domain model, the user model is not used separately from other models to generate the UI since there are various aspects of the UIs that are adapted according to user models.

Such models are related to one another. For instance, the relationship between the task and domain models can be described like the connections between the domain objects and the tasks themselves that enable the user to perform operations on the domain objects. These operations are creating, deleting, modifying or selecting the objects in domain model. The UI is specified by selecting the elements of a domain model for the relevant tasks. The tasks can also be used to filter the

domain model elements that are further needed for the UI design [Brib05]. The task model fragments, this time at operational level, can be produced through a mapping between the tasks and the domain objects and their attributes [Brid07]. Such tasks are: “Add new object”, “Edit object attributes”, “Delete object”, “Display object attributes” or “Search object”.

Considering these three models, three main approaches in automatic UI generation can be identified: the first one based on the task models, the second one based on the domain model such as Janus [Balz95] and Mecano [Puer94a, Puer96], and the last one based on the task and domain models such as Trident [Boda95a], Fuse [Lonc96], Teallach [Grif01]. The task model approach performs better since the UI generated can depict the restructured and summarized information obtained from multiple objects. The domain model approach can only produce simple interfaces; typically each UI is represented by one object domain. The last approach, the generation from task and domain models, provides with better result but currently is not well defined nor investigated in the literature.

Hence, combining models is an important concept in UI generation since the different models describe different aspects of the UI. For example, a UI generated from a task model is expected to be a means by which the user can communicate with the system to accomplish his/her task. The UI generated from a user model is expected to support the users based on their characteristics. The UI generated from several different models carry many needed aspects of a UI. Various research works have focused on such models to generate UIs. For instance, Tood [Mahf05] uses the task and user models to generate the UI. Mecano [Puer94a, Puer96] generates UI based on the domain model. Trident [8] or Fuse [Lonc96] combines the task and domain models while Supple [Gajo04] combines the user and device models.

Unfortunately, the researchers who have focused on the UI generation have not investigated and studied the implementation and code generation of application tasks, especially, those generic ones related to editing, inserting, deleting and searching data. This research focuses on the code application generation for the tasks of a data-oriented application coming from the following main points:

- The data manipulation demand is very high. It is repeated regularly in most database applications especially for common tasks such as editing, inserting, deleting and searching data.
- These functions are easily performed through receiving data from the user, displaying data to the user, executing the different SQL select, insert, update and delete queries for the different user requirements.
- The main difference in our work from other ones is to contribute to the mapping rules for creating the UI objects and to combine of the three task, domain and user models to generate the code for performing both the UI and the generic functions of a database application such as Display(), AddNew(), Update(), Delete(), Search(), Review().

The different models serve a specific purpose at different stages of our design process. The task model expresses the knowledge required or procedures used to

Reading map

perform some task; the user model describes the user abilities and beliefs; the domain model defines the aspects of the application which can be adapted or which are otherwise required for running the system. Therefore, the task model is used to specify a generic UI; the domain model is used to specify the control of this UI (See Table 4-3) – at this level the UI is specified in more details; the user model is used to influence the design and to select among alternative solutions in the design space.

This research is aimed at defining and using a methodological framework for developing a user interface to database application from the task model, domain model and user model combined together. This framework consists of:

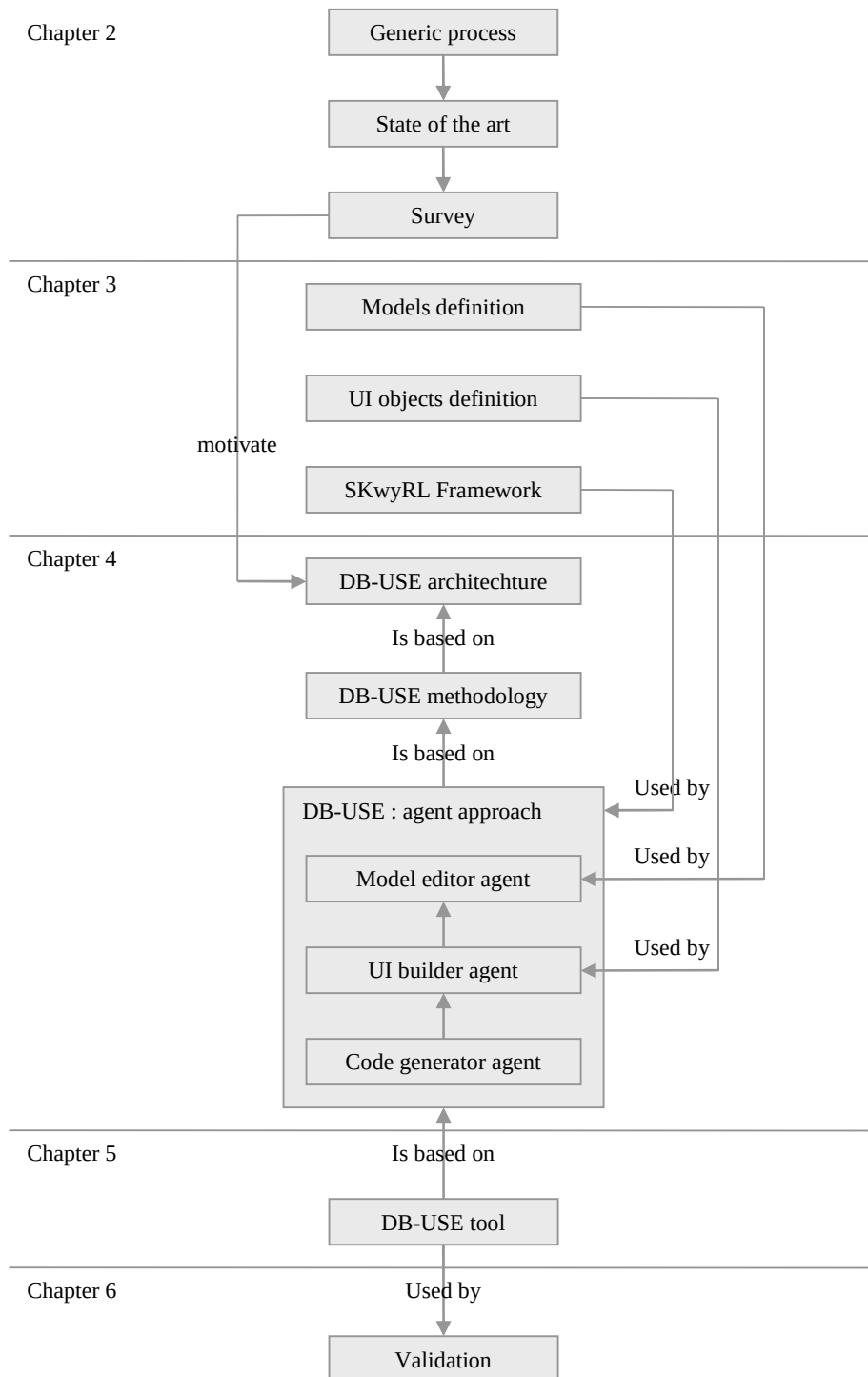
- A meta-model that governs the semantics of the task, domain and user models;
- A methodological process that provide the UI designer/developer with guidance in order to create the UIs systematically from task, domain and user models;
- A software supporting the methodological process in order to generate both the user interface code and the application code that performs the basic functions of the database application.

Our framework is based on and supported by declarative technologies we have pointed out above. More specifically, we will adopt the agent paradigm (models, language, methods, ...) to analyze task, user and domain models and generate the database UI specifications and application code.

The purpose of this framework is to offer a low cost, short time-to-implementation and efficient development environment from the business user side. Indeed, the objective is not to provide a tool for professional database management development staff but to support non-IT end-user with the generation of database applications they do not need to program anymore.

1.4 Reading map

The rest of this thesis is organized as follows: first, in Chapter 2, we overview the related work in the field of automatic UI generation. The task, domain, user models are presented in Chapter 3; the UI objects and the SKwyRL framework are also discussed in this Chapter. Then, we present in Chapter 4 the DB-USE architecture, DB-USE methodology and DB-USE process by using the SKwyRL framework. We explain the roles of the main agents that participate in DB-USE process such as the Model editor, the UI builder, the code generator for both the UI and application. In Chapter 5, we present the tool that we developed for the purpose of this thesis. The case study is discussed in Chapter 6; it describes how the process works through some concrete examples. Finally, we propose some conclusions and proposals for future plans complete the thesis. The reading map is depicted as follows



Reading map

Chapter 2

State of the Art

The purpose of this chapter is to review the advantages and disadvantages in the user interface generation and describe current approaches for generating the user interface from the declarative models. Section 2.1 provides an overview of the user interface development environments. Section 2.2 describes the different approaches to model-based user interface development environment. The approaches in the same domain are summarized in Section 2.3. Section 2.4 lists the advantages that DB-USE wants to learn from current approaches and the new technique that DB-USE will do to improve the UI generation process.

2.1 Model-based User Interface Development Environments – An Overview

There have been many works in the field of automatic user interface generation such as the user interface generation from scenarios [8] (a scenario is a description of a set of users, a work context, and a set of tasks that end-users perform or want to perform); the UI generation based on patterns [Budi96, Rade07] (a pattern is used to describe a UI component including problem, context, solution and force). This thesis focuses on user interface generation from declarative models. As already pointed out, the declarative human-based characteristics and properties offered by agent models typically allow to fill the semantic discrepancy between the software/database side and the organizational/human UI side [Ange94]. Specially, declarative user interface models facilitate the creation of methods to design and implement the UI in a systematic way and also provide the infrastructure required to automate the task related to the UI design and implement process. We will present in the following a state-of-the-art UI generation from declarative models, and we will overview in further details task, domain and user models.

Model-based user interface development (MB-UIDE) is a research area where declarative models are receiving increased interest. This emerging technology is centered around the premise that a declarative interface model can be used as a basis for building interface development environments. The model-based approach facilitates the automation of the design and implementation of user interfaces [Pinh00]. The process of developing a user interface with a MB-UIDE is an iterative process of developing and refining a set of declarative models using graphical editing tools or high-level specification languages. Once these models are developed, they are transformed according to ergonomic rules and style guidelines into an interface specification. This specification is subsequently linked with the

underlying application code to generate a running application. This process is depicted in Figure 2-1.

Figure 2-1 illustrates the main components in the Model-Based Interface Development Environment (MB-IDE). In this process, the user interface developers have to specify the declarative models as task, user, presentation, domain models by using the modeling tools. User interface code is automatically generated based on these declarative models through automatic generation tools. These tools require additional design knowledge including notably guidelines and transformation rules. Finally, the application code created by the application developer and the executable UI code are built by compiler to deliver the desired application.

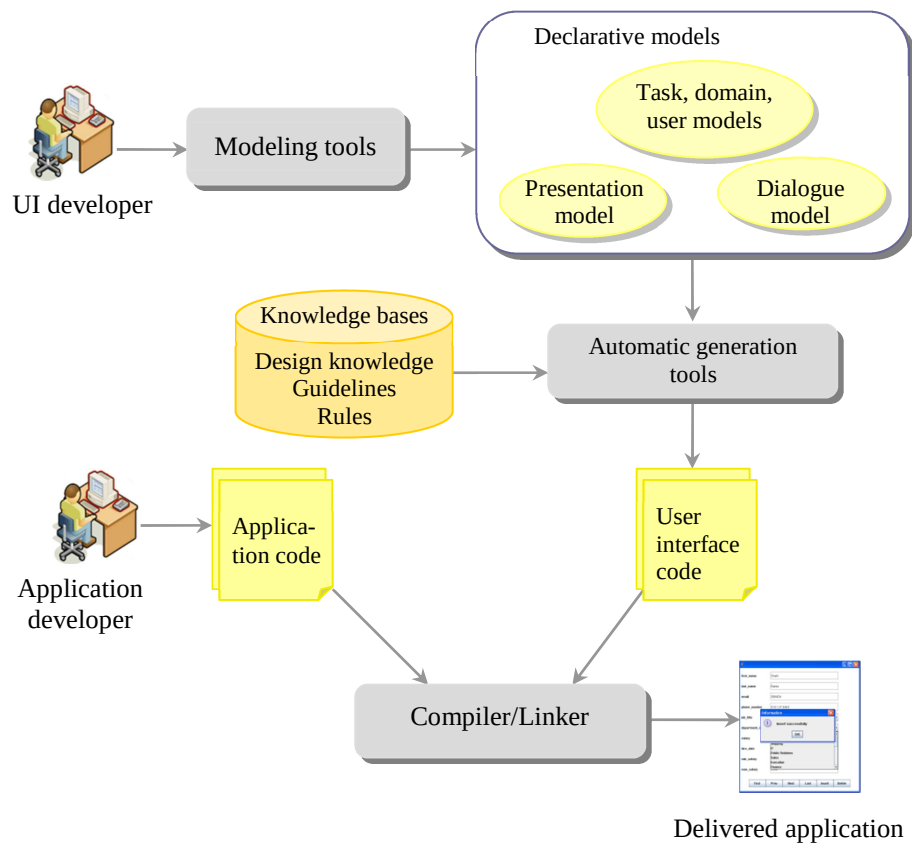


Figure 2-1: Automatic UI Generation Process Based On Declarative Models

From the developer's viewpoint, the main components of a MB-UIDE are the declarative models, which store the conceptual representation of the interface. During recent years, these models were developed and increased both in number and quality specially, the developer focuses on the domain, task, user, dialog and presentation models. In model-based user interface development, these models are

applied to describe different aspects of the user interface or to provide a varying level of detail. The relations between the models representing different levels of abstractions are established through transformations, a concept which is also applied in software engineering with the Model Driven Architecture (MDA).

The Task and domain models are widely used in the automatic user interface generation such as Trident [12], Mecano [Ange94], Teallach [Grif01], Goliath [Juli04a]. During the last few years, research on automatic user interface generation has dramatically increased, specially dedicated around on declarative models. Approaches and frameworks like Trident [12], Genius [Jans93], Janus [Balz95], Mecano [Ange94], typically based on such high abstractions models will be detailed in the following. Declarative models aim at describing UIs in order to provide MB-IDE in which developers can design and implement them in an efficient and systematic way, more easily than when using traditional UI development tools.

2.2 Approaches to Automatic User Interface Generation

To highlight the importance and need of declarative models in automatic UI generation, we present some major processes of user interface generation based on such models. These processes differ from each other in terms of input information, mostly represented by declarative models and from which the generation is effectively done, the generation target, and the generation process itself. In this section, besides discussing the different techniques of the different approaches, we also discuss the different models used in these approaches because the models are differently created then the user interfaces generated based on these models are different.

2.2.1 Teallach System: A Tool for Generating the User Interface

Teallach [Grif01, Barc99, Silv00, Barc00, Grif98] developed a model-based user interface development environment for use with object databases. It seeks to perfect two issues that researchers believe to be important to the wider acceptance and uptake of MB-UIEs. These two issues are the role of such systems within existing software environments and the generation of executable programs from declarative models.

2.2.1.1 Teallach's Models

Teallach uses the task, domain and presentation models to generate the user interfaces. Each model in Teallach defines a view of the information required to generate the interface to a particular application. The domain model describes the underlying application in terms of its data and operation; the task model describes what the user can do with the user interface in terms of its dynamic and information processing requirements; and the presentation model indicates how the resulting interface will appear. These models are in intrinsic relationships with each other.

a Domain Model

Unlike in other MB-UIEs, the domain model of Teallach is automatically created from the information contained in an object database schema. Additionally, the domain model's main ability is to automatically analyze the API (Application Programming Interface) of non-database class libraries and to transmit this information to Teallach's other models. This model provides a degree of sophistication that is not found in other model-based systems. In Teallach, the domain model represents both the objects in the database, and the functionality of the underlying application. These functionalities are used by the interface and provide support for interaction with the database. There are four different kinds of information represented in Teallach's domain model. They are persistent data type, generic database functionality, auxiliary definitions and class libraries.

- Persistent data type is a kind of information which is created from the information contained in the database schema.
- Generic database functionality is the management of the queries, the transactions and connections.
- Auxiliary definitions that have been implemented for use with the application. For instance, the classes not defined in the database can be defined as this type.
- Class libraries are used as the bridge between the Java API and the data type of application. For instance, the integer class is used to validate numeric data.

b Task Model

The task model of Teallach is a goal-oriented task hierarchy in which the leaf tasks represent the action tasks or interaction tasks; the non-leaf tasks are named composite tasks. The relationships between the composite tasks are named sequential, order-independent, optional, repeatable, concurrent, choice, conditional.

- An action task is one that corresponds to some low-level activity carried out by the application and user. An action task may be automatic or manual. An automatic action task is performed behind the scene along with the interface; a manual action task is performed by the user's interaction with the system.
- An interaction task is a task which involves some form of interaction with the user.

The Teallach's task model allows the passing of information into or out of tasks and the declaration of state associated with tasks. This is an important improvement of Teallach's task model. Generally, the other task models do not support this step.

c Presentation Model

Teallach's presentation model describes the presentation of the designed user interface. There are two levels of presentation, the abstract presentation model and concrete presentation model.

The Abstract presentation model describes the user interface in terms of a number of abstract categories which are used for the actions of the users. There are seven categories; they are named FreeContainer, Container (for grouping the interface elements), Inputter (for entering data), Display (for showing data), Editor (for modifying data), Chooser (for choosing among existing data), and ActionItem (for invoking behavior).

The Concrete presentation model contains the user interface components which are used to construct the user interface. These components are designated concrete interaction objects. Typically, they are the buttons, menus, dialogs and so on. Teallach uses the Java Swing widget set because it provides a rich set of widgets, has an efficient underlying implementation, and has a pluggable look-and-feel, facilitating the construction of different styles of interface. Teallach also allows the use of other, custom widgets when building interfaces; this is essential if we are not to rule out domain-specific widgets that particular users may require. The concrete presentation model is represented in two ways: a GUI view, which may be considered a "preview" of part of the user-interface being constructed; and a tree view, which represents the interface as a hierarchical structure, based on the containment hierarchy of its components.

d Interactions between Models

There are two ways to associate the elements between the different models; they are named *linking* and *deriving*. In *linking*, an association is made between existing elements in two models. In *deriving*, elements in one model are constructed based on components in another.

There are four associations between the three Teallach models that are the associations created: between domain and task models, presentation and task models, task and presentation models and, finally, presentation and domain models.

Linking an action task node of task model to an operation of domain model is to specify that the linked operation is performed by this linked task.

Linking a node of the presentation model to a node of the task model specifies that an *instance* of the presentation object is to be used to perform the task in the generated interface. In other words, the interface generated based on the linked presentation node is the representation of the interface of the linked task.

Linking a task node to a node of the presentation model is to derive generation of an abstract interaction object for the linked task node. The abstract interaction object is determined based on the following rules:

Approaches to Automatic User Interface Generation

- If the task node is a non-leaf task then a FreeContainer abstract interaction object is generated.
- If the task node is an interaction task then an Editor abstract interaction object is generated.
- If the task node is an action task then an ActionItem abstract interaction object is generated.

Linking a node of the domain model to a node of the presentation model has two purposes:

- Determining the domain's operations invoked by the presentation's nodes.
- Generating the presentation node if the linked node of the presentation model is a non-leaf node.

2.2.1.2 Architecture

Figure 2-2 presents an overview of Teallach's architecture. The central component of this architecture is the Teallach Model Store which provides some easy means for creating and maintaining the associations between the models. Two main aims of the Model Store are to maintain the links created by the developer between related model components; to provide the mapping rules which are used when automatically generating model fragment in one model from the information captured in another.

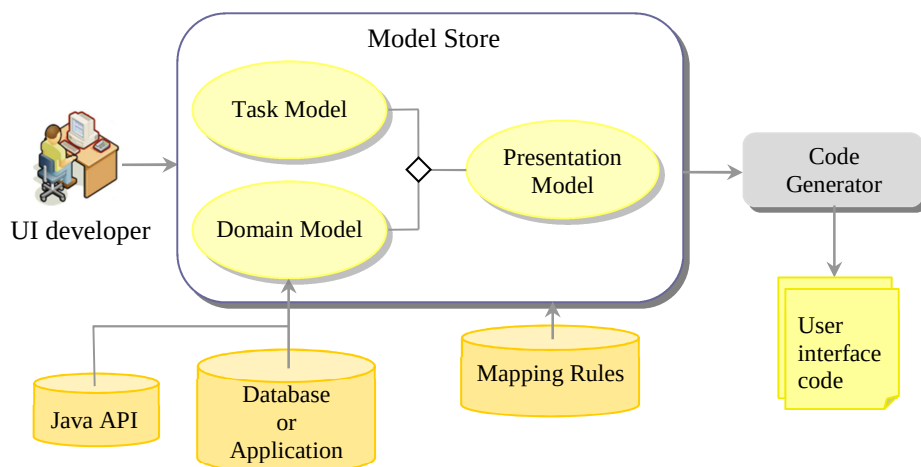


Figure 2-2: The Teallach Architecture

The domain model is created based on a concrete database and the Java's libraries. The task model and the presentation model are created by the UI developer. The system generates the mapping rules automatically.

Once models have been constructed and their components are linked by the UI developer, the user interface code is automatically generated by the code generator. During the process of code generation the links between the components of the Teallach's models are verified. And the calls to underlying domain data and functionality are resolved.

2.2.1.3 Teallach's Tool

Teallach has been developed to facilitate the design of the user interfaces for pre-existing object oriented database. The Teallach tool [Barc99] was created to help the UI developer construct and link the Teallach models. The interactions between the models are described by steps (D), (E) and (F) as follows.

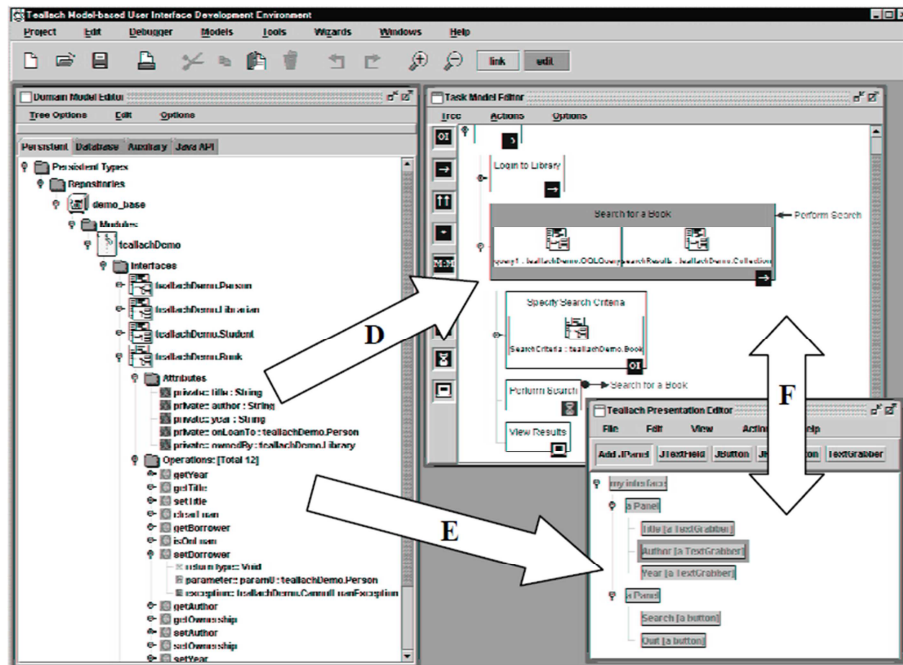


Figure 2-3: Teallach tool and its models (From [Barc99])

The Teallach tool has been implemented using Java 2.0 and the swing GUI tool-set. The user interface code generated is also implemented in Java on all major hardware and OS platforms. The Teallach tool provides three separate editors for three Teallach models. These models exchange the information with each other through drag-and-drop or cut-and-paste manipulations.

The domain model editor is used for three purposes:

- Automatic generation of the components of the application based on the schema of the underlying database.

Approaches to Automatic User Interface Generation

- Creation of the data types and the data structures which are not database classes, but which provide the functionality required for the runtime operation of the application.
- Creation of the standard queries that can be run over the underlying database.

The presentation model editor is used to create the abstract user interface and the concrete user interface.

The task model editor is used to construct and edit the hierarchical task model. The task model is structured by the composite and the leaf tasks (interaction and action tasks). These tasks will be linked to components of the domain and presentation models by the UI developer.

Once the UI developer has made the links between the models, the UI code is automatically generated.

2.2.2 Goliath: A Model-and-Agent Based Environment to Design User Interfaces

Goliath [Juli04a, Juli04b] uses a set of declarative models including application, dialogue, and presentation models to generate the runnable interfaces. To facilitate the use of models, the design and run-time environment has to rely explicitly on expert's knowledge. This knowledge is encapsulated in a multi-agent system based on the eco-resolution algorithm to build the models.

2.2.2.1 Goliath's models

The models used in Goliath are application, dialogue, and presentation models.

- The application model is used to describe application data, functions, notifications, re-conditions, and consequences of the function-calls on the application. An application data term is the output of a function call or a part of this output if the function has several out parameters.
- An **application model** is structured as follows:

```
<application-model> ::= functional-core <functional-core-name> (  
  types : <type-decl-list>  
  data-functions : <function-decl-list>  
  side-effects-functions : <function-decl-list>  
  pre-conditions : <preconditions-decl-list>  
  notifications : <function-decl-list>  
  subscribe-notifications : <subscribe-list> )
```

- The Dialogue model is used to describe user interface in terms of abstract containers and the relationship between these containers. An abstract container is represented by the views (What application data is displayed) and the operations (what functions may be triggered by users).

An **abstract container** is structured as follows:

```

<abstract-container> ::= <ac-name> use ( none | <presentation-element-name> ) (
    variables = <localvar-decl-list> ,
    signals = <signal-decl-list> ,
    views = <view-list> ,
    operations = <operation-list> )

```

A **View** is structured as follows:

```

<view> ::= <view-name> (
    update = ( <condition-decl-list> ) ,
    pdata = <presentation-data> ,
    data-term = <application-data> ,
    default-value = ( <data-value> | none ) ,
    error-signals = ( <fire-signal-list> ) )

```

- The presentation model is used to describe the native interface elements provided by toolkits and presentation elements defined by the developer. Each presentation element has an interface part; it is defined by actions and data. An example of presentation element is represented as following:

```

contact:Contact (
    status = EDIT, actions = ( contactModified ),
    data-content = (
        civility:Civility (
            status = DISPLAY, actions = ( ),
            data-content = none)
        firstName:String (
            status = DISPLAY, actions = ( ),
            data-content = none)
        lastName:String (
            status = DISPLAY, actions = |( ),
            data-content = none)
        birthDate:Date (
            status = EDIT, actions = ( ),
            data-content = ( ... ) ) ) )

```

2.2.2.2 Goliath and its agents

The multi-agent system is applied in Goliath because it has dynamic and adaptive abilities. An agent has autonomy (based on its own state, an agent can make the decisions), reactivity (once an agent has been situated in an environment, it

is able to perceive this environment, to respond to changes that occur in this environment), and social ability (the agents can interact with each other by sending the message to achieve their goals) properties.

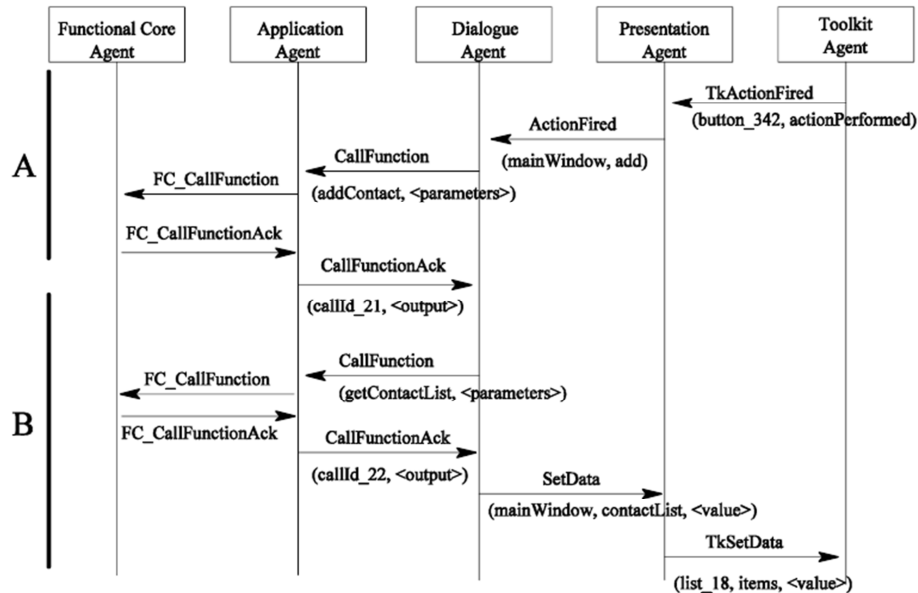


Figure 2-4: The five agents of Goliath (From [Juli04a])

Goliath has five different agents including *Functional core*, *Application*, *Dialogue*, *Presentation* and *Toolkit*. Each agent is composed of an extensible set of services which allows running an interface from the models. These agents communicate by sending the messages each others. See Figure 2-4.

The *Function Core* and the *Toolkit* agents are wrapper agents. They encapsulate the functional core and the toolkit. These agents provide a standard interface to communicate with the functional core and the toolkit, independently of their implementation language. The *Presentation* agent interacts with the *Toolkit* agent to generate automatically the physical presentation. The *Application* agent allows providing higher level functions from the functional core. The *Dialogue* agent manages the relationships between the presentation and the application.

2.2.2.3 Goliath's Tool

The tool of Goliath has been implemented in Calm [INRIA]. Each agent of Goliath is described through a module. The developer interacts with these agents through a user interface (See Figure 2-5).

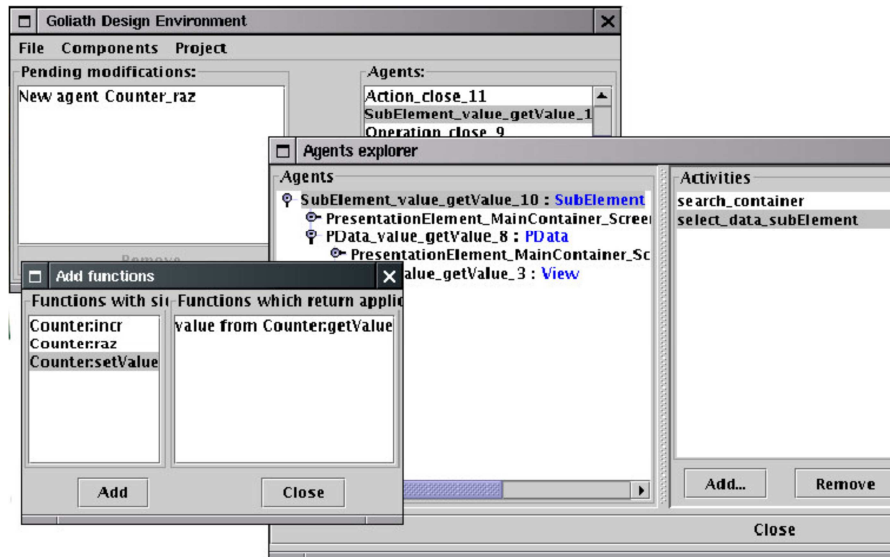


Figure 2-5: Goliath's design tool (From [Juli04b])

Two main objectives of Goliath tool are to generate a first interface from few details automatically and to improve easily the proposed interface. Specially, the tool allows introducing new agents in order to response to changes during the interface generation process. Moreover it allows designer to select an agent, to view its activities and selected techniques, and to change its decisions.

2.2.3 Wold: A Mixed-Initiative Wizard for Producing Multi-Platform User Interfaces

Wold [Stoc04a, Stoc04b] comes from **W**izard **f**or **L**everaging the **D**evelopment of multi-platform user interfaces. Wold produces running user interfaces for the database applications by using a wizard application. The main goal of Wold is to speed up the development life cycle. The user interfaces are structured and described according to characteristics that remain independent of computing platforms.

2.2.3.1 Wold's Domain Model

The domain model of Wold is built based on the information gathered from the schema of a database which is specified by the designer. Its components consist of the tables, the relationships between these tables, the views, the procedures, and also the queries. These components are presented to the end-user in a graphical way. This model is used to specify the control type and build the OpenRst and CloseRst procedures. OpenRst procedure is used to retrieve data from the database and CloseRst procedure is used to update or add new data into the database.

2.2.3.2 Architecture

The main purpose of Wold is that the generated UI supports the end-user to easily communicate with the database and the generated code should be simple and clear. In that sense, the architecture takes into account both the points of view of the end-user and the developer.

The UI generation is performed in seven steps. In the first step, the designer has to specify the database he/she wants to interact with. Once the designer has chosen the data source, the control types are created based on the data type and foreign key attributes in the second step. Then the properties of these controls are specified in the third step; these properties are its name, its value, and its label; *OpenRst* procedure is also built in this step. The purpose of the fourth step is to get the data list for the combo box and the list box which were created in the second step. *CloseRst* procedure is built in the fifth step. The dimension and the position of the controls are specified in the sixth step. Finally, the final user interface is generated in the seventh step.

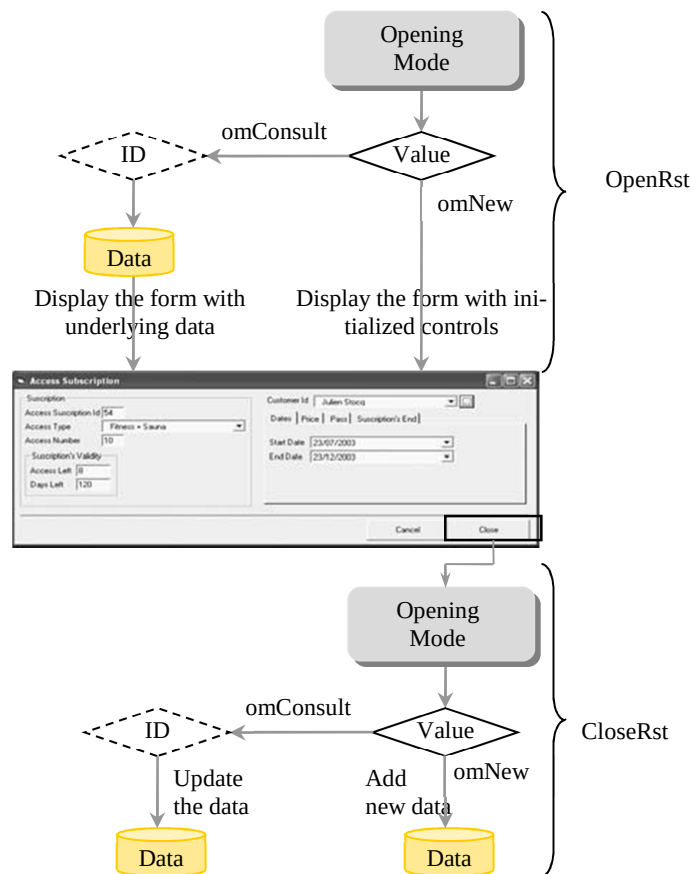


Figure 2-6: The application code generation (From [Stoc04a])

Figure 2-6 depicts the application code generation process. Once, the UI has been specified, Wold system will generate the application code for displaying the data, updating the data, and creating new data. This process starts with determining the *Opening Mode* and ends with updating or creating the data.

2.2.3.3 Wold Tool

Wold tool [Stoc04b] has been developed in Visual Basic. It produces the UI code for both the Microsoft Visual Basic and the eMbedded Visual Basic platforms. Its goal is to generate a ready-to-run user interface.

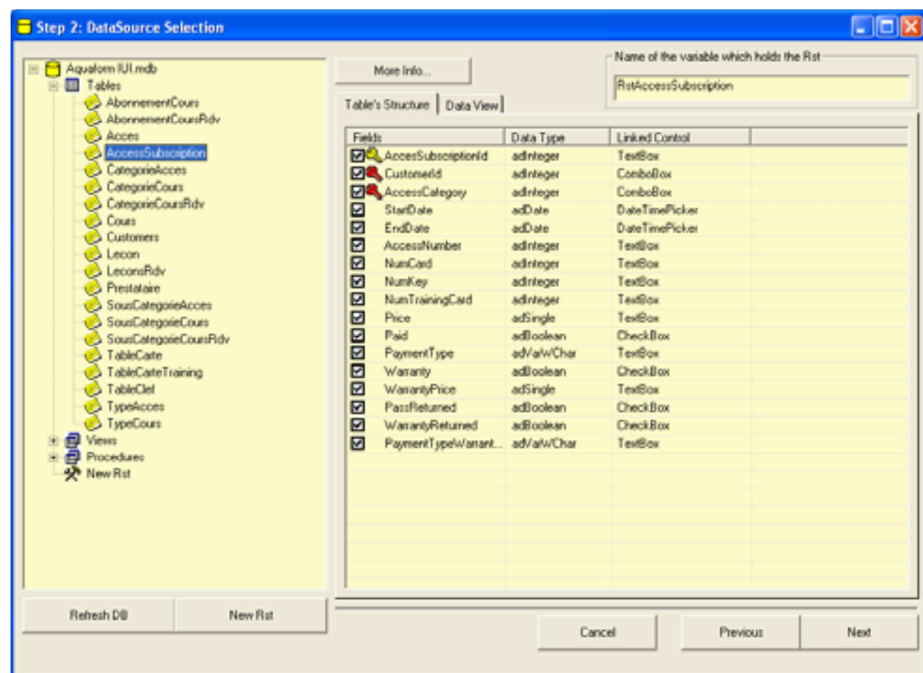


Figure 2-7: Domain model contained in database structure (From [Stoc04b])

Wold tool is a wizard tool considered to perform seven following steps:

- Step 1: The designer starts with selecting the database he/she wants to interact with.
- Step 2: In this step, he/she has to select the data sources including the tables, the views, the procedures, and the queries. Based on the attributes of the fields in the selected data source (such as data type, foreign key), the control types are specified (Figure 2-7).
- Step 3: The *OpenRst* procedure is built to determine the position of control in the form, control label, the value of the control when the form is opened.

Approaches to Automatic User Interface Generation

- Step 4: Once the combo-box/list-box has been created, the designer builds the datasource (a list of values) for this control.
- Step 5: To save the change of the data, a *CloseRst* procedure is built in this step based on the attributes of field such as: a name, linked control, data type, an assigned value.
- Step 6: The size of the controls is specified based on the length attribute of linked field in this step.
- Step 7: The controls are lay-outed based the notion *area* to align and size the controls between themselves. An *area* is a portion of the form which presents a set of related information's to the final user.

2.2.4 Oracle JDeveloper and MS Access

A common property of Integrated Development Environment (IDEs), such as Oracle JDeveloper and Microsoft Access, is that their UIs are generated based on only the domain model and in an inflexible way: usually only one possible UI is generated since only one model is used and one possible layout is produced for the same reasons. In addition, the layout of the generated UI is often simplistic: all fields are arranged one after another without any task consideration. In order to create the UIs, the designer has to select the objects or attributes from a domain model which relate to these UIs.

The advantage of MS Access is that it is easy to use; the designer generates UI by using form wizards. Its disadvantage is that the designer cannot select the function that the generated UI will perform such as a function for modifying data, displaying data, deleting data or searching data. Usually, the generated UI aims to display, insert and modify data but not search or delete or even any other function. If she wants to implement these two functions, she can do it with a builder wizard once the UI has been generated. Therefore, MS Access always generates the same UI for the same selected attributes while the required functions are different.

With Oracle JDeveloper, the designer can work in a way more flexible, he/she can create UIs by dragging the domain attributes and dropping them into a form. To create UIs, firstly, the developer has to make a connection to a database to create the domain model called EJB diagram (in Oracle JDeveloper). With Oracle JDeveloper, the developer can choose a database type such as Oracle, My SQL, and SQL Server. Moreover, UIs that are created are richer; beside of creating the general controls, it also allows create the graph controls. Like in most other applications, UIs in Oracle JDeveloper are created based on the domain attributes selected by developer. Unlike MS Access, Oracle JDeveloper allows the developer to select the function that the generated UI will perform. It typically aims to create UI for Web service-oriented applications in Java.

To give the developer more choices in their working environment, DB-USE allows him/her to select not only the database type but also the programming language. Its purpose is to create UIs that can be executed in the different platforms.

Beside of using the domain model, in DB-USE, the task and user models are also used to create UI. So DB-USE does not only consider how to create UI but also the goal that this UI achieves and the preference of its end-users.

- The task model used to create UI shows that this UI aims to a goal. Moreover, the task model helps the developer understand the purpose of creating UI through task name, task type, action task ... so that he/she can select the domain's attribute easily and correctly.
- The user model is used to create UI that considers the preference of the end-users; so DB-USE generates the different UIs for different end-users.

With DB-USE, the UI is generated based on not only the domain model but also the task and user models.

2.3 Model-Based User Interface Development Environment: A Survey

MB-UIDEs may be compared with respect to the number and expressiveness of their models and the design tools that construct and make use of the model-based information.

2.3.1 Declarative Models: Generic Definitions

Different MB-UIDEs use different set of models to generate the user interface. These models are differently described in the various MB-UIDEs. For example: Teallach's domain model describes both database and non-database objects while Trident's domain model describes the domain's objects. In this Section, we present some generic models used by the MB-UIDEs discussed in this thesis. Table 2-1 will help the reader understand the definitions of the declarative models. These definitions are generic and can be modified by different concrete environments in which these models are used.

Model	Description
Presentation	The presentation model is used to describe how the interface appears to the user. This model allows the designer to specify both static interface components such as button, text field, menu and dynamic interface components. Dynamic interface components are used to display the application-dependent data;

they can be changed during run-time.

Task	Task model is used to describe the tasks that the user can perform with a system in order to achieve the goals. A task is described by a goal as well as actions needed to achieve this goal. Goals in a task model specify when a desired state is reached; methods describe procedures to achieve a goal. Especially, in agent orientation, goals are specified according to model and design the autonomous nature and pro-active features of agents.
Domain	Domain model is used to describe the information about an application that is independent of how the objects are displayed, and how the operations are invoked. This information is represented by the objects in a domain and their relationships. The domain relationships are a key for determining the dynamic behavior of user interfaces; the dialogue and the layout specification for UIs are generated based on these relationships. The domain model establishes a vocabulary to access the objects in an application.
User	User model is used to describe the characteristics of the system's users. These characteristics are their abilities, knowledge and styles of information processing.
Dialogue	A dialogue model is used to describe the structure of the dialogues. This model describes the human-computer conversation. It describes interactions, especially when end-users can invoke commands, input data and present data.

Table 2-1: Model's notion

2.3.2 Applications and their Declarative Models

To highlight the importance and need of declarative models in automatic UI generation, we present some major processes of user interface generation based on these models. These processes differ from each other in terms of input information, mostly represented by declarative models and from which the generation is effectively done, the generation target, the generation process itself and the tools.

Most MB-UIEs such as Tritdent, Teallch, Goliath, and Fuse use the task and domain models to specify the user interfaces. Different MB-UIEs define different sets of models to describe the interface. Table 2-2 presents the MB-UIEs and their

models. In the table if a model is used, it is marked with “+”; if not it is marked with “—”; and if it is present but there is no example which makes use of it, it is marked with “+/-”.

MB-UIDE	Models				
	Task	Domain	Presentation	User	Dialogue
Trident	+	+	+	—	—
Teallach	+	+	+	+/-	—
Tadeus	+	+	—	+/-	+
Fuse	+	+	—	+/-	+
Goliath	+	+	+	—	—
Janus	—	+	—	—	—
Mecano	—	+	—	—	—
Tood	+	—	—	—	+
Emode	+	—	+	—	—
Wold	—	+	—	—	—
Mastermind	+	+	+	—	—
Genius	—	+	—	—	+

Table 2-2: The different approaches in MB-UIDE and the used declarative models

Trident [Boda93, Boda94, Boda95a, Boda95b] uses task, application (domain) and presentation models to specify the interface. The task model of Trident is represented by an Activity Chaining Graph; the domain model of Trident is represented by entity-relationship diagram and the presentation model is represented by the abstract user interface objects

Teallach [Grif01, Barc99, Silv00, Barc00, Grif98] uses the task, domain and presentation models to generate the user interfaces. Each model in Teallach defines a view of the information required to generate the interface to a particular application. The domain model describes the underlying application in terms its data and operation; the task model describes what the user can do with the user interface in terms of its dynamic and information processing requirement; and the presentation model indicates how the resulting interface will appear.

In Tadeus [Elwe95], like in most other MB-UIDEs, the UI developer has to create the task model which is represented by a hierarchical structure of the tasks; the domain model which presents the important entities of the application and the user model which presents the characteristics of the users. Specially, the dialogue model is semi-automatically created based on the three created models. The dialogue model is used to describe the interface in term of views and dialogues.

In Fuse [Lonc96, Lonc97], the use of declarative models is similar to that of Tadeus. The task model is used to describe the task hierarchy of the application model. The domain model is used to describe the functions and the data structure of the relevant user interfaces. The user model is used to describe the static and dynamic properties of the user groups and individual users which influence both the UI generation process and the kind and depth of the help offered by the user guidance component. The dialogue model of Fuse is generated based on three models; task, domain and user models. This model is used to describe the transformation of the task, domain and user models.

Goliath [Juli04a, Juli04b] uses declarative models including the application model, the presentation model and the dialogue model to generate the user interfaces. Goliath's application model is used to define the data types and function signatures¹. Besides, this model also supports the definition of pre-condition² and dependency declarations³. The presentation model is used to describe the basic presentation elements that play a part in the interface. Moreover, it allows defining a new element by composing from the existing elements. The salient points of Goliath's presentation model are its dependence on toolkits and its ability to simplify the automation of the aforementioned behavioural aspects. This dialog is used to describe the associations between the functional core and the presentations. The dialogue model is defined in terms of abstract containers which are more concrete than abstract tasks. Each container represents an independent part of the interface linked to a composite presentation element. Besides, each container is also used to describe the operations that the user can trigger and the data which will be displayed to the user.

Janus [Balz95] emphasizes the use of object-oriented domain model to generate the interface. In this model, a domain object is represented by a class and a class is described by its attributes and methods. During the automatic generation process, there is a generated window for each class; the attributes of this class are

¹ Function signatures are the generic functions of an application such as GetContacts(), DisplayContacts().

² Pre-conditions are the commands defined by Goliath which are used to verify the constraints in the database and the needed conditions before performing a task.

³ Dependency declaration tasks also are the commands defined by the procedures of Goliath; but these procedures are used to remind the developer about changing data which is affected by change of the other procedures.

transformed into the controls; and the methods of this class are transformed into the buttons or the menu items.

Like Janus, Mecano [Puer94a, Puer94b, Puer96] uses the domain model to generate both the static layout and the dynamic behavior of an interface. The domain model of Mecano is a representation of the objects in a domain and their relationships. This model is constructed by the class hierarchy. The interface model is generated based on the domain model. This model contains all the facets of an interface design including interface object, presentation, dialog and behavior.

Tood [Mahf05] uses task and dialogue models to generate user interface. Task model is concerned with the description of the user tasks of the system and dialogue model describes the abstract interfaces. These interfaces are specified based on the task model.

Emode [Hein07] uses only task model to specify the user interface.

Genius [Jans93] generates the interface for database oriented applications. This approach uses an existing domain model that is represented as an extended entity relationship model. In order to structure the information contained in the domain model with respect to tasks of the user, the views are defined in the domain model. A view consists of a subset of entities, relationships and attributes of the overall data model. Based on these views the dialogue model is generated. For each view a dialogue is generated.

In Mastermind [Szek95], the user interface is generated based on the task, domain and presentation models. Its task model describes the user's tasks in a hierarchical structure and contains the necessary ordering information to control the user interface at run-time. A task in the task model is specified by the goal of the task, the condition in which the task can be performed, the effects of the task, information requirements of the task and the breakdown into sub-tasks that specifies how the task must be performed. Mastermind's domain model defines the capabilities of the application. It is an extension of the CORBA object model. Mastermind's domain model is not restricted to contain only objects representing the data structure of the application; the interface designer can model new object that combine attributes from other objects in order to better model the user's view of the data. Mastermind's presentation model describes the layout of the user interface including static and dynamic display. A presentation is specified by a name, a prototype, the conditionals and so on. This model allows the specification of automatic presentation update when the application data or the presentation context changes.

2.3.3 Design Tools and Generated UIs

The design tools are very important for modeling the tasks, domain's objects, UI's objects, so on and for generating the user interfaces.

SEGUIA [Vand99] and SIERRA [Vand95] tools developed for the Trident [7] approach. SEGUIA tool semi-automatically generates the user interface by making questions and suggesting responses to the designer. SIERRA tool maintains the usability guidelines for designing user interface, some of which are used in the SEGUIA tool.

MB-UIDE	Tool	Language used for describing generated user interface
Trident	SEGUIA SIERRA	AION/DS User interface language (DSL)
Teallach	Teallach	Java code
Tadeus	Tadeus	Textual description for UIMS
Fuse	FIRE FLUID BOSS PLUG-IN	C++ code
Goliath	Goliath's design tool	Calm
Janus	OOA-Tool	C++ code
Mecano	Browser tool Intelligent designer tool	
Tood	In progress	
Emode	Emode tool	Java D3ML
Wold		
Mastermind	Application modeling suite Task modeling suite Presentation modeling suite	C++ code
Genius	ER diagram editor ER : Entity Relationship	Textual description

Table 2-3: The different approaches in MB-UIDE and their tools

Teallach [Barc99] has been developed completely. This tool provides three separate editors for editing the models including task, domain and presentation models. Most of the operations for modeling are performed in a straightforward way by drag-and-drop or by cut-and-paste metaphors: for instance, a UI model can be created by dragging information from the domain model and dropping them in the corresponding model. While this approach is straightforward, the mapping rules

remain inflexible. The models are made easier and the UI developer does not need to spend a lot of time to design them. The user interface is generated by the tool implemented in Java.

In Tadeus [Elwe95], the UI prototype is automatically generated from the task, domain and dialog models. This prototype is described in terms of the textual description which can be implemented by an existing User Interface Management System (UIMS).

Fuse [Lonc96, Lonc97] consists of BOSS (BedienOberflächen-Spezifikations-System), FLUID (FormaL User Interface Development), PLUG-IN (PLan-based User Guidance for Intelligent Navigation) and FIRE (Formal Interface Requirements Engineering) components. Each of these components may also be used independently. FIRE provides graphical editors for setting up the task, domain and user models. FLUID is used to generate the specification of static and dynamic properties of a logical user interface. Based on this logical user interface, BOSS is used to generate the implementation of the user interface in C++ code. Finally, PLUG-IN is used to generate the user guidance components.

Goliath's design tool provides a graphical editor for modeling the task, domain and presentation models. The interface generated by Goliath is the complete one implemented in Caml [INRIA].

The OOA-Tool (Object-Oriented Analysis) of Janus is presented in [Balz95]. The generated user interface is described in terms of C++ code and is designed to be as ergonomic as possible.

Mecano [Puer94a, Puer94b, Puer96] provides two separate tools, including the browser tool and the intelligent designer tool. The browser tool is used to define, review and inspect the Mecano's domain model. Based on this model, the dialog specifications are generated. These specifications are classified into high-level dialog and low-level dialog. The Intelligent tool is used to create the abstract interface object based on the created dialogs.

The tool of Tood [Mahf05] has not been presented yet in the existing literature.

The Emode [Hein07] tool is divided into a design time environment and run time environment. In design time environment, the developer modelling the model such as task model and dialogue model. In run time environment, the code is generated based the created models. Emode generates code in Java and D3ML [Burm06].

WOLD tool [Stoc04b] is discussed in Section 2.2.3.3.

The tool of Mastermind [Szek95] includes the application modeling suite (used to modeling application model), the task modeling suite (used to create task model) and the presentation modeling suite (used to generate presentation model based on task and domain models). Mastermind generates code for C++.

In Genius [Jans93], the models are created by an ER diagram editor. The generated user interface is described by an UIMS, but the user interface editor is not

Conclusion

presented. The ability of Genius is to generate the executable user interfaces for database-oriented applications.

2.3.4 Design Tools and Generated Application Codes

Table 2-4 depicts the application code generation of approaches. The automatic application code generation is only considered by the later approaches. Most approaches generate the code for performing the generic functions of an application such as login, delete data, add new data, and modify data. Specially, Fuse generates the user guideline of application.

MB-UIDE	Application code
Trident	—
Teallach	Application code
Tadeus	—
Fuse	User guidance component
Goliath	Application code
Janus	Application code
Mecano	—
Tood	—
Emode	—
Wold	Application code
Mastermind	—
Genius	—

Table 2-4: Application code generation of the approaches

2.4 Conclusion

Most approaches discussed above try to generate a complete and executable user interface based on task, domain, user, presentation, and dialog models. Especially, such approaches use the domain model although they define it differently. The domain model plays an important role in user interface generation since it covers the important features of the user interface creation. The goal of our framework, DB-

USE, is not only to generate the executable user interface code but also the UsiXML (User Interface eXtensible Markup Language) [Limb09] specification since it is portable and can be implemented in the different environments.

Beside of user interface generation, some of the approaches have also generated the application code for performing generic application functions such Teallach, Goliath. Most of these functions have a connection with the edition of data in the database such as adding a new data, deleting existing data, modifying data or retrieving data. This is also a goal of DB-USE, but unlike the current approaches, these functions will be automatic created by our framework instead of being created by the designer.

Teallach especially allows designer to create the interface for non-database application by creating domain components which do not derive from the underlying database but from the Java library which still is a limitation of DB-USE.

The lack of use of the user model in most approaches shows that the end-user of the system is not adequately taken into account while they are the human component of the information system who will use the generated user interface. In DB-USE, the user model is used to classify the end-users so that the user interfaces are differently generated for the different groups of users.

The developed tools we have studied do not perform most of the goals they claim to fulfill. For example, Goliath is too simple to cover complex generations, the model editors have never been detailed; Teallach has been developed completely but it seems difficult to use: the task tree is presented in terms of horizontal tree instead of vertical ones with which task trees can be observed and modified easily. To overcome these shortcomings, DB-USE was developed to match the following goals:

- Executing the objectives we have discussed.
- Supporting the designers to make the relations between the task and the domain models easily and quickly with mouse clicks.
- Facilitating the edition of the models by displaying them in terms of graphic objects. Especially, DB-USE's model can be loaded from a CTT task model or created by designer.

Most of the models used in these approaches are newly created. In practice, they can be created based on the existing resources such as a database, task model. In DB-USE, the domain model is created based on a concrete database and task model by transforming the task model created by CTTE.

Specifying the control type of DB-USE is similar to Wold but in a more flexible way. The rules used for specifying the control type are created based not only on the data types, primary key attribute, foreign key attribute, relationships between objects but also the actual needs of users. For example, in Wold, a combo-box/list-box will be created if a foreign key attribute is added; in fact, this does not happen often (for example, ID Department is a foreign key attribute of Employee object so that a combo-box/list-box ID Department is generated for editing the department

Chapter Summary

information, but usually, the end-users work with the department name). In DB-USE, a combo-box/list-box will be created for an attribute if this attribute comes from the foreign object; it may be the foreign key or an attribute determined by the designer (ID Department, or Department name, or Address). In addition, the data of combo-box/list-box is automatically initialized; in Wold, it is initialized by the designer through a specific wizard.

The navigation between the dialogues, views, and frames has been discussed in various approaches such as Tadeus, Genius, Fuse, and Goliath. Usually, it is described in the presentation or dialog model. In DB-USE, it is determined by the developer/designer; but it should/could be automatically made based on the temporal operator between the tasks.

To do that, it is necessary to extend the mapping rules. For example, in Teallach, once an interaction task has been linked to a task node then an editor abstract interaction object is generated in the presentation model. That is not correct since we do not know what is exactly generated, it may be a display or inputter or chooser abstract interaction object.

2.5 Chapter Summary

In this chapter, we have discussed about the advantages and disadvantages of current approaches. Besides, the models used and the UI generated by these approaches have been also discussed. In next chapter, we present the models of DB-USE including task, domain and user models and the UI objects used in this approach; these objects are the concrete interaction object and the final interaction object.

Chapter 3

Conceptual Modelling of Graphical User Interfaces

In this chapter, we define the principal models and elements used in our research. We present the models in Section 3.1: task, domain and user models. User interface objects including concrete interaction objects and final interaction objects are discussed in Section 3.2. Finally, the SKwyRL framework is presented in Section 3.3; this framework is represented by agents, plans, events, and diagrams used to describe the communications between the agents.

3.1 DB-USE's Models

The task, domain and user models will be presented in this section. They are three main models that we use to generate the user interface.

3.1.1 Task Model

The task model records the tasks that potential end-users of the system may need to perform in their jobs, independently of dealing with a particular computer platform [Pate01a, Pate01b, Pate01c]. Many designs of an interactive system were generated to support these tasks.

Task models play a particularly important role since they indicate the logical activities an application should support. A task is an activity that should be performed in order to reach a specific goal. A goal is either a desired modification of the current state or a query to obtain information on the current state. For example, querying the available flights from Paris to London is one task that must be performed in order to book a flight to London (the relative goal). Tasks can range from a very high abstraction level (such as deciding a strategy for solving a problem) to a concrete, action-oriented level (such as selecting a printer).

The task model provides support for modeling both the structure of the tasks and the flow of information between the tasks. The structure of the tasks in the task model is described by the relationships between these tasks. These relationships can be of various types such as temporal and semantic relationships.

To be meaningful, the development of the task model for a new application should be the result of an interdisciplinary discussion involving the various

viewpoints that should be considered (such as those of the designers, developers, end-users, managers, and domain experts). This would be useful in order to obtain user interfaces able to easily support the desired activities, thus implying that the user task model (how users think that the activities should be performed) and the system task model (how the application assumes that activities are performed) match together.

The task models can be useful for the following different purposes [Pate01c]:

- Understanding an application domain.
- Recording the results of interdisciplinary discussions.
- Designing new applications consistent with the user's conceptual model.
- Analyzing and evaluating usability of the interactive systems.
- Supporting the user during a session.
- Documenting interactive software.

Task models have been used in various approaches to support different phases of the software development life cycle such as:

- Requirement analysis in which, through a task analysis, designers identify requirements that should be satisfied in order to perform tasks effectively.
- Design of interactive applications ([Puer99], [Boda95]) in which the goal is to use information contained in logical models to identify the interaction and presentation techniques best suited to support the tasks at hand and the dialogues whose temporal constraints are closest to those of the model.
- Usability evaluation which can be supported in various ways: for example, identifying task efficiency or analyzing logs of user interactions with the support of task models (see, for example, [Lece98]).

Task models are also used for descriptive and prescriptive purposes, for example, to describe how tasks are supported by an existing system or to prescribe how tasks should be supported by a new system to-be. They can be developed at different levels of abstractions (artifacts-independent or related to the use of specific devices).

Based on the pointed advantages above, most MB-UIDEs use the task model to generate the user interface.

3.1.1.1 CTT: ConcurTaskTrees

This thesis uses the task models created by ConcurTaskTrees [Pate01a, Pate01b, Pate02] for generating the user interface. There are many task-based approaches using ConcurTaskTrees notation [Reic08, Espa06, Luyt06, Prib02, Prib07]. In this section, we describe the notion and the aspects of ConcurTaskTrees's task model.

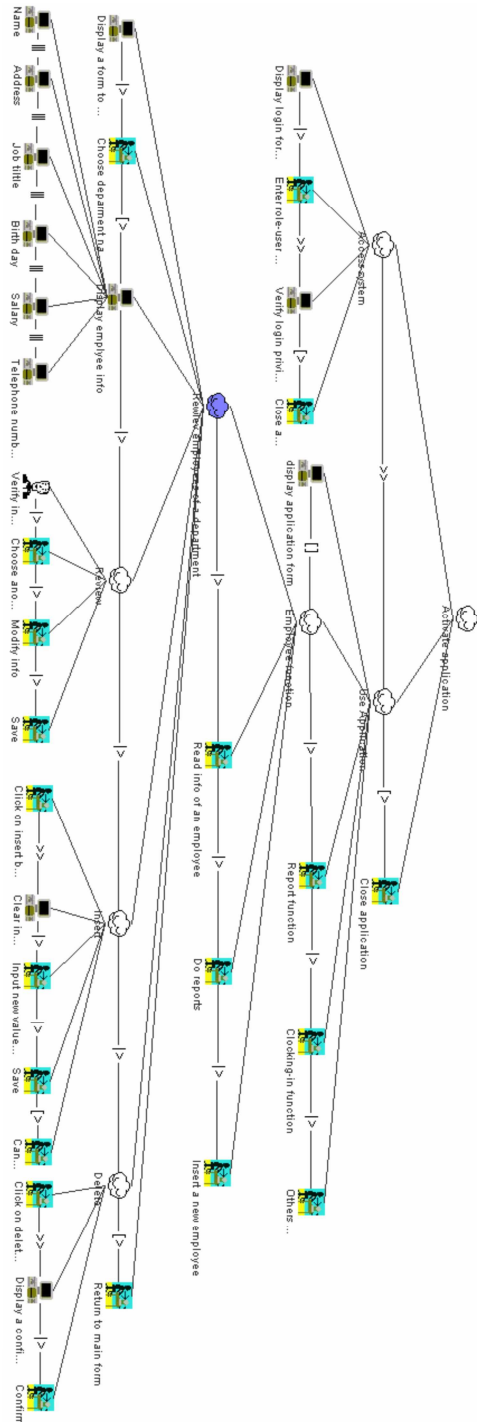


Figure 3-1: A Task Model in ConcurTaskTrees

Figure 3-1 shows a task model created by ConcurTaskTreesEnvironment (CTTE). CTTE is a tool that can be used to support the design of interactive applications, for teaching purposes, and to produce interactive documentation of applications. This tool was developed by the researchers of ConcurTaskTrees.

The task model of ConcurTaskTrees is built at different levels of abstraction which depend on the characteristics and attributes that the notation of a task model could cover. The main purpose of ConcurTaskTree is to be an easy-to-use notation that can support the design of real industrial applications of medium to large dimensions. At each level the task specification is built in two steps: first the objects are identified and then the actions which allow the communication among them are defined. *Objects* are entities which are manipulated to perform tasks by the associated actions. *Actions* are associated with objects. Actions can be cognitive, logical, or physical.

A task in the ConcurTaskTree is described by the following attributes:

- Name: Used to identify the task.
- Type: There are five possible types: abstract, user, application, interaction and Cooperation.
- Subtask of: Name of father task.
- Objects: Including name of object, type of object (internal, perceivable), list of input object actions, list of output object actions.
- Iterative: A Boolean type indicating whether the task is iterative.
- First action: The set of possible initial actions is indicated.
- Last action: The set of possible final actions is indicated.

The tasks are classified into five types: user, application, interaction, abstract and cooperation tasks. See Table 3-1

Task type	Description
	<i>User task:</i> performed by the user, it requires cognitive or physical activities without interacting with the system. The user task is associated with some processing performed by the user on information received from the environment. For example: in order to verify the information of the flights, the user reads this information which is displayed by the system. He/She doesn't need to interact with the system.
	<i>Application task:</i> executed by the system. It receives information from the system and it can supply information to the user. This task is activated by the application. For example: displaying the information of the students to the

user after he/she entered their class's name.



Interaction task: performed by user interactions with the system. These interactions are activated by the user. For example: in order to login into the system, the user has to enter his/her user name and password then the system verifies this information and opens a new connection.



Abstract task: is used to describe the complex actions, and its performance does not completely fall into one of the three previous cases. For example: in order to edit the information of the employees; the system will display a list of the employees then the user chooses one of them; once the user has finished the change of the information; the system will save these changes.



Cooperation task: performed by the cooperation of the different users. In other words, the task is only completed once the user cooperates with the other users. For example: *Sending message* task; in this task, there are minimum two persons including the sender and the receiver. The message is sent from the sender to receiver and this task is completed when the message is received by receiver.

Table 3-1: Types of task.

The temporal operators are used to link the subtasks at the same abstract level. ConcurTaskTrees provides a rich set of temporal operators which allows designers to describe flexible, and more realistic, behaviours. These temporal operators are described in the following table:

Operator	Description
$T1 \parallel T2$	Interleaving: the actions of the two tasks can be performed in any order.
$T1 \sqcup T2$	Choice: It is possible to choose from a set of tasks and, once the choice has been made, the task chosen can be performed; the other tasks are not available at least until the chosen task has been terminated.
$T1 \mid T2$	Order independent: both tasks have to be performed but once one is started it has to be finished before starting the second one.
$T1 \parallel\!\!\!\sqcup T2$	Synchronization: the two tasks have to synchronize on some actions in order to exchange information.

T1 >> T2	Enabling: when the first task is terminated then the second task is activated.
T1[]>>T2	Enabling with information passing: in this case we want to highlight that when T1 task terminates it provides some value for task T2 besides activating it.
T1 [> T2	Deactivation: when one action from the second task occurs the first task is deactivated.
T1 > T2	Interruption: T2 can interrupt T1, T1 can resume at the end of T2.
T1*	Iteration: the task is iterative.
T1(n)	Finite iteration: how many times the task will be performed is specified.
[T1]	Optional task: its performance is not mandatory.
T	Recursion: the possibility to include in the task specification the task itself.

Table 3-2: Temporal Operators Used in ConcurTaskTrees

The relationships between the father task and its subtasks are specified based on the use of the operators described in Table 3-2. If these operators are used in a hierarchical context, we can achieve great flexibility by being able to specify different behaviours. Based on Figure 3-1, the same tasks are involved but we have two possibilities: In the first case, if the *Finite Iteration* is assigned to task father then the tasks T1 and T2 can be performed in any order but before performing them again, both tasks T1 and T2 have to be finished. In the second case, if the *Finite Iteration* is assigned to all of the subtasks then once one task is finished it can be performed again without waiting for the completion of the other task. These relationships can be classed as follows:

Relationship	Description
Sequence	The subtasks must be performed in the specified order; all subtasks must be completed before the task's goal is considered to be achieved.
Order-independence	The subtasks may be performed in any order; all subtasks must be completed before the task's goal is considered to be achieved.

Repetition	The sub-tasks are repeated a specific number of times or until a condition is satisfied.
Concurrence	The subtasks are performed in parallel; all subtasks must be completed before the task's goal is considered achieved. Two kinds of concurrent task are distinguished usually by a flag: truly concurrent tasks occur at the same time, while interleaved tasks have only one sub-task in common at a particular time, although all are progressing.
Choice	The user must decide which one of the subtasks is to be performed; the chosen subtask must be completed before the task's goal is considered to be achieved.
Option	Enabling with information passing: in this case we want to highlight that when T1 task terminates it provides some value for task T2 besides activating it.
Condition	A choice between sub-tasks exists that is dependent on a specified condition.

Table 3-3: Relationships table.

3.1.1.2 DB-USE's Task Model: A Task Model for User Interface Design

In this Section, we present DB-USE's task model which is derived from a task model created by CTT. This task model will be used in user interface generation.

The task model described by CTT is created at the analysis level. In order to reuse it in our process, it has to be transformed into a task model at the design level. At this level, the task model has to describe the function the user can do with an interface, how the user can interact with the system. DB-USE's task model is used to derive the interface. Its task types differ from CTT's and are defined in Table 3-5

Transforming the abstract, interaction, application and user tasks of the task model at the analysis level to action and information tasks of the task model at the design level is discussed in Section 4.2.2.2.i. In this Section, we only discuss about the notion of DB-USE's task model and its structure.

In DB-USE, a task is defined by the task ID, the task name, the parent task ID, the task type, the action type, the right task ID, and the temporal operator (See Table 3-4).

Task model of DB-USE is specially defined as a flexible one that allows the designer/developer to expand this model by creating the new task types which may be needed for new goals in future.

DB-USE's Models

Attribute name	Attribute type	Description
Task ID	long	Used to identify the tasks.
Task name	Text (50)	Used to name the task.
Parent task ID	long	Used to specify its parent task.
Task type	Text(10)	Used to specify the type of task.
Action type	Text(1)	Used to recognize the action that the task has to perform.
Right task ID	long	Used to specify its right task.
Temporal operator	Text(20)	Used to specify the temporal operator between this task and its right task.

Table 3-4: Task Attributes

There are six different action types used in DB-USE task model. They are defined in Table 3-5. The list of action types can be changed by the developer/designer by adding a new action type, modifying the existing action types, and deleting an existing one.

Symbol	Action type	Description
I	Indeterminacy	The action type has not been determined.
C	Create, New, Add, Insert	The action of creating new data.
D	Delete, Remove	The action of deleting existing data.
U	Update, Modify, Edit, Change, Alter	The action of modifying the data.
S	Search, Find, Filter	The action of searching the data
R	Read, Select, Display, Review, Show, View	The action of showing the data

Table 3-5: Action types

There are two types of task including Action and Information tasks. They are defined in Table 3-6.

Task type	Description
Action task	Is a task used to describe the end-user's command to the system such as close a dialogue, delete a data record, search information, open a dialogue and so on.
Information task	Is a task which is used to describe the display of information to the end-user or the reception of the information from the end-user.

Table 3-6: Task types

a. XML format of DB-USE's task model

The task model of DB-USE is stored using XML specifications. The task model begins with `<!DB-USE Task Model>` and ends with `</DB-USE Task Model>`; one task in the task model begins with `<Task TaskID="" TaskType="">` and ends with `</Task>`.

```
<?xml version= "1.0" encoding="utf-8"?>
<!DB-USE Task Model>
  <Task TaskID="10" TaskType="Information">
    <TaskName> Enter user name </TaskName>
    <ParentTaskID> 9 </ParentTaskID>
    <ParentTaskName> Login </ParentTaskName>
    <RightTaskID> 11 </RightTaskID>
    <ActionType> I </ActionType>
    <Operator> Interleaving </Operator>
  </Task>

  ...

  <Task TaskID="102" TaskType="Action ">
    <TaskName> Delete User</TaskName>
    <ParentTaskID> 86 </ParentTaskID>
    <ParentTaskName> Manage users </ParentTaskName>
    <RightTaskID> 103 </RightTaskID>
    <ActionType> D </ActionType>
    <Operator> Enabling </Operator>
  </Task>

  ...

</DB-USE Task Model>
```

In the new task model, the structure, the relationship between the supertask and its subtasks, the temporal operators are all reused for the syntactic and semantic aspects (See Figure 3-2).

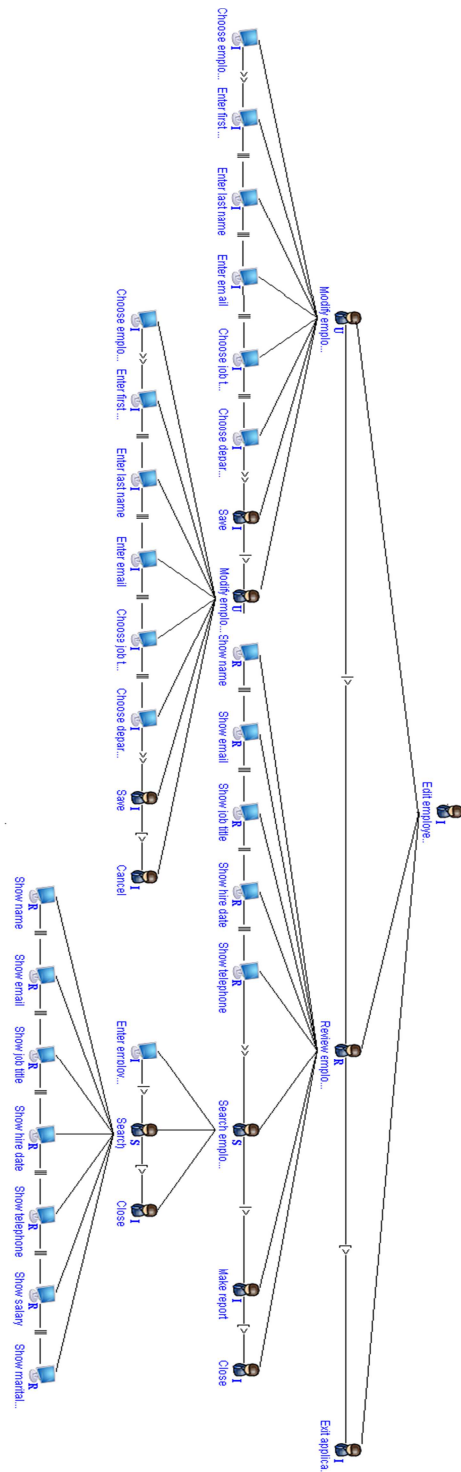


Figure 3-2: DB-USE Task Model

Figure 3-2 depicts DB-USE task model. The action task is represented by icon  and the information task is represented by icon . The action type is also represented by the symbol I, C, D, U, S, R at the right bottom of the icons.

b. Task library

In practice, the same tasks can be used in different task models. One way to reuse these tasks is through the use of task patterns. A task pattern defines a reusable and generic task structure that encapsulates a well-defined functionality for a recurrent design problem in task modeling [Wurd07]. The task pattern technique has been presented in [Rade06, Rade07].

The idea of building a task library is to create a set of task patterns that can be reused in task models. These task patterns are created by both the developers and the users of DB-USE.

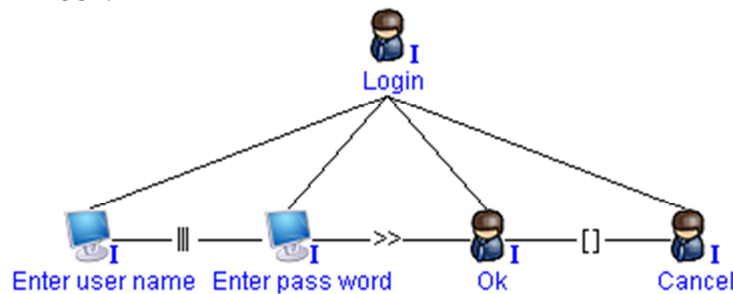


Figure 3-3: Login task pattern

```
<?xml version= "1.0" encoding="utf-8"?>
<!DB-USE Task Pattern>
  <Task TaskID="1" TaskType="Action ">
    <TaskName> Enter user name </TaskName>
    <ParentTaskID> 0 </ParentTaskID>
    <ParentTaskName> </ParentTaskName>
    <RightTaskID> 0 </RightTaskID>
    <ActionType> I </ActionType>
    <Operator> </Operator>
  </Task>

  ...

  <Task TaskID="4" TaskType="Action ">
    <TaskName> Ok </TaskName>
    <ParentTaskID> 1 </ParentTaskID>
    <ParentTaskName> Login </ParentTaskName>
    <RightTaskID> 5 </RightTaskID>
    <ActionType> I </ActionType>
    <Operator> Resume </Operator>
  </Task>
</DB-USE Task Pattern>
```

DB-USE's Models

One task pattern of DB-USE is a subtask in the task model so that can have the same structure task model. Figure 3-3 depicts the *Login* task pattern. Task patterns of DB-USE can be created manually or by extracting from an existing task model.

The developer and the user of DB-USE can classify pattern tasks into two different types including generic and specific task patterns.

- Generic task patterns can be used in the task models of different domains.
- Specific task patterns can be used in the task model of specific domains.

Task pattern	Description	Used to
<i>Login</i>	Provides the user name and password.	Login to the system.
<i>Provide personnel information</i>	Provides the first name, last name, address, phone number, email address; make a confirmation.	Create un account (a client, a user, a patient, a criminal...).
<i>Search user</i>	Provides the ID user or user name.	Search user based on the ID user and user name.
<i>Display information of a user</i>	Displays the first name, last name, address, phone number, email address.	Display the information of one person (Client, user, patient...).
<i>Ask user for the chance of data</i>	Data is checked by the system, a message is displayed to the user to ask him/her to save data, Data is saved by the system	Check if the data is changed then ask the user if he/she wants to save it.
<i>Check if the user exists</i>	Provides the user name or user ID, make a search.	Check if the user exists in the system.
<i>Make a contact</i>	Provides email address of sender and of receiver, content of the email, send email, save email	Send a request or an answer.
<i>Change contact information</i>	Provides new email address or new phone number, make a confirmation.	Change email address or phone number.

Table 3-7: Generic task patterns for electronic commerce, eManagement applications

Conceptual Modelling of Graphical User Interfaces

To illustrate these task patterns, we have created some task patterns for electronic commerce and eManagement applications. The generic task patterns for two these application types are described in Table 3-7 and the specific task patterns of electronic commerce application are described in Table 3-8. Specially, these task patterns in terms of DB-USE task tree are presented in the appendix.

Task pattern	Description	Used to
<i>Create a new product</i>	Provides the code, mark, name of product, category, price, amount, description.	Create or insert a new product.
<i>Make a report of inventory</i>	Displays the date, code of product, name of product, category, and amount.	Check the current amount of a product.
<i>Display a list of products</i>	Displays the product name, image of product, see detail and buy.	List the current products.
<i>Display a product in details</i>	Displays the product name, image of product, price, and amount, choose shipping type, description, send an email, buy, and see other products.	See all of information of a product.
<i>Make an order</i>	Provides the product name, display price, provide quantity, display price total, select delivery type, make a payment.	Order the products.
<i>Make a payment</i>	Displays payment amount, provide payment type, account holder, account number, Address, expiration date.	Pay the bill.

Table 3-8: Specific task patterns for electronic application

Some important points for creating the task model are listed as follows

Determine the goal of the task

Determining the goal of the task plays an important role in the creation of the task model. Based on the goal, the type of task is specified. For example: if the goal of a task is to delete data in the database then the task type is an action task; if the

goal of a task is to display the value to the end-user then the task type is an information task.

Name the task based on its goal

Naming the task based on its goal helps the developer/designer understand what the task has to perform. For example: if the goal of the task is to display data to end-user then the task should be named *Display value* or *Display data* or in detail *Display name*.

Determine action type

Beside the determination of the task type, the developer/designer needs to determine the action type. The action type represents the function which will be performed to obtain the goal of the task. So the developer/designer can choose or create the exact functions for the tasks when he/she implements these tasks.

3.1.2 DB-USE's Domain Model

A domain model is a representation of the objects in a domain, their relationships, and the generic functions. The information in the domain model is basically the data model where the data objects are defined, including the relationships between these data objects and other information that is pertinent to the relationships such as business logic. A domain model may thus include a data model of the domain. Similarly to interface models providing developers with access to a higher-level representation of the design knowledge, domain models also allow access to a level of representation higher than that of data models. Whereas data models establish a vocabulary to access the data structures of an application, domain models establish a vocabulary to access the objects in an application domain [Ange94].

The domain model should not only represent data, but also business logic: it should take care of what can and what cannot be done with the data, and how the data is connected. It should also derive data not stored in the database. However, it should not contain presentation behavior for the user interface. It may contain persistency, but if the application has to run on multiple databases, all database specific behavior should be delegated to a database abstraction layer.

In other words, the domain model consists of one or more abstractions of the system. It defines the aspects of the application which contain facts about the domain such as the objects, their attributes and the relationships between objects. The domain model is the designer's definition of the aspects of the application relevant to the needs of the adaptive system. For example, the domain model of Employee Management in Figure 3-3 is a representation of certain aspects of Employee, Department, Flight, Job history, Location, Country, Job and the generic functions.

Important benefits of a domain model are the description and constraints specification of the system scope. The domain model can be effectively used to verify and validate the understanding of the problem domain among various stakeholders of the project. It is especially helpful as a communication tool and a focusing point between technical and business teams.

The domain model will serve the following purposes.

- It will define the aspects of the system which are adaptive by describing alternative representations of domain features.
- It will hold the characteristics of the application, which are measurable, so that they can be evaluated for effectiveness against the required criteria.

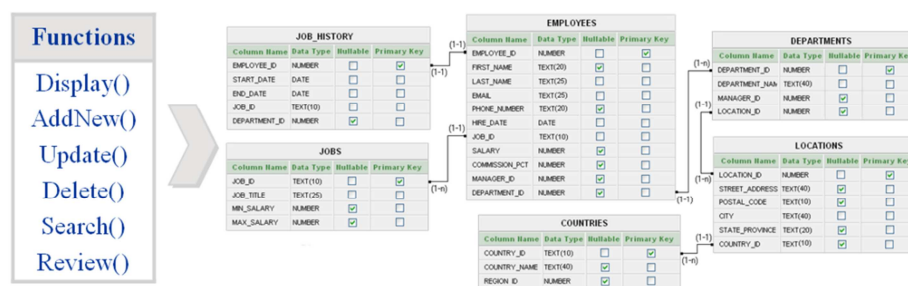


Figure 3-4: Employee Management Domain Model illustrating the domain objects and its generic functions

In the domain model of DB-USE, there are two main parts. The first part contains the domain's objects, their attributes and the relationships between these objects which are created based on a certain database (See Figure 3-4). These components are used to identify the attributes of a concrete user interface. Besides, in order to obtain the desired behavior of a database application task, the generic functions are also defined in the second part of domain model; these functions are, for instance, Display(), AddNew(), Update(), Delete(), Search(), Review(), Cancel(), Exit(), and Default() functions. These functions have been created based on the actual needs of the end-users and based on a list of canonical task types in [Gonz09].

A class diagram or data model can provide most the information for building domain objects but they do not support performing the functions defined by DB-USE. In DB-USE, the database is chosen to build the domain model instead of the class diagram or data model since the database is used to not only provide the information to build the domain objects but also to run the functions created by DB-USE.

The goals of the functions are showed in Table 3-9. These functions will be discussed in detail in Section 4.2.2.4.

DB-USE's Models

Function	Description
<i>Display()</i>	Used to select the data stored in the database and to display this data to the user
<i>AddNew()</i>	Used to insert a data record into the database
<i>Update()</i>	Used to modify the data of an object in the database
<i>Delete()</i>	Used to delete the data records of an object in the database
<i>Search()</i>	Used to filter the data records based on some search conditions which are determined by the user
<i>Review()</i>	Used to review the data records by displaying the first, next, previous and last record
<i>Cancel()</i>	Used to reset the data for the controls
<i>Exit()</i>	Used to close the dialogue or frame
<i>Default()</i>	Used to represent the functions which are not specified by DB-USE

Table 3-9: Defined Functions.

Unlike Teallach and Goliath, the generic application functions are simply defined in the domain model and they will be analyzed and built more in details by the system once the function is linked to a concrete task in a concrete context.

3.1.3 DB-USE's User Model

When users want to use a program, they do not come with a completely clean requirement list. They have some expectations of how they think the program is going to work. If they have used similar software before, they will suppose the system-to-be to work like other software. If they have not used any software before, they will think that the software-to-be conforms to certain common conventions. They may have intelligent guesses about how the UI is going to work. This example of behavior and expectation based on the background and knowledge of the user is called the user model: it is their mental understanding of what the program is doing or going to do for them. [Schl96]

The user model describes the characteristics of the desired users or groups of users such as experience, skill, knowledge, character. The main purpose of a user model is to support the designer in creating the user interfaces which tend to the preference of the user. A user model contains the rules to reason from a certain user

Conceptual Modelling of Graphical User Interfaces

characteristic to a user interface design decision (e.g., selection of interaction objects).

The user model captures capabilities and limitations of the user population, for instance the kind of interaction techniques that are available for visually disabled people differs from the techniques for other human beings. The user model plays an important role in the user interface design; based on it, the designer will specify a complex or a simple user interface. For instance, creating the user interface for the people who have experience with the software applications is different from creating the user interface for newcomers who have never used the computer.

User characteristics can be classified as application-independent or -dependent. Application-independent characteristics include preferences, capabilities, psychomotor skills, etc. Application-dependent characteristics include goals, knowledge of system and application, etc.

In the DB-USE's user model, the users of the system are grouped into three groups (Expert, Intermediate and Novice) based on the user characteristics mentioned above.

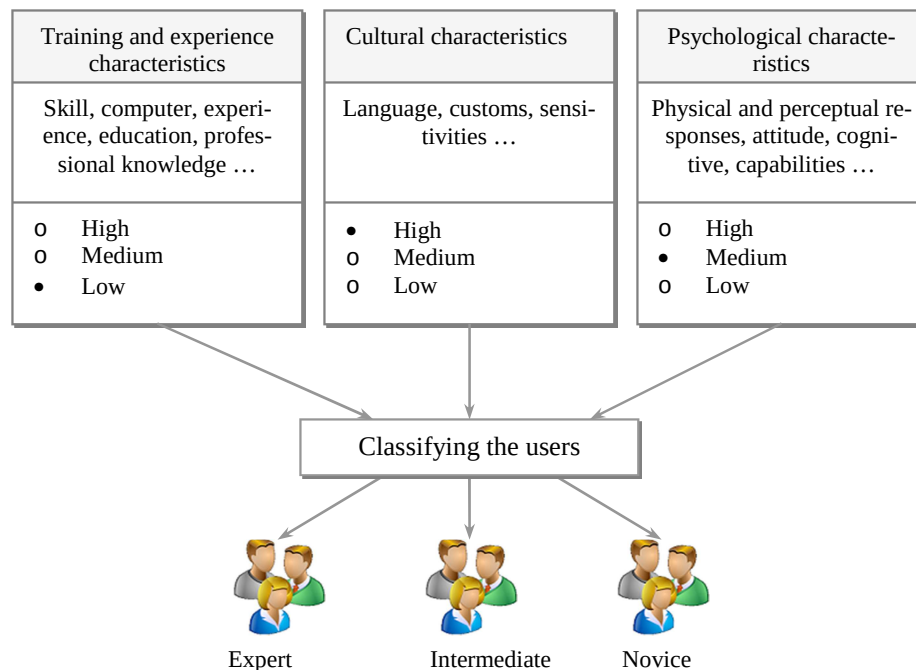


Figure 3-5: Grouping users based on user characteristics

Figure 3-5 depicts DB-USE's user model which describes the user's experience, cultural and psychological characteristics. Then users are grouped based on these characteristics.

Here are some important points needed for creating the user model.

Be a user of the system

The Business Analyst should visit the customer site and put himself into the user's position so that he/she has a feel about how to gather correct requirements. Besides, he/she can question the users on their jobs. For example, let's consider that we need to describe a doctor's job including the management of patient's documents, prescriptions, patient's appointments... Since we are not doctors, we do not have the professional knowledge and do not know the working process of a doctor. We should visit the hospital where the product will be used and imagine what a doctor does and how. Through experience, the Business Analyst can observe and question to better understand the doctor's job.

Determine user experience

Defining the user experience is very difficult. One reason is that humans experience all the time, because they live in interaction with environment and other people. Basically, a human constantly experiences when he/she is interacted with a product. However, a concept of user experience is not so simple. Several aspects relating to users, products and environments need to be considered as well. [Olse93].

Currently, the term "experience" has been defined at three different levels.

The first one is experience, which means a continuous stream that flows through human's mind. People acknowledge the passing of experience by self-talking or self-narration.

The second, humans experience things. For example, they observe, think or desire something or they start and stop to do something. However, they do not stop because experience has reached but they stop because something else came into their mind or something interrupted them. Hence, they have got some experience in the context of use. This experience is not relating to material experiences. For instance, when a popup dialogue is displayed on the screen; with the experience of how to close it, the user has to click on one of the buttons of this dialogue.

The third definition of experience is *experience as story*. Stories are ways to organize and remember experiences and they enable humans to communicate experiences in different situations to particular people.

Therefore, using user experience in user interface design is a necessary condition to help the user to use the software products. The following experiences are applied to the user interface design.

Determine the user's characteristics

It is necessary to know the "class" (i.e., type) of the people who will be using the system. The created UIs are the most useful if they fit the requirements of most of the users. In some situations this isn't easy since it is impossible to identify these

users as concrete individuals. Different groups of users will vary in their characteristics. Whenever possible, the BA should try to include a range of users as large as he/she can, but think carefully about designing the UI for everyone. If a user group has many different characteristics, he/she should satisfy the most and identify what characteristics to address in priority.

The user characteristics are defined based on the user's work experience, educational level, age, previous computer experience and so on. It is possible to anticipate their learning difficulties to some extent and to better set appropriate limits for the complexity of the user interface. For instance, we have two ways to open a student's name dialogue; one is to click on students menu then choose the student's name menu, another is to click on a student's name button. If the user has no computer experience the button option is the best one. He can understand exactly, quickly what he/she needs to do to open a dialogue to see the student's name.

Determine the user's capabilities of using, study

To determine the user's capabilities of using and study a new product the BA should consider the following user aspects. They have already used some products which are similar to new one. They would like to use a new product.

Group the users

And finally, based on the important points mentioned above the users can be grouped in terms of the knowledge of systems required or associated with jobs and roles they occupy; in terms of the user's age as to specify their capability of using the product,

3.1.2 User Interface

User Interface (UI) is that subset of a software system that interacts with the user of the system. In practice, it consists of forms, windows, dialogues ... with text fields, combo boxes, buttons, options, check boxes and so on.

Humans directly or indirectly interact with computers through user interfaces to exchange information, semantics, data and knowledge. Users are responsible for their behaviors and activities such as clicking the buttons of the mouse, typing and entering characters; the computer is responsible for the computation results which are outputted to users. For instance, if the user wants to make a simple operation such as $6 * 7$, he/she has to tell the computer that he/she wants to know the result of this operation by typing the numbers 6 and 7 through the a number box or text field. The computer displays the result also via a number field or a message box. These controls are called the entities of the user interface and the container which includes them is called the user interface.

In our process, we utilize two types of user interface object. They are the concrete user interface and the final user interface.

3.2.1 Concrete User Interface

A **Concrete User Interface** (CUI) describes a potential user interface after a particular interaction modality has been selected (e.g., graphical, vocal, multimodal). This step is supported by several tools helping designers and developers to edit, build, or sketch a user interface. It concretizes an abstract UI for a given context of use into Concrete Interaction Objects (CIOs) in order to define widgets layout and interface navigation. It abstracts a final UI into a UI definition that is independent of any computing platform. Although a CUI makes explicit the final look and feel of a final UI, it is still a mock-up that runs only within a particular environment. Each tool pursues a particular goal [UsiXML, J Vanderdonckt].

A Concrete UI consists of a hierarchy of Concrete Interaction Objects (CIOs) resulting from mapping the task, domain and user models.

A **Concrete Interaction Object** (CIO) is a graphical object for gathering and displaying data the user can see, feel and manipulate [Vand93]. A CIO is synonymous to a control, a physical interact or, a widget or a presentation object such as text-field, combo-box, check-box, button ... A CIO in our process is defined by the label, control type, and editable attributes such as represented below:

Concrete Interaction Object
<i>Label</i> : The label of the CIO; it will be used to label the control
<i>Control type</i> : The control type which is used to communicate between the user and computer's system
<i>Editable</i> : Yes if this control can be edited by the end-user; otherwise No

A Concrete User Interface abstracts the Final User Interface (see Section 2.6) into a UI definition that is independent of any computing platform.

The generic control types used in DB-USE are presented in Table 3-10.

Control type	Description
Text field	Is a text component that allows the editing of a single line of text.
Text box	Is a multi-line region that displays text.
Combo box	Is a commonly-used GUI widget. It is a combination of a drop-down list and a text field, allowing the user to either type a value

Conceptual Modelling of Graphical User Interfaces

directly into the control or choose from the list of existing options.

Radio button	Is a type of graphical component that allows the user to choose one of a predefined set of options.
Check box	Is a graphical component that can be in either an "on" (true) or "off" (false) state.
Table	Should provide a clear summary record of a collection of data. Tables have a number of columns and rows, depending on the amount of data and the details shown.
Tab	Is a navigational widget for switching between sets of controls.
Menu	Is a list of commands presented to user by the system.
Button	Allows the user to click to perform an action. The button can display both text and images. When the button is clicked, it looks as if it is being pushed in and released

Table 3-10: CIO type table.

This layer assumes a commitment to a class of devices and a choice of modalities. This can include the provision of alternative user interfaces as appropriate to the delivery context (user preferences, device capabilities and environmental conditions). For example: The large screen of a desktop can contain more information or more controls than the small screen of a mobile phone.

The CUI should define some, but not all, aspects of the final presentation. This ensures that there is sufficient flexibility in being able to realize the presentation on a variety of devices and platforms. Application developers should be able to define themes and other policies for guiding the transformation for a particular device/platform. For example, the application can define the type and size of the font.

The CUI can be stored in a user interface markup language such as UIML, UIXML, UsiXML, Microsoft's XAML and Adobe's MXML. These languages are mainly used to declaratively lay out the interface of applications in a way which is independent of the programming language, the environment and the platform, and can also be used to implement the business logic and internet application behaviors. Its goal is to help the application developers make the user interfaces independently from the different platforms or devices.

3.2.2 Final User Interface

A **Final User Interface** (FUI) represents the operational user interface that is running on a special computing platform either by interpretation (e.g. through a web browser) or by execution after compiling the code in an interactive development environment. The FUI is determined based on the concrete user interface. For example, a combo-box is created for a Palm by using Java. On the next level, the code of a user interface is produced from the CUI level.

The FUI is the result of rendering such choices but also considering the specific target environment of the device: it could involve attributes like the type and size of font, location of control and so on. The FUI is the running interface implemented in a specific software environment. In other words, it is the UI which is ran on a particular platform either by interpretation or by execution. Figure 3-6 shows two examples of final user interfaces determined from a CUI in different environments: (a) a dialogue on Windows in Visual Basic and (b) a web page on Windows in ASP.

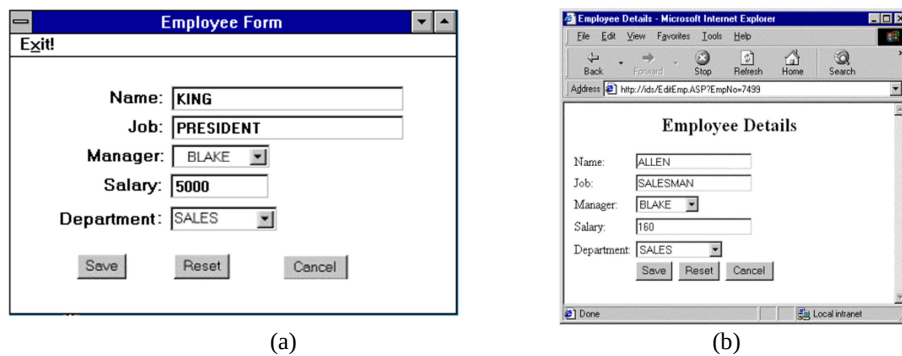


Figure 3-6: FUIs for viewing the employees in two different environments

A FUI consists of a hierarchy of Final Interaction Objects (FIOs) resulting from a transformation from Concrete Interaction Objects in a certain language, on a certain platform and so on. For example, a dialogue in Java or a text-field in Visual Basic.

A **Final Interaction Object** is defined as represented below:

Final Interaction Objects
<i>Label:</i> The label of the control
<i>Control type:</i> The control type is specified in certain platform
<i>Editable:</i> Yes if this control can be edited by the end-user; otherwise No
<i>Location:</i> The location (X, Y) of control in a form or in the screen
<i>Dimension:</i> The dimension of the control, it contains width and height

3.3 The SKwyRL Framework

The SkwyRL [Do05] framework is an agent-oriented framework composed of four dimensions, each one reflecting a particular aspect of a MAS (Multi-Agent System). The first dimension is requirement-driven, the last three dimensions are design-driven. The general concepts proposed by the BDI (Belief-Desire-Intention) agent-oriented models and programming languages are used to define the concepts of the framework. Consequently it makes the passage from design to implementation easier. The framework dimensions may be briefly defined as follows.

3.3.1 The social dimension

This dimension identifies the relevant agents in the system and their goal's interdependencies. As shown in Figure 3-7, for the goal dependency 'Generate the UI code', 'Code generator' is the depender and 'Developer' is the dependee. The fulfillment of 'Generate the UI code' is the responsibility of Code generator. We graphically use the i* notation to visually represent this dimension.

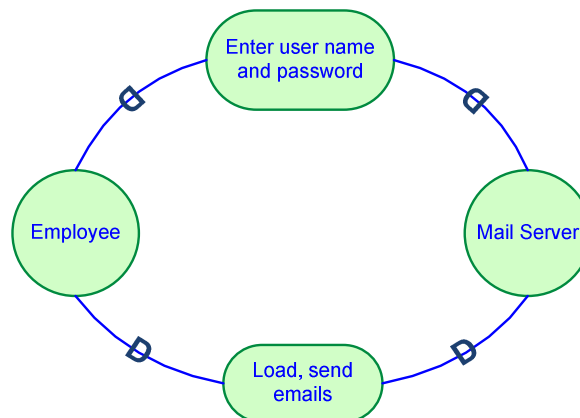


Figure 3-7: Social Diagram – Mail services

3.3.2 The structural dimension

The structural dimension describes the agent in terms of agent-oriented concepts like beliefs, events, plans, and their relationships. The agent concept defines the behavior of the agent such as shown in Figure 3-8:

- The declaration of agent attributes.
- The plans that the agent can execute to obtain its goal.

- The beliefs the agent can use and refer to. The beliefs of an agent can be of type private, class, or global. A private access is restricted to the agent to which the belief belongs. Class access is shared with other agents of the same class, while global access is unrestricted.
- The declaration of agent methods (e.g., the constructor of an agent)

<<Agent>> MailServer
Attribute String Name
Plan Signin() LoadEmails() SendEmail() Signout()
Belief private belief Database
Method private OpenConnection(String UserName, String Password) private LoadEmails() private SendEmail() private Disconnect()

Figure 3-8: Agent – MailServer

<<Plan>> OpenConnection
Event post LoadEmails handle SignIn
Used Belief use Database
Method main() { // Handle a SignIn event // Connect to server mail // Post LoadEmails event }

Figure 3-9: Plan - OpenConnection.

A plan describes a sequence of actions that an agent can take when an event occurs. Plans are structured by the Event part, the Belief part, and the Method part

such as shown by Figure 3-9. The Event part declares events that the plan handles (i.e., events that trigger the execution of the plan) and events that the plan produces. The latter can be either posted (i.e., sent by an agent only to itself) or sent (i.e., sent to other agents). The Belief part declares beliefs that the plan reads and those that it modifies. The Method part describes the plan itself, that is, the actions performed when the plan is executed.

3.3.3 The dynamic dimension

The dynamic dimension models the synchronization mechanisms between events and plans.

A plan can be invoked by an event that it handles and can create new events. Relationships between plans and events are synchronized and modeled with activity diagrams extended for agent-oriented systems. These diagrams specify the events that are created in parallel, the conditions under which events are created, which plans handle which events, and so on.

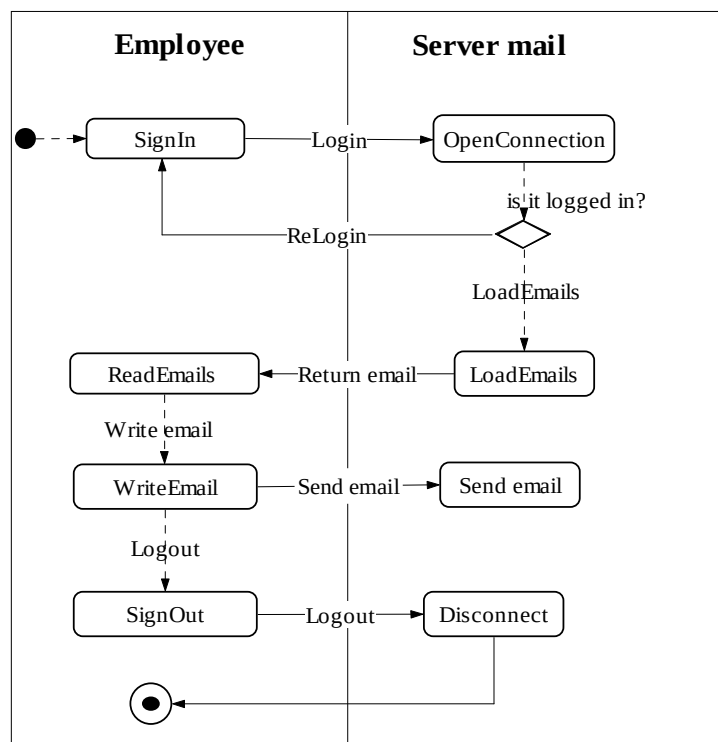


Figure 3-10: Dynamic Diagram – Email services

Figure 3-10 shows an example of a dynamic diagram. Each agent constitutes a swimlane of the diagram (e.g., employee and mail server). Plans (e.g., SignIn,

Chapter Summary

ReadEmails, OpenConnection ...) are represented as round-corner boxes and placed in the swimlane corresponding to the agent it belongs to. An internal event (e.g., Login, ReLogin .. Logout) is represented by a dashed arrow and an external event (e.g., LoadEmails, WriteEmail) by a solid arrow. At a lower level, each plan could also be modeled by an activity diagram for further detail if necessary.

Complementarity of the four dimensions

The social dimension gives an overview of the pattern. It answers the question: what does the pattern do? However, this dimension does not give any information about the internal content of each agent in the pattern.

The last three dimensions give more specific information about the design of the implementation of the pattern.

The structural dimension describes “How can each service and agent be implemented?”. Its role (for designing agent-oriented systems) corresponds to the role of the class diagram (for designing object-oriented systems in UML).

The communication dimension describes “How do agents exchange information between them?”. Its role (for designing agent-oriented systems) corresponds to the role of the sequence diagram (for designing object-oriented systems in UML).

The dynamic dimension describes “How is the flow of control between services modeled?”. Its role (for designing agent-oriented systems) corresponds to the role of the state chart / activity diagram (for designing object-oriented systems in UML).

The models, UI objects and SKwyRL framework discussed in this chapter will be used in DB-USE process which is presented in chapter 4 by its architecture, methodology and multi-agent system.

3.4 Chapter Summary

The models, UI objects and SKwyRL framework discussed in this chapter will be used in DB-USE process in chapter 4. In this process, the task, domain and user models are used as its inputs; the CIO and FIO are the objects that DB-USE process has to create based on these models; SKwyRL framework is used to detail the process.

Chapter 4

DB-USE: Generating UI from Task, Domain, User Models Combined Together

The purpose of this chapter is to describe the user interface generation process from the task, user and domain models at the comprehensive and at detail levels.

At the comprehensive level, we discuss the engineering process and its main components. Section 4.1 describes the working process in practice; this process starts with the specification of the requirements; then the models are created; next, the user interface design is determined; and finally, the application is implemented. Section 4.1.2 shows the architecture of DB-USE. We discuss the role of the main components and the resources used in this process to specify the user interface. The DB-USE workflow is described in Section 4.1.3. It shows how the user interface code and the application code are generated from the models. The DB-USE's methodological process is finally discussed in Section 4.1.4.

At the detail level, the user interface generation process is analyzed using the agent framework. The process components modeled as agents are represented with the use of i* social dialog in Section 4.2.1; we discuss about the role of the agents and their relationships. Section 4.2.2 presents the structure of these agents and the plans. Section 4.2.3 describes the dynamic diagram that models the events and the control flow of the plans of all the involved agents.

4.1 Engineering User Interfaces from Combining Task, Domain and User Models

In this section, we discuss the engineering process and its main components for the generation of the user interface from task, user and domain models.

4.1.1 Working Process in Practice

According to the Unified Process adapted for UI design, Figure 4-1 depicts the practical working process one can follow from getting the requirements to implementing the UI application. In this process, one has to analyze the requirements specs, specify the (object or agent) structural and operational models at the conceptual and logical levels, select the layout platform, design the user interface so on and finally perform the UI design and deployment.

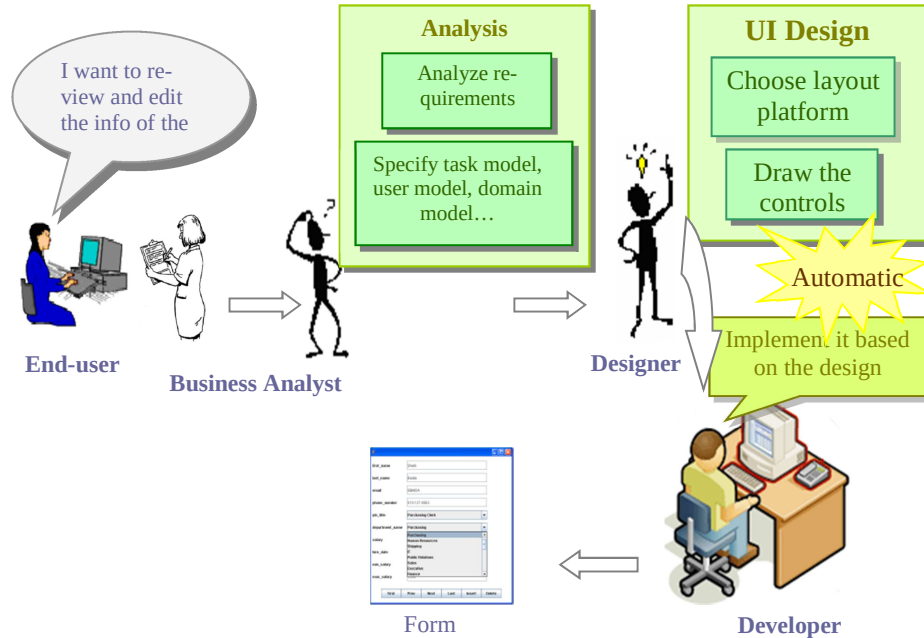


Figure 4-1: The Working Process in Practice

In this process, we have three main steps. The first step is to create the declarative models. The process starts with the production of the requirements which are created by the business analyst. In order to create these requirements the business analyst communicates with the end-users to understand their work, domain, skills, goal so on. Based on these requirements the Business Analyst creates the declarative models such as task, domain, user, application, presentation models. The second step is to design the user interface. A platform is chosen and the user interface design is determined by the designer based on the models created in the first step. The final step is to implement the application. The developer implements the application on the platform determined based on the user interface design.

There are many various researches which have focused on the automatic user interface design such as Trident, Fuse, Teallach. Our desired goal is to automate both the user interface design and the implementation.

4.1.2 DB-USE Architecture

As discussed in Section 1.3, the task, user and domain models are used to generate the user interface for the following reasons:

DB-USE: Generating UI from Task, Domain, User Models Combined Together

- The *task model* describes how the user interface provides information to the user and how the actions that users and system perform can be sequenced. These two aspects are two main purposes of a user interface [Pate01c].
- The *user model* describes the characteristics of the end-users or groups of end-users. The user model is used in order to generate individual user interfaces, to reconfigure the interface to the end-user, to provide adaptive user interfaces, to provide an appropriate level of help, to actively guide the user during interaction [Schl96].
- The *domain model* defines the aspects of the application which can be adapted or which are otherwise required for running the system. Therefore, the domain model is used to specify the control of this user interface – at this level the user interface is specified more in details.

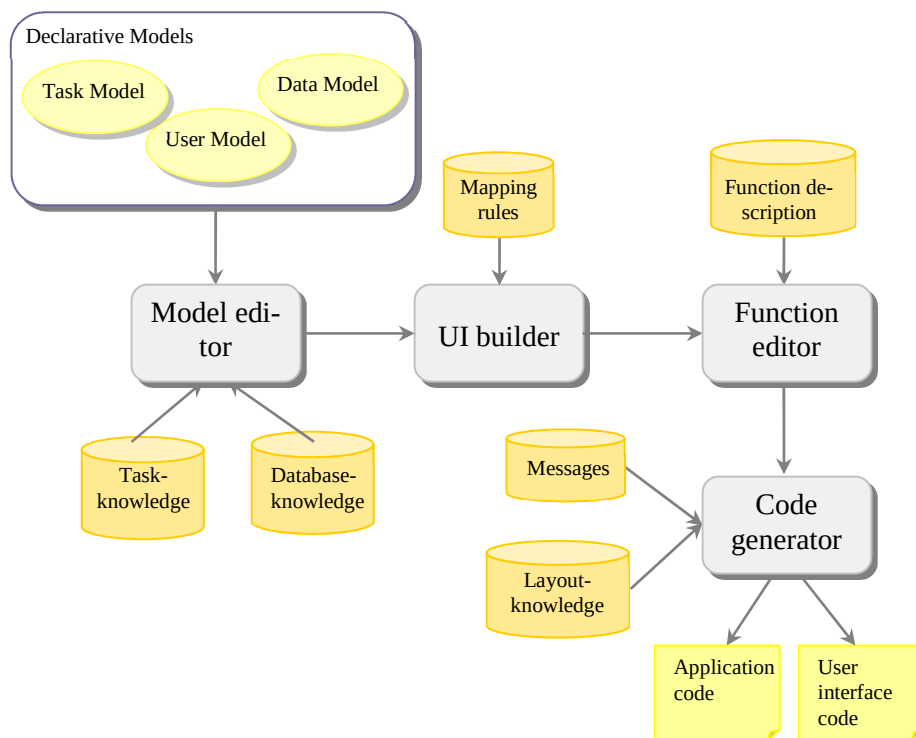


Figure 4-2: Main components of our UI and Code Generation Architecture

Figure 4-2 depicts the main components and the architecture of DB-USE. The **Model editor** agent uses the task- and database-knowledge bases to load the task model from a XML file, the domain model from a concrete database, and the user model from a diagram file. The loaded tasks have already been manually linked to the attributes of the domain objects. From these linked objects, the **UI builder** agent automatically creates the user interface (UI) objects based on the mapping rules

(these rules will be discussed in subsequent sections). The **Function editor** agent uses the Function description base to build the functions of the application which are defined in the domain model. Once the UI objects have been created and the functions structured, the **code generator** agent produces the code. Especially, our process does not only generate the user interface code, but also the application code behind needed to perform these pre-determined functions.

The **Model editor** agent loads the tasks from the task model which is stored in term of XML specifications; the domain model is loaded from the database by executing pre-compiled SQL queries. Executed SQL queries differ when the databases are heterogeneous since the domain model information is stored specifically for each type of database. The user model is also loaded by the **Model editor** agent.

Once the tasks in the task model have been linked to the attributes of the domain's objects in the domain model, Concrete Interaction Objects (CIOs) are created based on the attributes characteristics and the relationships between the domain's objects by the **UI builder** agent. These characteristics are for instance the data types, data length, is-key flag.

Once the CIOs have been created, they are transformed into Final Interaction Objects (FIOs). A FIO is described as a user interface control unit on a concrete platform.

In order to obtain the desired behavior of a database application task, the Function editor agent builds the generic functions of an application by using the function description base. These functions are, for instance, Display(), AddNew(), Update(), Delete(), Search(), Review(), Cancel(), Exit(), and Default() functions. These functions are defined in the domain model.

Finally, the code generator agent uses the Layout-knowledge base to generate the user interface code based on the FIOs and uses the Message base to generate the application code based on the defined functions. The application code is generated to perform the tasks linked to the functions which are defined by the Function editor agent.

In summary, the components of DB-USE are:

- The Task-knowledge base that describes the rules of the task model. These rules are described in the rules RTMs (**R**ules of **T**ask **M**apping) and Table 4-1.
- The Mapping rules base that describes the rules for specifying the concrete user interface from domain's objects and the relationships between these objects and for transforming the concrete user interface into the final user interface. These rules are the rules RSTs (**R**ules for selecting the **S**ub-**T**asks), RLIs (**R**ules for making **L**ink between **I**nformation tasks), RLAs (**R**ules for making **L**inks between **A**ction tasks), RSDs (**R**ules for Specifying **D**ata-type), RGAs (**R**ules for **G**rouping **A**tttributes), RDCs

DB-USE: Generating UI from Task, Domain, User Models Combined Together

(Rules for Determining the Control) and they will be discussed in detail later.

- The Database-knowledge base that describes generic aspects of the database tasks, the syntax and the structure of a query.
- The Layout-knowledge base that contains the syntactic design guidelines for controls, windows and other widgets layouts. It also describes the semantic rules from which the control types are defined.
- The Messages base that contains the generic messages such as errors, warnings, information to users messages, ... See Table 4-5.
- The Function description base that describes the generic functions of a task of the database application.

We will discuss these components more in details later.

4.1.3 DB-USE Workflow

Our process for generating the user interface is depicted in Figure 4-3. The code generation process starts with loading the domain model from the database and the generic functions defined by DB-USE system, the user model from a text file, the task model from a XML file. Besides, DB-USE also defines the functions in the domain model which are used to perform the generic database tasks; these functions are Display(), AddNew(), Update(), Delete(), Search(), Review(), Cancel(), Exit(), and Default() functions

Once the models have been loaded, the developer determines the tasks from which the user interfaces can be generated based on the domain model. Typically, these tasks are database manipulation tasks. The developer then makes the links between the information tasks and attributes of the domain's objects in the domain model and the links between the action tasks and the defined functions. The CIOs are created based on the linked objects; then these CIOs are transformed to FIOs and stored in terms of UsiXML specification. Besides, the DB-USE determines the linked functions to construct the methods for performing these functions. Finally, the user interface code and application code are automatically generated.

In this process, we use the XML file, diagram file and database as resources to load the task, user and domain models. These resources are created by the business analyst/designer:

- The database is used to get the information of the domain model.
- The XML file is used to load task model; the XML file stores the task model which is created by task model case tools such as ConcurTaskTreeEnvironment (CTT) [Pate01a, Pate01b].
- The diagram file is used to get the user's characteristics.

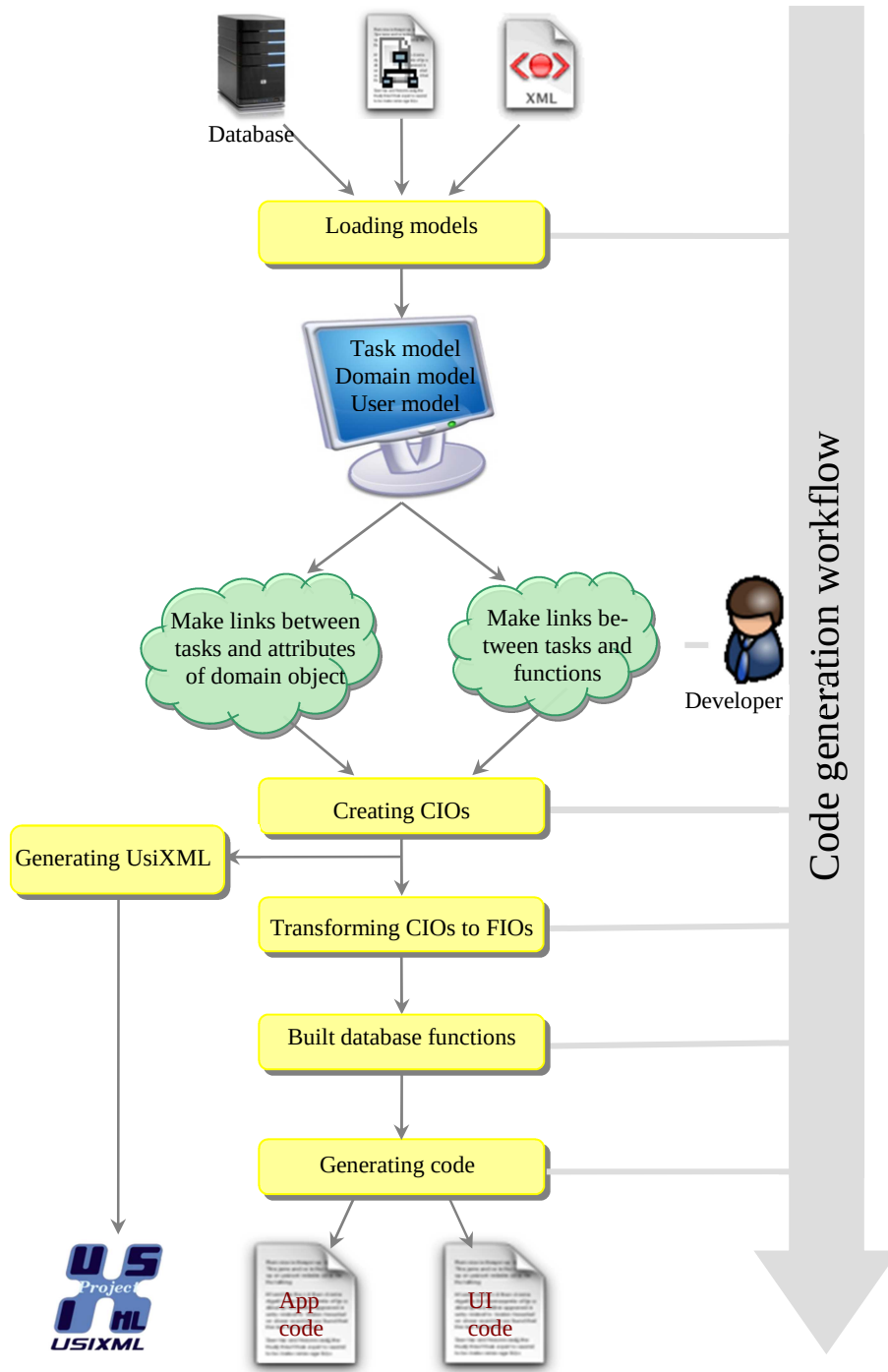


Figure 4-3: Code generation workflow: UIs are generated from task, user and domain models

4.1.4 DB-USE Methodology

DB-USE is specially interesting in the way that it benefits the UI designers. It is built to support them as much as possible in every phases of the UI generation. UI generation process of DB-USE consists of five phases: the model analysis, the relation making, the UI design, the Application function design, and the code generation. There is one person who is responsible for each phase. This methodological process is depicted in Figure 4-4.

The **Model analyst** is responsible for the model analysis phase. The task, domain and user models are automatically built by the DB-USE system from the existing resources. The **User Model Editor** is used to load the user model from a diagram file; the **Task Model Editor** is used to load the task model from a XML file; the domain model is loaded from a concrete database by the **Domain Model Editor**. The generic functions of a database application which are defined by DB-USE are included into its domain model. These three components are implemented independently. Once these models have been loaded, they are verified and modified by the **Model analyst**. He can change the attribute of the tasks such as name, type, right task, action type, temporal operator or delete an existing task or add a new one. Specially, he/she can create a new DB-USE task model with the **Model Editor** if it is not created yet.

The associations between the components of the task and domain models are made by the **UI Designer** in the relation making phase. First, **UI Designer** has to specify a composite action task that he/she wants to be used to generate the UI. Then DB-USE displays its sub-tasks based on RST rules (**R**ules for selecting the **S**ub-**T**asks). DB-USE's task model includes two types of task: information and action tasks. The information tasks are linked to the domain's attributes and the action tasks are linked to the defined functions. The **UI Designer** selects the tasks and the domain's attributes which participate in the associations.

In the UI design phase of the DB-USE process, the CIOs are created by the **UI Builder** based on the associations made in the relation making phase and on the user model. The user model is used to generate individual user interfaces and to select the control type among the possible ones that matches the preference of the end-users. Once CIOs have been created, in order to reuse them in the different running environments (including platform, device and language); DB-USE stores them in terms of UsiXML [Limb09]. Since the programming language has been determined, the FIOs (Final Interaction Object) are created based on the CIOs. In order to specify the attributes of the UI objects, DB-USE uses the **Mapping rules** base which defines the rules for making the associations between the tasks and domain's components, specifying the control type, location and dimension. In this phase, the **UI Designer** plays a crucial role since he/she has to make sure that the created UI objects satisfy the goal of the tasks.

Engineering User Interfaces from Combining Task, Domain and User Models

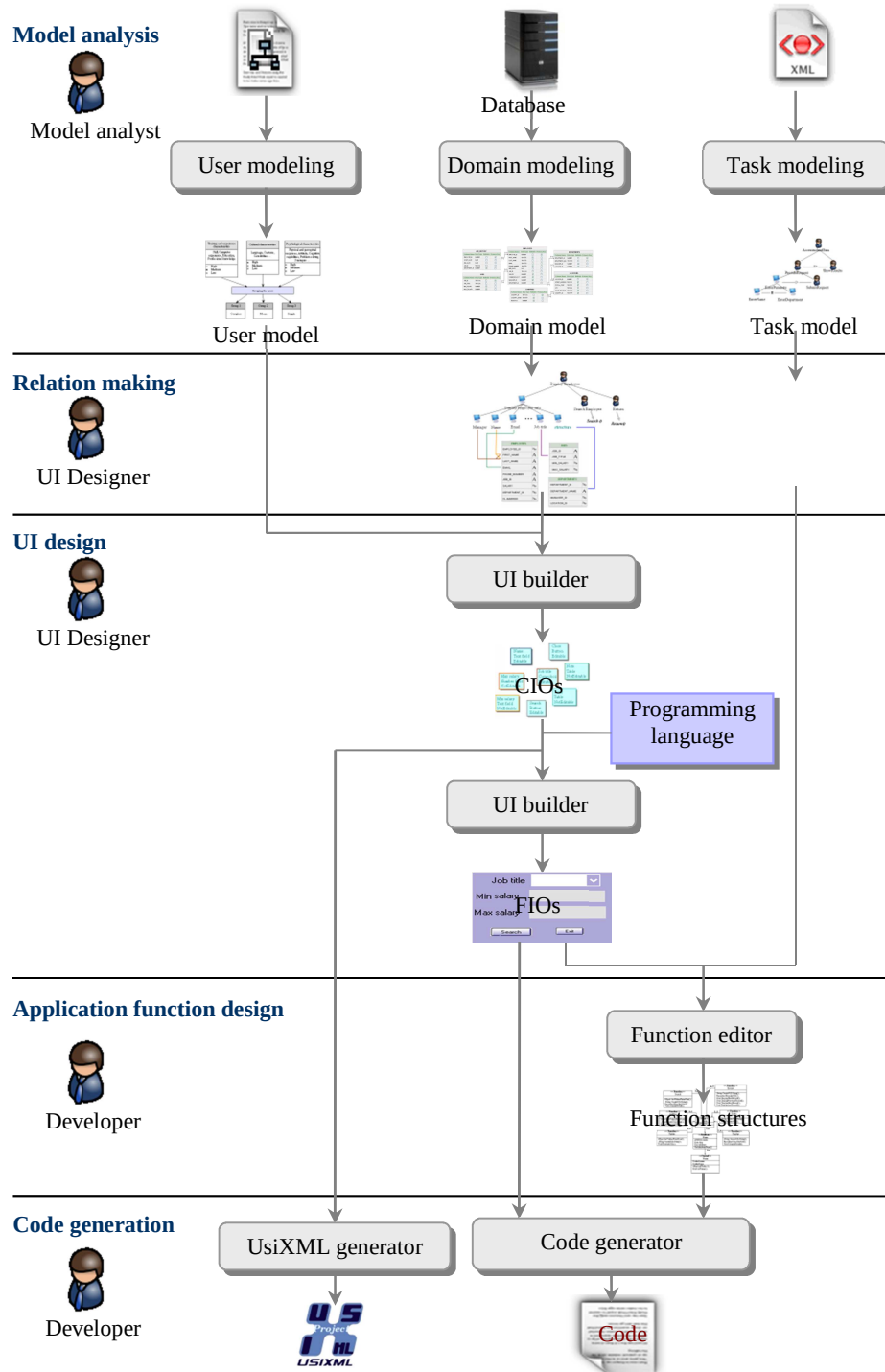


Figure 4-4: DB-USE - Methodology

DB-USE: Generating UI from Task, Domain, User Models Combined Together

The application function design phase is started once the FIOs have been completely created. The **Function** will determine the linked functions of the created FIOs to construct the structure for these functions (Display(), AddNew(), Update(), Delete(), Search(), Review(), ...). One strong advantage of DB-USE is that application functions are automatically constructed by the system not by the **Developer**. But the **Developer** can modify the structure of the function, define a new function or delete an existing one.

Finally, the UI code and the application code are generated in the code generation phase by the **Code generator**. The role of the **Developer** in this phase is to verify the generated code then to modify it if needed in order to ensure that it is successfully compiled. Typically, the code generated by DB-USE is clear and complete enough to be compiled and executed immediately.

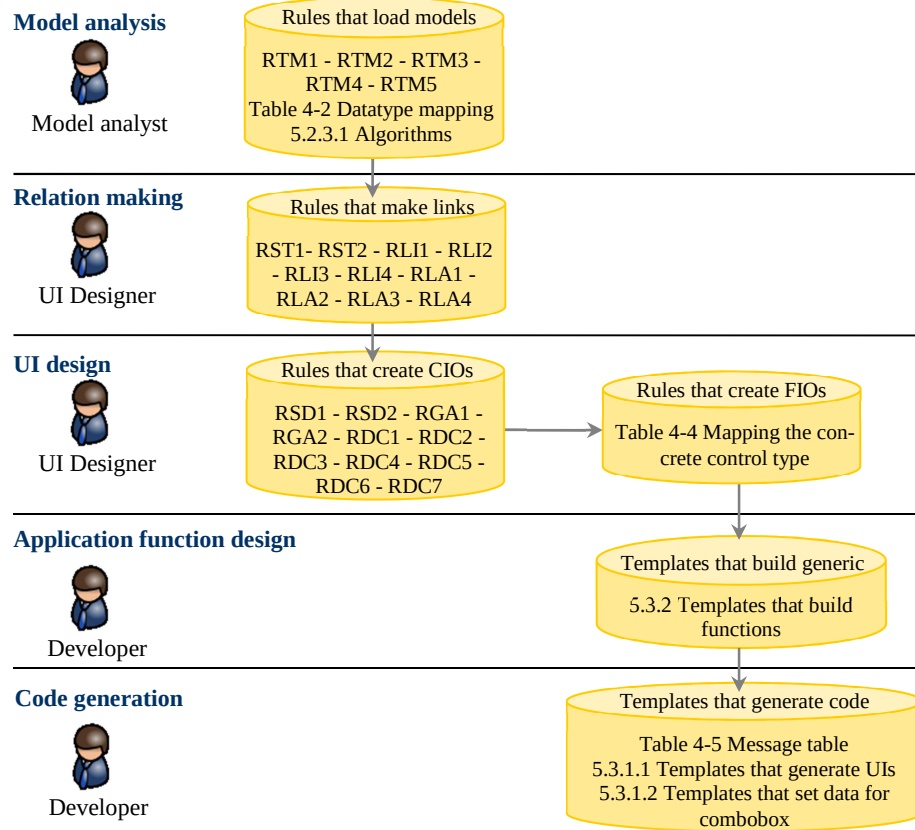


Figure 4-5: Mapping rules, algorithms, templates used in DB-USE methodology

In order to perform the methodology depicted in Figure 4-4, DB-USE provides the mapping rules, algorithms and templates that are used to load DB-USE's models, to create UI objects and to generate the code. These resources will be discussed in detail with the concrete examples in Chapters 4 and 5. Most mapping rules are used in three first phases; and the templates are used in two last ones. Specially, these

resources can be reused by the approaches of model-based user interface development environment (MB-IDE). For example, the mapping rules used to create UI objects and the templates used to generate code are the ones that are used in most processes of MB-IDE; the mapping rules and algorithms used to work with the models can be reused in the approaches if the models of these approaches have the same format with models of DB-USE; and finally, the mapping rules used to make relation between components and the templates used to build functions are the special ones, they are defined only for performing the process of DB-USE.

4.2 DB-USE Process

Our process is organized around the components mentioned in Section 4.1. The different components play different roles; they interact with one another by sending the events and transferring the information. Besides, the components use the resources database, mapping rules base, layout-knowledge base and so on to achieve their goals. The pro-active and adaptable aspects of the components motivate the adoption of the agent paradigm [Do05] to analyze the models and generate the UI and code. Each component is described as an agent and the various components interact with each other to compose a MAS architecture.

4.2.1 DB-USE - Social Model

The process proposed in Section 4.1.2 is detailed using i* in Figure 4-6. In a few words, i* is an organizational modeling framework [Yu95] proposing concepts such as actor (actors can be agents, positions or roles), as well as social dependencies among actors. Each node represents an actor and each link between two actors indicates a dependency (called dependum) between two actors: the depender and the dependee. The depender is the depending actor, and the dependee, the actor who is depended upon. The type of the dependency may be a goal, a softgoal, a task, or a resource. Goal dependencies represent delegation of responsibility for fulfilling a goal; softgoal dependencies are similar to goal dependencies, but their fulfillment cannot be defined precisely; task dependencies are used in situations where the dependee is required to perform a given activity; and resource dependencies require the dependee to provide a resource to the depender. As partially shown in Figure 4-5, actors are represented as circles; dependums (goals, softgoals, tasks and resources) are respectively represented as ovals, clouds, hexagons and rectangles; and dependencies have the form depender → dependum → dependee.

The **Model Editor** agent displays the task model from the XML file (e.g., exported from the CTT - ConcurTaskTrees tool [Pate01a, Pate01b]), the information of the user model from the diagram file and the domain model from the database, to the developer. It determines the tasks from which the UI should be generated; the information tasks are then linked to the attributes of domain's object in domain model by the Developer. On the other hand, the action tasks are linked to the functions which are defined in the domain model. Based on these links, the **UI**

DB-USE: Generating UI from Task, Domain, User Models Combined Together

Builder agent automatically creates CIOs then these CIOs are transformed to FIOs. Once the FIOs have been created, the **Function Editor** agent specifies the structure of the functions which are linked to action tasks based on these FIOs. Finally, the **Code Generator** agent outputs the user interface code and application code based on the analyzed results.

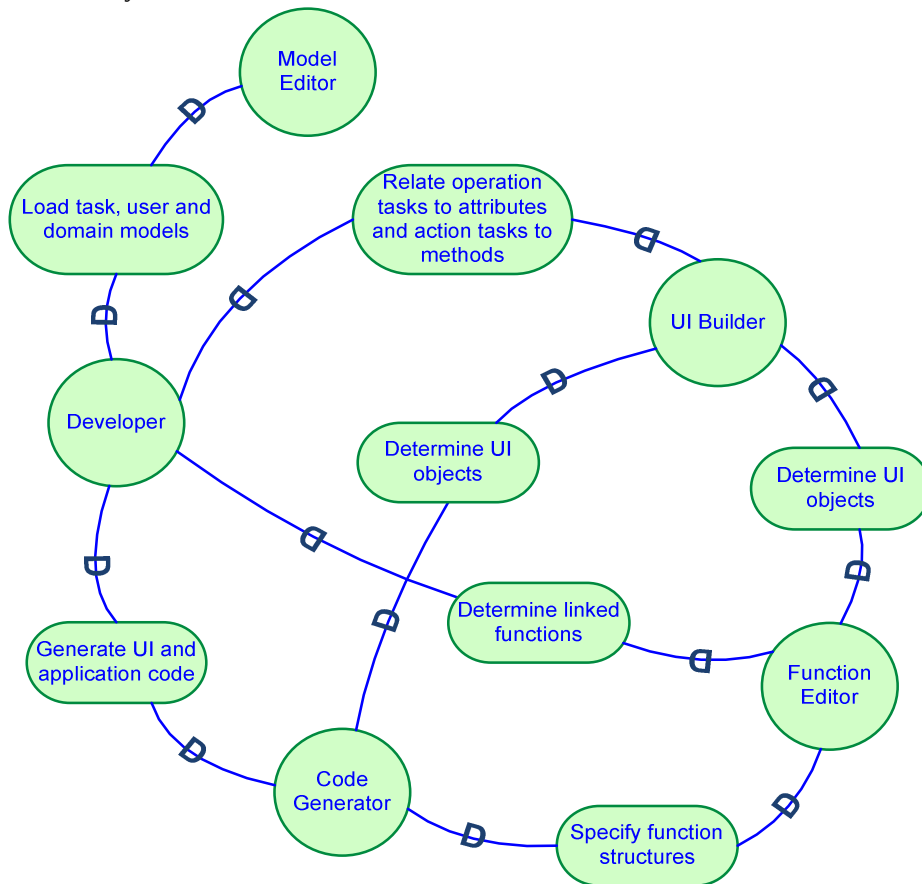


Figure 4-6: i* Social Diagram – Processing code generation from task, domain and user models.

The **Developer** depends on the **Function Editor** agent to define the basic functions of the database application and on the **Model Editor** agent to load the task, user and domain models. The **UI Builder** agent depends on the **Developer** to make the links between the information tasks and the attributes of domain's object and the links between the action tasks and the defined functions. The **Code Generator** agent depends on the **UI Builder** to create the UI objects. Finally, the **Developer** depends on the **Code Generator** agent to generate the user interface code and application code.

4.2.2 DB-USE - Agent Structure

The **Function Editor**, the **Model Editor**, the **Developer**, the **UI Builder** and the **Code Generator** agents are described in detail. An agent is structured by the name, the plans, the belief and the functions. In order to achieve its goal, the agent performs its plans based on its beliefs. An agent is activated by the other agents by sending the events to each other.

4.2.2.1 Model Editor Agent

The **Model Editor** agent uses the *LoadTaskModel* plan to load the task model from the XML file, the LoadUserModel to load the user model from the diagram file and the LoadDomainModel to load the domain model from the database.

The structure of the **Model Editor** agent is illustrated in the Figure 4-7.

<<Agent >> Model Editor	
Attribute	
Plan	LoadTaskModel() LoadUserModel() LoadDomainModel()
Belief	private belief XML file TXT file Database
Method	Private TaskModel LoadTaskModels(XMLfile f) Private UserModel LoadUserModel(TXTfile f) Private DataModel LoadDomainmodel(String DbName, String UserName, String Password)

Figure 4-7: The Model Analyst agent structure.

a LoadTaskModel Plan

The *LoadTaskModel* plan (Figure 4-8) is used by the **Model Editor** agent to display the task model from the XML file; this XML file can be built by tools like CTT [Pate01a, Pate01b]. When the *LoadTaskModel* plan receives a *LoadTaskModel* event, it will read the XML file built by the developer; the tasks are then created based on the information in this XML file.

DB-USE: Generating UI from Task, Domain, User Models Combined Together

<<Plan>> LoadTaskModel
Event <i>post</i> <i>LoadUserModel</i> <i>handle</i> <i>LoadTaskModel</i>
Used Belief XML file
Method <i>main()</i> { // Handle a <i>LoadTaskModel</i> event // Read the information of the tasks from a XML file // Map the Abstraction, Application, Interaction and // User tasks into action and operation tasks // Draw the tasks to the developer // Post <i>LoadUserModel</i> event }

Figure 4-8: The LoadTaskModel plan structure.

The information of task model is stored in a XML file. This XML file is built by the developer by using tools like CTT [Pate01a, Pate01b]. A task is described by the name, the category, the name of the task parent, the description, and so on; this information is stored in the XML file like in the following example.

The task starts at syntax *<Task ...>* and ends at syntax *<SubTask>*. Each task has many different attributes, but only some of them are used such as the task name, the task's category, the type of task, the name of task parent, the name of sibling left task and the name of sibling right task.

We utilize the task model created by ConcurTaskTree for the following reasons:

- Focus on activities: the activities that a user aims to perform are described in details.
- Hierarchical structure: the tasks described in terms of hierarchical structure. The hierarchical structure of this specification has two advantages: it provides a wide range of granularity, allowing large and small task structures to be reused and it enables reusable task structures to be defined at both low and high semantic levels.
- Graphical syntax: it is easier to read and understand.
- Concurrent notation: the relationship between the tasks is defined clearly and richly.
- Objects: once the tasks are identified, they will be represented by the user interface objects and the application domain objects.

- Capacity to extend: has a rich set of formally defined temporal operators (i.e. Lotos operators), probably the most extensive one.
- Tool: is supported by a usable graphical tool (CTT) which facilitates its dissemination and communication among practitioners.

```
<Task Identifier="Review employees" Category="Abstraction Task" Iterative="false" Optional="false" PartOfCooperation="false" Frequency=" ">
  <Name> name </Name>
  <Type> </Type>
  <Description> </Description>
  <Precondition> </Precondition>
  <TemporalOperator name="Enabling"/>
  <TimePerformance>
    <Max> </Max>
    <Min> </Min>
    <Average> </Average>
  </TimePerformance>
  <Parent name="Employees"/>
  <SiblingRight name="Modifying employee"/>
  <Object name="" class="" type="" access_mode="" cardinality="">
    <InputAction Description="null" From="null"/>
    <OutputAction Description="null" To="null"/>
  </Object>
  <Object name="" class="" type="" access_mode="" cardinality="">
    <InputAction Description="null" From="null"/>
    <OutputAction Description="null" To="null"/>
  </Object>
  <Object name="" class="" type="" access_mode="" cardinality="">
    <InputAction Description="null" From="null"/>
    <OutputAction Description="null" To="null"/>
  </Object>
  <Object name="" class="" type="" access_mode="" cardinality="">
    <InputAction Description="null" From="null"/>
    <OutputAction Description="null" To="null"/>
  </Object>
  <Object name="" class="" type="" access_mode="" cardinality="">
    <InputAction Description="null" From="null"/>
    <OutputAction Description="null" To="null"/>
  </Object>
  <SubTask>
```

Example 4-1: An example of task *Review employee* created by CTT.

Our tasks are classified into two types: Action task and Information task (See Section 3.2). The defined types of tasks at the design level are action and information types. While the types of tasks in the CTT, read from the XML file, are abstraction, interaction, cooperation, application and user [Lyna03]; these types are defined at the analysis level. At the analysis level, the defined tasks represent different kinds of information among which some are unnecessary for UI generation. For instance, the user task is a cognitive task that the end-user selects as a strategy to solve a problem or checks the result; typically this type of task is not used to generate the user interface. Therefore, the user task is translated to none and the others are translated to action and information tasks at the design level based on the following **Rules of Task Mapping (RTM)**:

- **RTM1:** Abstraction, cooperation tasks are automatically mapped to action tasks.

DB-USE: Generating UI from Task, Domain, User Models Combined Together

- **RTM2:** Application tasks are automatically mapped to information tasks.
- **RTM3:** We do not consider user tasks since these are tasks from users and they do not communicate with the system.
- **RTM4:** Interaction tasks are automatically classified into the action and information task based on the task type. If the type of a task is *Edit*, *Monitoring* or *Selection* then this task is mapped to information task; otherwise it is mapped to action task.
- **RTM5:** All sub-tasks of a task mapped to an information task from an interaction task are also mapped to information tasks (See, for instance, task *EnterParameter* in Example 4-2).

These rules are summarized in the table 4-1.

Task mapping	
Analysis level	Design level
 Abstraction task	 Action task
 Interaction task (Control type)	 Action task
 Interaction task (Edit\Monitoring\ selection)	 Information task
 Cooperation task	 Action task
 Application task	 Information task
 User task	None

Table 4-1: Task Mapping.

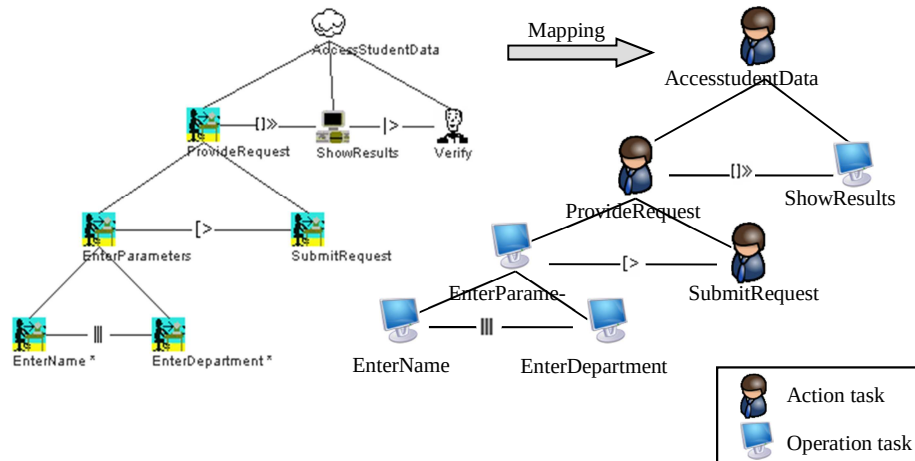
A task in DB-USE is defined by the task ID, the task name, the parent task ID, the task type, the action type, the right task ID, and the temporal operator. See Table 3-4.

The created tasks will be displayed to the developer/end-user in terms of the tree.

Example 4-2 shows a mapping between tasks in ConcurTaskTrees, in our process considering a typical *AccessStudent* Data task. Task *Verify* is not focused; task *AccessStudentData* is automatically mapped to action task; task *ShowResults* is automatically mapped to information task; and other tasks are mapped to action and information tasks based on the type of each task. For example: the task type of *SubmitRequest* is Control type so it is mapped to action task; the task types of

DB-USE Process

EnterName and *EnterDepartment* are Edit and Selection so they are mapped to information tasks.



Example 4-2: Mapping of AccessStudentData task

Beside creation of the DB-USE's task model based on the CTT's task model, **LoadTaskModel** plan also permits the designer/developer to modify it by changing the attribute of the task (such as the task name, the task type, the action type, the right task, and the temporal operator), creating new a task, or deleting an existing one. This is one of strong point of **LoadTaskModel** plan.

After the task model is loaded, the plan then posts a **LoadUserModel** event to the **LoadUserModel** plan.

b LoadUserModel Plan

The **LoadUserModel** plan is depicted in Figure 4-9. The **Model Editor** uses this plan to load the user model from a diagram file; this file is created by the developer based on the characteristics of the end-users.

The information in the **user model** is analyzed to classify the users into three different classes based on their ability to use the software. The analyzed information is the characteristics of the users such as the experience, skill, knowledge, behavior and so on. The three classes of the user model are named "Expert", "Intermediate" and "Novice" corresponding to three levels of ability to use the software. Based on these user classes, the designer will choose to build a complex, medium or simple user interface.

<<Plan>> LoadUserModel
Event <i>post</i> LoadDomainModel <i>handle</i> LoadUserModel
Used Belief <i>use</i> <i>Txt file</i>
Method <i>main()</i> { // Handle a <i>LoadUserModel</i> event // Read the information from a txt file // Analyze this information to determine the types // of end-users // Post <i>LoadDomainModel</i> event }

Figure 4-9: The LoadUserModel plan structure

The *LoadUserModel* finishes its task by posting the *LoadDomainModel* event.

c LoadDomainModel Plan

The ***LoadDomainModel*** plan (Figure 4-10) is used to load a domain model from a database specified by the developer. Once the ***LoadDomainModel*** plan received a ***LoadDomainModel*** event, a database connection is opened based on the type of database (Oracle, SQL server, MySQL ...), database name, user name/schema, password entered by the developer. The domain model is loaded by executing the SQL queries to get the domain's objects, the attributes of domain's object, the characteristics of these attributes and relationships between domain's objects. Besides, ***LoadDomainModel*** plan also defines the generic database functions into the domain model.

The Domain model in DB-USE is represented by five objects which are domain's object, attribute, attribute's characteristics, relationship between the domain's objects, and the generic function of a database application. These objects are built based on the information obtained from the database.

In order to create an opened development environment for the domain model, ***LoadDomainModel*** plan permits the designer/developer to add a new function into, or to delete an existing one from the domain model. For example, if deleting data is not necessary then function Delete() can be deleted; if making reports is needed then the designer/developer can add the function Report() into the domain model.

Based on the different databases, the different SQL queries are executed to get the information of domain model. This information consists of the name of the

domain's object, the name of the domain attribute, the attribute's characteristics and the relationships; which are stored in the database data dictionary.

<<Plan>> LoadDomainModel
Event <i>sent</i> MakeLinks <i>handle</i> LoadDomainModel
Used Belief <i>use</i> Database
Method <i>main()</i> { // Handle a <i>LoadDomainModel</i> event // Connect to the database // Get the needed information to create the // domain's objects, their attributes, relationships // Define the generic database functions // Send <i>MakeLinks</i> event }

Figure 4-10: The LoadDomainModel plan structure

For example, in Oracle, the name of the domain's object is stored in the *User_tables* view; the attribute's characteristics are stored in the *All_tab_cols* view; the relationships are stored in the *All_constraints* view. All these are view of the Oracle Database data dictionary stored in the system user/schema. In SQL server, the name of the domain's object is stored in the *information_schema.tables*; the attribute's characteristics are stored in the *information_schema.columns*; the relationships are stored in the *information_schema.constraints*. They are here meta system views owned by *dbo*, the database owner.

Concretely, the steps performed to build domain's object, attribute of domain's object and relationship are as follows:

- Obtain all of the names of the domain's objects.
- Obtain all of the names of the attributes and the characteristics of the attributes.
- Obtain the relationships of domain's objects.

The following queries are executed in Oracle to obtain the information of the domain model.

Get the name of the domain's objects:

DB-USE: Generating UI from Task, Domain, User Models Combined Together

```
Select table_name
From User_tables
```

Get the characteristics of the attributes of object A:

```
Select column_name, data_type, data_length, data_precision, data_scale, nullable
From all_tab_cols
Where table_name = 'A'
```

Get the relationships of object A:

```
Select constraint_name, constraint_type, column_name, r_constraint_name
From all_cons_columns left join all_constraints on
    all_cons_columns.constraint_name = all_constraints.constraint_name
Where all_cons_columns.table_name = 'A'
```

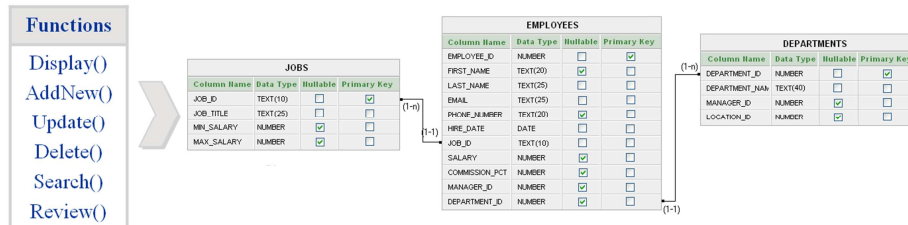
The characteristic data type in different databases is named by the different names. For example text type in SQL server is varchar or varchar2; in oracle is text, char, varchar and in My SQL is tinytext, text, mediumtext. So that we suggest a datatype mapping table for mapping the data type between the different databases as to determine the data types of the attribute from the different databases (Table 4-2).

The datatype mapping table is built for Oracle, SQL server, My SQL databases.

Data type	Oracle	SQL server	My SQL
boolean	None	Bit	bit
date	date	datetime timestamp	date
text	char	text	tinytext
	nchar	nchar	text
	varchar	char	mediumtext
	varchar2	nchar	longtext
	nvarchar2	varchar	binary
number	number long	nvarchar	varbinary
		int	
		smallint	
		tinyint	int
		decimal	float
		numeric	double
		float	decimal
		real	
datetime	timestamp	datetime timestamp	timestamp

Table 4-2: Datatype mapping table.

Example 4-3 shows a domain model. This model consists of objects Employees, Jobs and Departments; the attributes ID, First_Name, Last_Name, Job_Title, Salary, and so on; the characteristic data type, nullable, primary key; the relationships between the Employees and others are 1-n, n-1 and 1-1; and the generic functions.



Example 4-3: Domain model

The **LoadDomainModel** plan sends the **MakeLinks** event to the **Developer** agent when the domain model is completely loaded.

4.2.2.2 Developer Agent

The **Developer** agent (Figure 4-11) uses the **DetermineATask** plan to determine a task, which is used to generate the user interface based on the domain model. The **MakeLinkForInformationTask** plan is used to make links between information tasks and the attributes of domain's objects. Finally, **MakeLinkForActionTask** makes the links between the action tasks and the defined functions.

The structure of the **Developer** agent is illustrated in Figure 4-10.

<<Agent >> Developer	
Attribute	
Plan	DetermineGeneratedTask () MakeLinkForOperationTask() MakeLinkForActionTask ()
Belief	Task model Domain model
Method	Private void DetermineGeneratedTask() Private void MakeLinks()

Figure 4-11: Developer agent structure

The main goal of this agent is to make links between the components in the task and the domain models. In our process these components are tasks, attributes of the domain's objects and functions defined in the domain model.

a DetermineATask Plan

The **DetermineATask** plan (Figure 4-12) is used by the **Developer** agent to handle the **MakeLinks** event and to post the **LinkITask**, **LinkATask** events. Besides,

DB-USE: Generating UI from Task, Domain, User Models Combined Together

the **ChooseATask** plan is also used to choose a task from the task model. For each chosen task, a dialog is generated.

The task model has many tasks and not all of them are used to generate user interfaces. In order to simplify the making of the links between these tasks and the attributes of the domain's object, the defined functions, it should focus on one task at a time. The focused tasks usually are the ones needed to access the database in order to select data, delete data, update data ... Therefore, the developers have to determine which ones can be used to generate the user interface based on the domain model. In practice, the chosen task is the composite action task.

A task is selected which enables the user to perform operations on the database. The operations may be performed at collection level (create new objects, delete or associate existing objects) or object level (modify the object attributes).

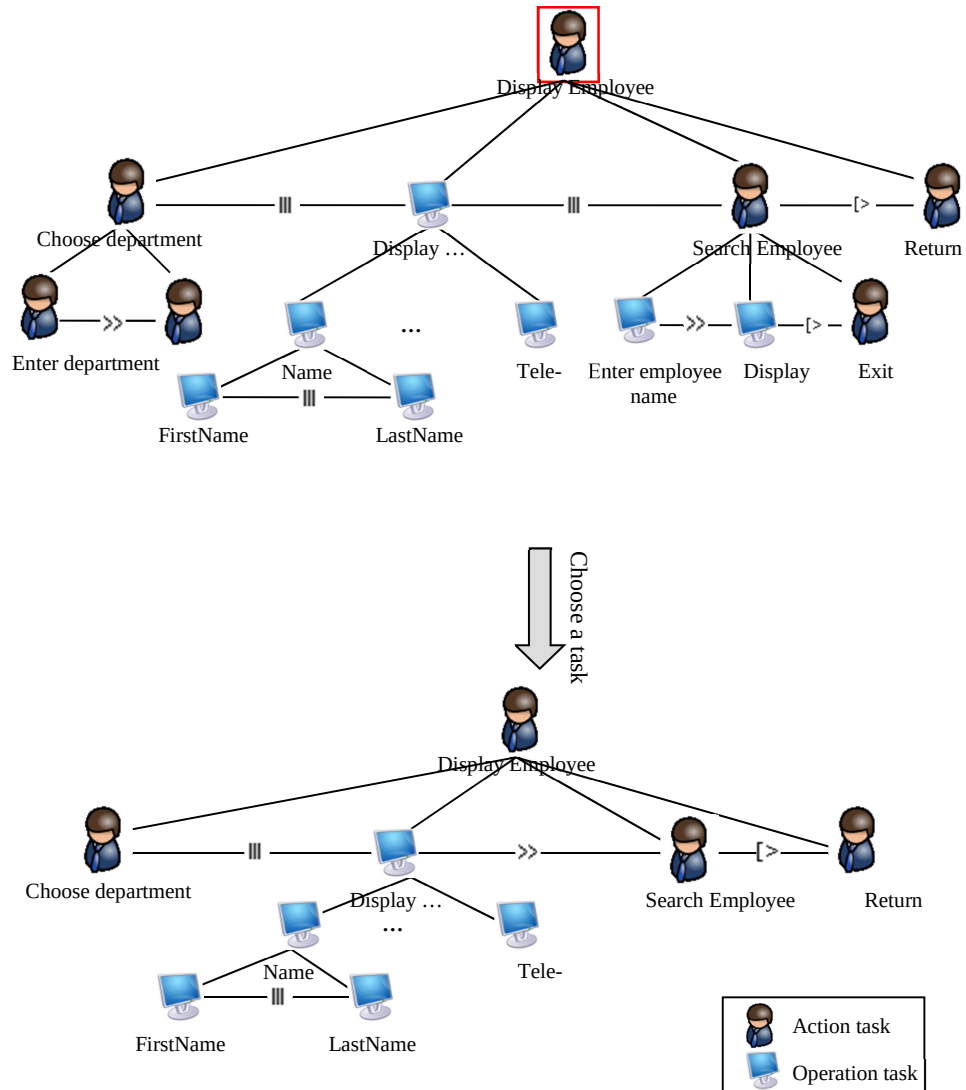
When a task is determined for generating the user interface, its sub-tasks are also selected based on the following RST rules; these sub-tasks are classified into information and action tasks by DB-USE as discussed for the **LoadTaskModel** plan.

<<Plan>> DetermineATask
Event <i>post</i> LinkITask LinkATask <i>handle</i> MakeLink
Used Belief <i>Task model</i>
Method <i>main()</i> { // Handle a <i>MakeLink</i> event // Determine a task // load the sub-tasks of the determined task // Post <i>LinkITask</i> , <i>LinkATask</i> events }

Figure 4-12: DetermineATask plan structure

The **Rules** for selecting the **Sub-Tasks** of a chosen task (RST) are:

- **RST1:** All sub-tasks at level 1 are selected.
- **RST2:** All sub-tasks of a sub-task of task type information are selected.



Example 4-4: Display a chosen task for generating the UI

Example 4-4 depicts how to display a chosen task. The first diagram shows tasks in our system; the second diagram shows a task chosen by the developer for generating the user interface. In this example, the task *Display employee* is chosen for generating the user interface. Sub-tasks at level 1 are tasks *Choose department*, *Display employee info*, *Search employee* and *Return*; the sub-tasks discussed in rule RST2 are *Display employee info* and *Name*.

b MakeLinkForInformationTask Plan

Once the **MakeLinkForInformationTask** plan received the *LinkITask* event, it makes the links between the information tasks and the attributes of domain's objects (See Figure 4-13). Linking a leaf task to the attributes of domain's objects specifies that the task will be an invocation of this attribute on a particular instance of the data type, the length, the primary key and so on that define this attribute. The attributes are selected according to task goals. In this respect, tasks are filtering domain model elements which are further needed for user interface design. In order to make this link, the developer must understand the means of the leaf tasks and then link the leaf tasks to the attributes of domain's object base on the following RLI rules.

<<Plan>> MakeLinkForOperationTask
Event <i>send</i> CreateUIO <i>handle</i> LinkITask
Used Belief Task model Domain model
Method <i>main()</i> { // Handle a <i>LinkITask</i> event // Make link between operation tasks // and attributes of domain's objects // Post <i>CreateUIO</i> event }

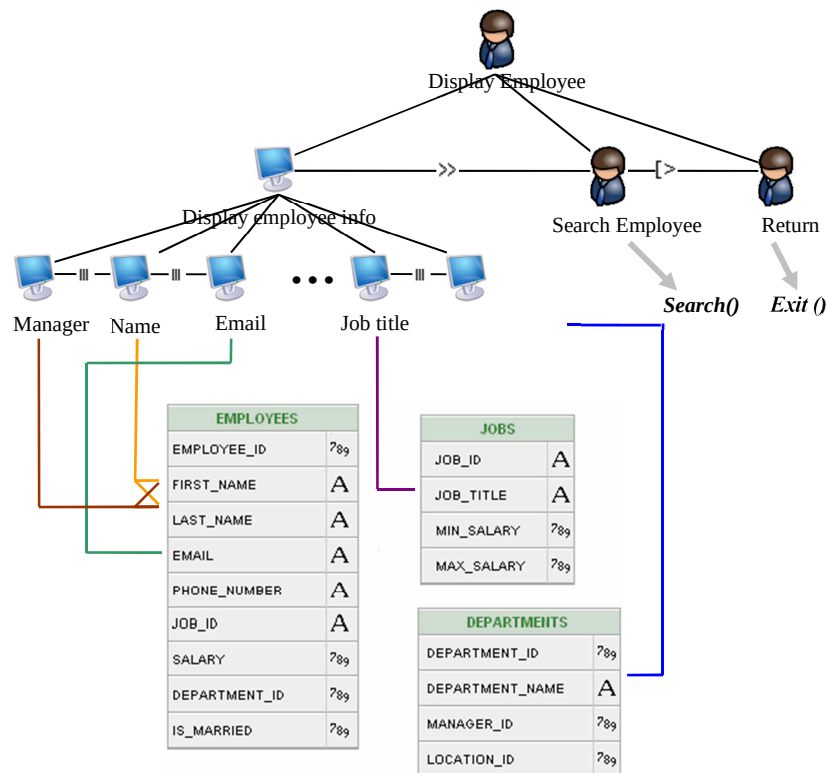
Figure 4-13: MakeLinkForOperationTask plan structure

The **Rules** for making **Link** between **Information** tasks and domain's attributes (RLI) are given below:

- **RLI1:** Information tasks that have at least one sub-task are not linked to the attribute of the domain objects in the domain model.
- **RLI2:** Each leaf information task (a leaf information task is a task which has no sub-task) is linked to at least one attribute of the domain object. This means that all leaf information tasks are used to generate the user interface.
- **RLI3:** One information task can be linked to more than one attribute of the domain object. For example, task Name is linked to attributes First Name and Last Name.

- **RLI4:** One attribute of the domain object can be linked to more than one information task. For example, attributes *First_Name* and *Last_Names* of object *Employees* are linked to tasks *Name* and *Manager*.

Example 4-5 depicts how to link information tasks to attributes of domain's objects. Based on RLI1, task *Display employee info* is linked to nothing. Based on RLI2, all information leaf tasks are linked to attributes of domain's objects. Based on RLI3, the task *Name* is linked to attributes *First Name* and *Last Name*. Based on RLI4, the attributes *First Name* and *Last Name* are linked to tasks *Manager* and *Name*.



Example 4-5: Linking tasks and attributes of domain's objects

The **MakeLinkForInformationTask** plan finishes its task by sending a *CreateUIO* event.

c MakeLinkForActionTask Plan

The **MakeLinkForActionTask** plan will make the links between the action tasks and the defined functions when it receives the *LinkATask* event. Besides, it is also used to send the *CreateUIO* event. The **MakeLinkForActionTask** plan is structured in the Figure 4-14. The functions are represented in Table 3-9. They are *Display()*,

DB-USE: Generating UI from Task, Domain, User Models Combined Together

AddNew(), Update(), Delete(), Search(), Review(), Cancel(), Exit(), and Default() functions.

The action type of the task is used as an advice for linking this task to a correct function: In order to make sure that the links are correctly created, the designer can use the action type as a counsel. These action types are defined in Table 3-5; they are **I** (Indeterminacy), **C** (Create), **D** (Delete), **U** (Update), **S** (Search), and **R** (Read). For example if the action type of a task is **C** then this task should be linked to *AddNew()* function; or if the action type of a task is **D** then this task should be linked to *Delete()* function.

<<Plan>> MakeLinkForActionTask
Event <i>send</i> CreateUIO <i>handle</i> LinkATask
Used Belief Task model Defined functions
Method <i>main()</i> { // Handle a <i>LinkATask</i> event // Make link between action tasks // and defined methods // send <i>CreateUIO</i> event }

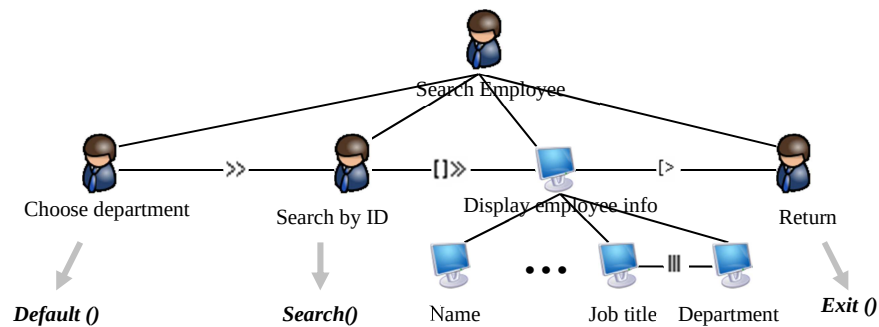
Figure 4-14: MakeLinkForActionTask plan

Linking the action tasks to defined functions is made by the developer based on the following RLA rules.

The **Rules for making Links between Action tasks and defined functions (RLA)** are:

- **RLA1:** At first, each action task is linked to *Default()* function which is defined by default and the name of the function generated by the system is the name of the task. For example, the Task *Delete employee* is linked to the function *DeleteEmployee()*.
- **RLA2:** Each action task is linked to only one defined function.
- **RLA3:** Each defined function is linked to only one action task.
- **RLA4:** The original action task may be linked to a function which is not based on the task goal. If this task starts by performing a function then the

original action task is linked to this function. For example, when the task *display employee information* starts, a *display()* function is performed so that its original goal is linked to function *display()*.



Example 4-6: Linking tasks to the functions defined in domain model.

Example 4-6 shows how to link action tasks to the defined functions. The *Choose department* task is linked to *Default()* function because the existing defined functions do not support the goal of this task.

Like the *MakeLinkForInformationTask* plan, the *MakeLinkForActionTask* plan also finishes its task by sending a *CreateCIO* event.

4.2.2.3 UI Builder Agent

Since the links are made by the **Developer**, the **UI Builder** agent uses the *CreateCIO* and *TransformCIOstoFIOs* plans to analyze the linked components to create the UI objects. These components are the tasks, the attributes, the domain's objects and the defined functions. The structure of the **UI Builder** agent is illustrated in Figure 4-15. The Database-knowledge base (See Section 4.1.1) is used to understand the relationships between the tasks and the domain's objects.

There are two important UI objects used in our process. These objects are Concrete Interaction Object and Final Interface Object.

- A Concrete Interaction Object (CIO) is a graphical object for entering and displaying the data that the user can see, feel and manipulate [Vand93]. A CIO is structured by the label, the control type and the editable (See section 3.2.1).
- The Final Interface Object (FIO) represents the operational interface object that is running on a special computing platform either by interpretation (e.g., through a web browser) or by execution. The FIO is determined based on the CIO in a certain language, on a certain platform and so on. A FIO is structured by the label, the control type, the editable, the location and the dimension (See section 3.2.2).

DB-USE: Generating UI from Task, Domain, User Models Combined Together

<<Agent>> UI Builder
Attribute
Plan CreateCIO() TransformCIOstoFIOs()
Belief <i>private belief</i> Database-knowledge base
Method Private Object CreateCIOs() Private void TransformCIOstoFIOs (Object CIOs)

Figure 4-15: UI Creator agent structure

In order to determine the attributes of a CIO such as control type we need to specify the domain object (called *edited object*) on which the data can be changed and to specify the main attribute of the domain object (called *main attribute*).

- An ***Edited object*** is an object that its data can be changed by the end-user. This object is determined by the developer.
- A ***Main attribute*** of a domain's object which relates to an *Edited object* through a 1-1 or n-1 relationship is an attribute determined by the developer. The *Main attribute* is used to determine the control type in the next step.

If the designer does not determine the main attribute, it will be automatically determined by DB-USE system. There are two ways to determine the main attribute. The first is that the main attribute of an object is the first linked attribute of this object. The second is that it is determined based on the action type. For example: if there are many attributes of object O which are linked to the tasks then the main attribute of this object is the attribute linked to the task of U action type (Update, Modify, Change, Alter, Edit).

a CreateCIO Plan

Once the ***CreateCIO*** plan (Figure 4-16) received the *CreateCIO* event, it creates the CIOs based on the links between the tasks and the attribute of domain. Like in other approaches, the user interface objects are specified based on the data type, data length, is-key attributes of the domain's object [Boda95, Silv00]. If one task is linked to more than one attribute then these linked attributes are grouped and they are understood as one attribute. This means that if one task is linked to many attributes then there is a new attribute created to represent these linked attributes.

<<Plan>> CreateCIO
Event <i>handle</i> CreateUIO post TransformCIOtoFIO
Used Belief use Database-knowledge
Method main() { // Handle a <i>CreateCIO</i> event // Specify the components linked // Create CIO // Post <i>TransformCIOtoFIO</i> event }

Figure 4-16: CreateCIO plan structure

Creating a new attribute if one task is linked to more than one object:

The data type of the new attribute (which is created once there is one task linked to more than one attribute) is specified based on the following RSD rules.

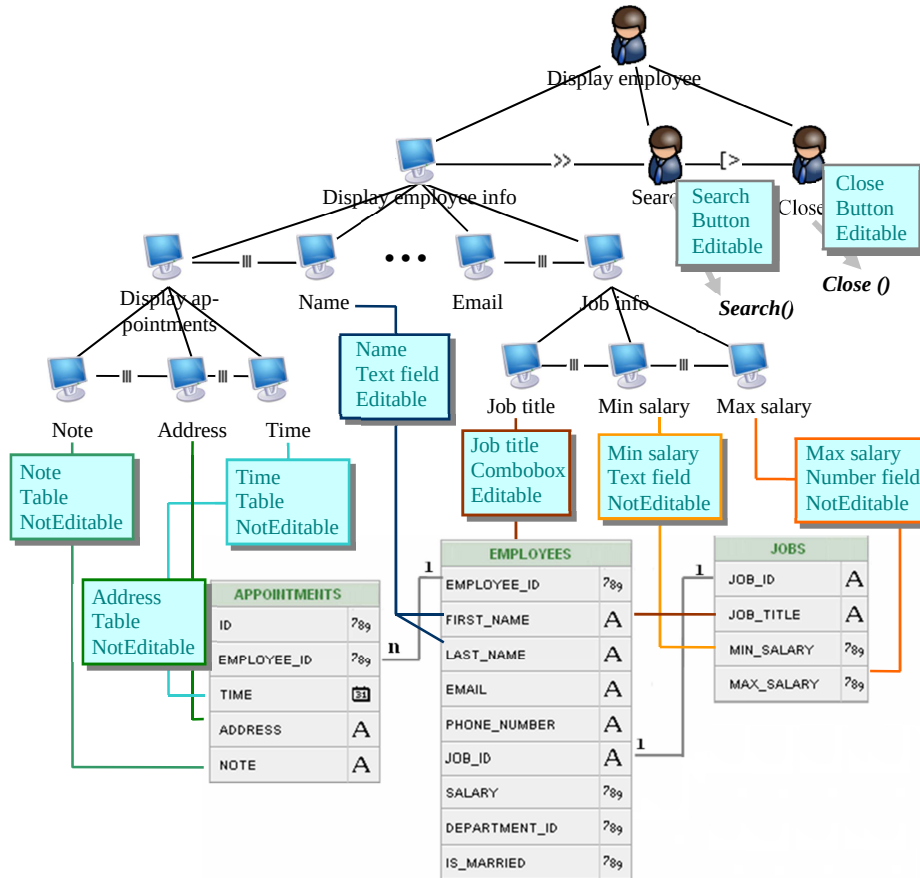
The Rules for Specifying Data-type (RSD) are:

- **RSD1:** If a task is linked to more than one attributes and if the data type of these attributes is T then its data type is T.
- **RSD2:** If a task is linked to more than one attributes and if the data types of these attributes are different (types text, number, date ...) then its data type is text.

For example, if a task **Total** is linked to attributes *Price* and *Amount* and the data type of both attributes is number, the data type of the new attribute *Total* is number. But if a task **Make Appointment** is linked to columns *Date* (datetime type) and *Address* (text type) then the data type of new attribute *Make Appointment* is text.

If the new attribute is created from the attributes of different domain's objects (for example objects A, B, C) then it is necessary to specify the domain's object of this new attribute. The domain's object of the new attribute is one of the above mentioned objects (the mentioned objects are A, B, C) and the one that is the most important and carries most information. For instance, IF the task **Total** is derived from the attribute *Price* in the object *Products* and the attribute *Amount* of the object *Bills*, the relationship between tables *Products-Bills* is 1-n and the edited table is *Products* THEN the new attribute *Total* belongs to the object *Bills*, since in this case, one product has many bills so that if the *Price-Amount* belongs to group *Bills* it carries most information. In order to specify the domain's object of the new attribute we propose some following RGA rules.

DB-USE: Generating UI from Task, Domain, User Models Combined Together



Example 4-7: Creating CUIs process.

The Rules for Grouping Attributes (RGA) are:

- **RGA1:** If all the attributes linked to a task *T* come from only one object *A* then the new attribute created from these attributes belongs to object *A*. For example, if the task *Name* is created from attributes *First_Name* and *Last_Name* of the object *Employees*, the new attribute *Name* belongs to object *Employees*.
- **RGA2:** If a task *T* derives from two objects *A*, *B* and the relationship between *A* and *B* is 1-1 or n-1 then the attribute *T* belongs to *A*, otherwise the relationship between *A* and *B* is 1-n or n-n then the attribute *T* belongs to *B*.

Example 4-7 depicts the creating process of CUIs based on a task, the components of domain model and the links between them. Each  represents a CIO; the attributes are represented in order: CIO's name, data type and editable. These attributes are determined based on the RGA and RDC rules. In this case, domain's object *Employees* is edited object and the attributes *Job_Title*, *ID* are the main attributes.

For each leaf task, a CIO is created. Each CIO is created based on the name of the task, the characteristics of the domain attributes which are linked to this task and the relationships between the domain objects. These characteristics are the data type, length, is-key flag and so on. The label of the CIO is the name of the linked task; the control type and editable attributes of the CIO are determined based on the RDC mapping rules presented below.

The **Rules for Determining the Control type** are:

- **RDC1:** When a task derives from the attributes of the edited object then the control type of the CIO created for this task is Text field or Text box if the data type is Text; Number field if the data type is Number; Check box/Radio if the data type is Boolean; Date picker if the data type is Date and the Editable attribute value of this CIO is Yes. For example, in Example 4-7 the CIOs Name, Email, Telephone ... derive from the object Employees.
- **RDC2:** When a task derives from an attribute of a domain object which is not the edited object and if this attribute is not the main attribute and the relationships between the edited object and another one is '1-1' or 'n-1' then the control type of the CIO is Text field or Text box if the data type is Text; Number field if the data type is Number; Check box/Radio if the data is Boolean; Date picker if the data type is Date and the Editable value of this CIO is No. For instance, in Example 4-7 the CIOs Min salary and Max salary derive from the object Jobs.
- **RDC3:** When a task derives from an attribute of the domain object that is not the edited object and if this attribute is the main attribute and the relationships between the edited object and another one is '1-1' or 'n-1' then the control type of the CIO is Combo box and the Editable attribute value of this CIO is Yes. For instance, in Example 4-7, the CIO Job title derives from the object Jobs.
- **RDC4:** If a task derives from the attribute of a domain object which is not the edited object and the relationships between the edited object and another one is '1-n' or 'n-n' then the control type of the CIO is table and the Editable value of this CIO is No. For instance, in Example 4-7 the CIOs Note, Address and Time derive from object Appointments.
- **RDC5:** If the CIOs of the table type belong to the same CIO class then they are assigned to the same table control. Each CIO of type table in our process is a separate interface, but in practice it is just a column of a table. For example, the control type of the CIOs Note, Address and Time is table and they are separate CIOs but belonging to the same group Appointments; they are then assigned to the object Appointments.
- **RDC6:** If the data type is container then the control type of CIO of this task is Tab or Panel control.
- **RDC7:** If a task is linked to a defined function then the control type of CIO of this task is Button, Menu or Pop-up Menu.

DB-USE: Generating UI from Task, Domain, User Models Combined Together

These rules are summarized in Table 4-3:

Data type of attribute of domain's object	CIO created derives from following components					(CIO) Control type	Ed-itable
	Edited object	Related object	Relationship between edited object - related object		Main attribute		
			1-1; n-1	1-n; n-n			
Text	☒	☐				Text field	☒
Text(length>500)	☒	☐				Text box	☒
Text	☒	☒	☒	☐	☐	Text field	☐
Text(length>500)		☒	☒	☐	☐	Text box	☐
Text		☒	☒	☐	☒	Combo box	☒
Text		☒		☒		Table	☐
Number	☒	☐	☐		☐	Number field	☒
Number		☒	☒	☐	☐	Number field	☐
Number		☒	☒	☐	☒	Combo box	☒
Number		☒		☒		Table	☐
Date\DateTime	☒	☐	☐		☐	Date picker	☒
Date\DateTime		☒	☒	☐	☐	Date picker	☐
Date\DateTime		☒	☒	☐	☒	Combo box	☒
Date\DateTime		☒		☒		Table	☐
Boolean	☒	☐				Check box Radio button	☒
Boolean		☒	☒		☐	Check box Radio button	☐
Boolean		☒	☒		☒	Check box Radio button	☒
Boolean		☒		☒		Table	☐
Container						Tab	☒
Container						panel	☐
Void						Button	☒
Void						Menu	☒
Void						Pop-up menu	☒

Table 4-3: Mapping the data type of the domain object to the control type of the CIO

Table 4-4 depicts classification of UI controls based on the ability of user in computer applications. In some cases, it is better if this table is used with the other conditions such as number of control in a concrete space; amount of data; if there are a lot of data and we need to choose a control type among radio button, check-box, combo-box and list-box, then it is better if a combo-box or list-box is chosen; the number of linked functions: if there are many linked functions, it is better to create CIOs of *Menu type* for these functions; because each button takes place on the screen so that too many of them cannot fit in.

Select a UI control from the possible ones				
User type	Execute a command	Make a choice	Make a choice	Make multi choices
	Novice			
	Intermediate			
	Expert			

Table 4-4: Classify UI controls based on ability of user in computer applications

DB-USE: Generating UI from Task, Domain, User Models Combined Together

Besides the use of the rules above to determine the control type of the CIOs; the User model is also used to determine the control type which matches the preference of the end-users. Based on the information from the user model, DB-USE selects a correct control type among the possible ones. In other words, if there is more than one control type determined for one CIO then the software chooses one of them for this CIO based on the information in the user model. For example the control type of the CIO Exit can be Button, Menu or pop-up menu types. If the users have no experience then the control type will be Button; if the users are professional then the control type will be Menu. Another example: the control type is specified based on the user characteristics: they are defined based on the user's work experience, educational level, age, previous computer experience and so on. It is possible to anticipate their learning difficulties to some extent and to better set appropriate limits for the user interface complexity. For instance: we have two ways to open a student's name dialog; one is to click on students menu then choose the student's name menu, one is to click on a student's name button. If the user does not have computer experience it is better if the UI is a button. He/She can understand exactly, quickly what he/she needs to do to open a dialog to see the student's name.

In order to reuse the CIOs in the different running environments, these CIOs are stored in a XML file by using the UsiXML. DB-USE uses UsiXML because this language is specifically defined for describing the user interface objects.

```
<outputText id="id" name="Employee ID lable" content="xpath" defaultContent="Employee ID" isVisible="true"
isBold="false" textColor="#000000"/>
<inputText id="id0" name="Employee ID control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id1" name="First name lable" content="xpath" defaultContent="First name" isVisible="true"
isBold="false" textColor="#000000"/>
<inputText id="id2" name="First name control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id3" name="Last name lable" content="xpath" defaultContent="Last name" isVisible="true"
isBold="false" textColor="#000000"/>
<inputText id="id4" name="Last name control" maxLength="50" numberOfColumns="1" isEditable="true"/>
<outputText id="id5" name="Job title lable" content="xpath" defaultContent="Job title" isVisible="true"
isBold="false" textColor="#000000"/>
<comboBox id="id6" name="Job title control" isEnabled="true" currentValue="" />
<outputText id="id7" name="Department name lable" content="xpath" defaultContent="Department name"
isVisible="true" isBold="false" textColor="#000000"/>
<comboBox id="id8" name="Department name control" isEnabled="true" currentValue="" />
<button id="id9" name="Add a new employee" content="xpath" defaultContent="New"/>
```

Example 4-8: CIOs are described by using UsiXML .

Example 4-8 depicts the text boxes and the combo boxes by using UsiXML.

b TransformCIOstoFIOs Plan

The *TransformCIOstoFIOs* plan (Figure 4-17) detects the *TransformCIOtoFIO* event to transform the CIOs to the FIOs. The FIOs are specified based on the attributes of the CIOs and the programming language determined by the developer.

<<Plan>> TransformCIOstoFIOs
Event <i>handle</i> TransformCIOtoFIO <i>send</i> BuiltFunction
Used Belief <i>use</i> Mapping-rule
Method <i>main()</i> { // Handle a <i>TransformCIOtoFIO</i> event // Transform CIO to FIO base on // mapping rule // Send <i>BuiltFunction</i> events }

Figure 4-17: TransformCIOstoFIOs plan structure

For each CIO, a correlative concrete control is created. As discussed, a FIO is defined by the attributes *Label*, *control type*, *editable*, *location* and *Dimension*. These attributes are created as follows:

- The *Label* of the FIO is the *Label* of the CIO.
- The *Control type* of the FIO is determined based on Table 4-4 for the Java and VB.Net languages.
- The value of the FIO's *Editable* is the value of the CIO's *Editable*.
- The *Location* and *Dimension* attributes of a FIO are determined based on the order of creation and the length characteristic.

Determining the control type of the FIO

Specifying the control type of FIO is not difficult, it is based on the control type of CIO and the language chosen by Developer. As shown in Table 4-5, mapping the control type of CIO to the control type of FIO depends on the concrete languages. In this case, these languages are Java and VB.Net.

Note: The controls *Number field* and *Date picker* are not present in Java and the controls *Compo box* and *Tab* are not part of Java-AWT. To implement them we suggest the following:

- Choosing other controls to replace them. For example, Combo box is replaced by List box; Number field is replaced by Text field ...
- Building a new control. For instance, we can create a new *NumberField* which is built by extending *TextField*; this new control only accepts numbers.

DB-USE: Generating UI from Task, Domain, User Models Combined Together

Control type's CIO	Control type's FIO		
	Java-AWT	Java-SWING	VB.Net
Text field	<i>TextField</i>	<i>JTextField</i>	<i>TextBox</i>
Text box	<i>TextArea</i>	<i>JTextBox</i>	<i>TextBox</i>
Number field			<i>NumTextBox</i>
Combo box		<i>JComboBox</i>	<i>ComboBox</i>
Date picker			<i>DateTimePicker</i>
Check box	<i>Checkbox</i>	<i>JCheckBox</i>	<i>CheckBox</i>
Radio control	<i>Choice</i>	<i>JRadioButton</i>	<i>RadioButton</i>
Table	<i>Table</i>	<i>JTable</i>	<i>DataTable</i>
Container	<i>Container</i>	<i>JContainer</i>	<i>ContainerControl</i>
Tab		<i>JTabbedPane</i>	<i>TabPage</i>
Panel	<i>Panel</i>	<i>JPanel</i>	<i>Panel</i>
Button	<i>Button</i>	<i>JButton</i>	<i>Button</i>
Menu	<i>Menu</i>	<i>JMenu</i>	<i>Menu</i>
Pop-up menu	<i>PopupMenu</i>	<i>JPopupMenu</i>	<i>PopupMenu</i>
Label	<i>Label</i>	<i>JLabel</i>	<i>Label</i>
List	<i>List</i>	<i>JList</i>	<i>ListBox</i>
Dialog	<i>Dialog</i>	<i>JDialog</i>	<i>Dialog</i>

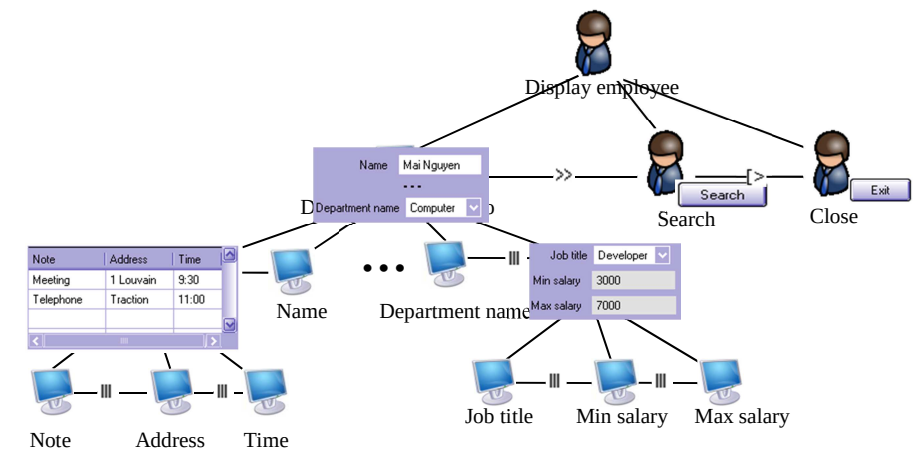
Table 4-5: Mapping the concrete control type based on the language.

Determining the dimension of a FIO

The dimension of a control is described by its height and its width. The controls are determined based on the Length characteristic of the attribute domain and the font size which will be chosen to display data. We suppose that the data that we need to display is a chain of characters; we have two ways to display data: the first way is to display data under a single line and the second one is to display it under multiple lines. The height of a control will depend on its display of a single or multiple lines. Both height and width of a control should have a maximum value and a minimum value. We cannot mention here how to exactly dimension a control but we will present some considerations to evaluate it.

Determining the location of a FIO

Before determining the location of controls we need to determine where the controls will be laid out. Here, we choose a form, a frame or a dialog to contain these controls. The controls are laid out on these containers based on the dimension of the container and the order of the control. For example if the container's width is important then the controls are laid out on more than one column. The controls created from one domain's object are grouped together; it means that they are laid out on the same place.



Example 4-9: The CIOs to FIOs Transformation process based on example 4-7.

The *TransformCIOstoFIOs* plan finishes its task by sending the *BuiltFunction* event to the **Code Generator** agent to generate the code for performing the user interface.

4.2.2.4 Functions Editor Agent

The **Function Editor** agent (Figure 4-18) uses the *BuiltFunctions* plan to build the generic functions based on the domain's attributes, FIOs and the functions are linked and created at previous steps. In the domain model, the functions are only defined in terms of prototype. Once, the components including domain's attributes, FIOs are determined then the linked functions are completely built to be able to lay out the code in such a way that it can be compiled then run immediately.

<<Agent >> Function Editor	<<Plan>> BuiltFunctions
Attribute	Event
	sent
	GenUICode
	GenAppCode
	handle
	BuiltFunction
Plan	Used Belief
BuiltFunctions()	use
	<i>Functions-Description base</i>
Belief	Method
<i>private belief</i>	main()
<i>Functions-Description base</i>	{
	// Handle a <i>DefinFunctions</i> event
	// Analyze these functions
	// Built the methods
	// Send <i>MakeLinks</i> event
Method	}
Private DefineFunctions()	

Figure 4-18: The Function Editor agent and DefineFunctions plan structure

DB-USE: Generating UI from Task, Domain, User Models Combined Together

In order to analyze these functions, the **BuiltFunctions** plan design them as design patterns and follow the design pattern framework proposed in [Forb00] for reusability purposes (the generic identified patterns can be reused by others, for instance, patterns objects, attributes and controls can be reused by functions such as Search, AddNew or Display), object-oriented features compliance with the agent paradigm and implementation traceability.

The functions are described in details in Figure 4-18. Each function is structured by the name, the objects and the methods. The different functions have different objects and methods. The number of objects depends on the number of links between the domain's objects and the tasks; these links will be discussed in the next sections.

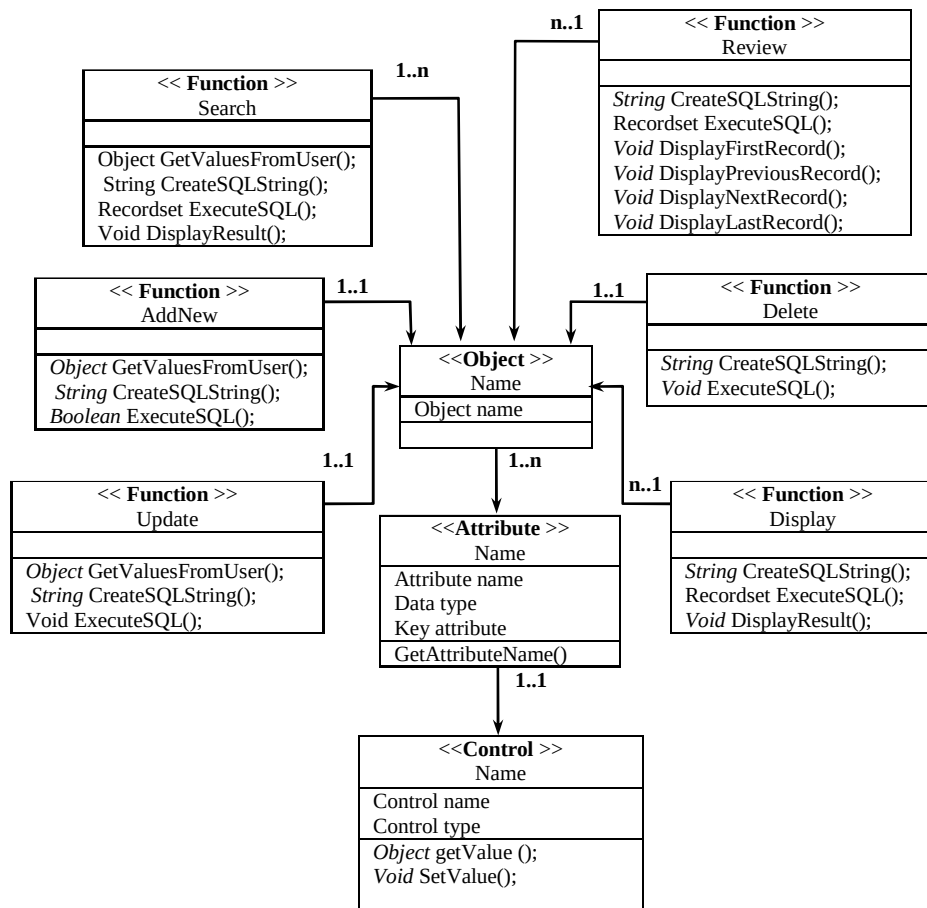


Figure 4-19: Function structures.

One object can have more than one attribute. These attributes are the attributes which are chosen to generate the user interface; they can be a number or all of the original attributes of this object.

One attribute associates with one control which is used to display the attribute's value to or receive the attribute's value from the end-user. One attribute is structured by the name, the data type and the key attribute. For example: attribute Manager{Attribute name: Employee name, Data type: Text, Key attribute: No}

One control is structured by the control name, the control type. The `getValue()` method is used to get data from the end-user and the `setValue()` is used to display data to the end-user. For example: control Manager{Control name: cbManager, Control type: Combobox}

In order to reach the goal, each function has to execute its methods. Each method is executed for different purposes.

- ***GetValuesFromUser()*** method is used to get the necessary values from the end-user. In other words, this method is used to get values from the controls.
- ***CreateSQLString()*** method is used to create the Select/Update/Insert/Delete SQL strings based on the goal of the function. For example: this method create Select SQL string for functions Search, Review and Display; Delete SQL string for function Delete and so on.
- ***ExecuteSQL()*** method is used to execute the SQL string created above. The result of this execution depends on the SQL string. If it is a Select SQL string then the result is a set of the records, otherwise the result is a value true or false.
- ***DisplayResult()*** method is used to display the data to the end-user. In other words, this method is used to set values for the controls.
- ***DisplayFirstRecord()/DisplayPreviousRecord()/DisplayNextRecord()/DisplayLastRecord()*** methods are used to display data from the first/previous/next/last occurrence of the current record.

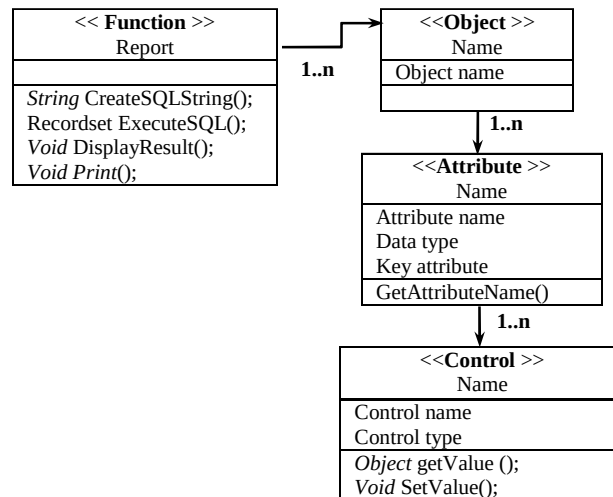


Figure 4-20: Function Report

DB-USE: Generating UI from Task, Domain, User Models Combined Together

If there is a new function added into the domain at the previous steps then there is a new pattern created at this step. For example: A new function Report() defined into domain by **LoadDomainModel** plan then a new pattern created by **BuiltFunctions** plan as depicted in Figure 4-19.

And finally, GenUICode, GenAppCode events are sent by **BuiltFunctions** plan.

4.2.2.5 Code Generator Agent

The **Code Generator** agent uses the **GenerateUICode** plan to generate the user interface code based on the determined FIOs and the **GenerateApplicationCode** plan to generate the application code based on the defined functions.

The structure of the **Code Generator** agent is illustrated in Figure 4-21.

<<Agent >> Code Generator	
Attribute	
Plan	GenerateUICode() GenerateApplicationCode()
Belief	<i>private belief</i>
Method	Private String GenUICode(Object FIOs) Private String GenAppCode()

Figure 4-21: The Code Generator agent structure

An application program can be divided in two parts: the user interface and the code behind the interface that implements the internal logic of the program and interacts with external entities (e.g., database servers). The user interface is determined by the agents presented above; the **CodeGenerator** agent will generate the code to implement the UI.

a GenerateUICode Plan

The **GenerateUICode** plan (Figure 4-22) is activated to generate the user interface code to implement the user interface based on the FIOs determined above when it received the *GenUICode* event.

Here are some important steps for generating the UI for one FIO:

- Create a new control.
- Set the label for created control.
- Set the dimension of the control.
- Set the location for this control on its container.

- Set the editable attribute of the control.

<<Plan>> GenerateUICode
Event <i>Send</i> Finish handle GenUICode
Used Belief
Method <i>main()</i> { // Handle a <i>GenUICode</i> event // Generate UI code based on the FIOs and // the language specified // Post <i>Finish</i> event }

Figure 4-22: GenerateUICode plan structure

For the same FIOs, the code that is generated is different depending on the selected language. For example, the following two code extracts are generated for implementing a FIO *Alias-name* of type *Text* in two different languages:

In Java AWT:

```
TextField name = new TextField();
Label l_name = new Label();
l_name.setText( "Alias-name " );
name.setLocation(100, 50);
name.setSize(200, 30);
name.setEditable(true);
```

In VB.Net:

```
Button1 As System.Windows.Forms.Button
Me.TextBox1 = New System.Windows.Forms.TextBox()
Me.TextBox1.Location = New System.Drawing.Point(32, 40)
Me.TextBox1.Multiline = False
Me.TextBox1.Name = "Alias-name"
Me.TextBox1.Size = New System.Drawing.Size(408, 208)
Me.TextBox1.TabIndex = 0
Me.TextBox1.Text = ""
```

The example shows that they use two different libraries; the syntax of the two languages is different.

Some important points need to be considered when generating code:

DB-USE: Generating UI from Task, Domain, User Models Combined Together

- The code syntax is generated differently for different languages.
- The code generated should be identified, clear and easy to understand; this is very important to maintain, enhance and develop this code.
- The control name is unique so we have to find a solution for naming a control automatically in such a way that the name generated relates to the FUI name and is unique too. Creation of the name must be uniform and standardized since we need to use these names when we generate the code to display and update data.
- The controls are created in a concrete language based on the attributes of FUIs; the control type is determined by *Control type* of FUIs; this control is named by *name* of FUIs, etc.

In our process, the concrete controls are laid out in a form, or a dialogue. We suggest create a template file to perform the form/dialogue. The format of the template file depends on the language on which our application is developed such as Java. Besides, mouse listener, action listener and their functions are added into the template file.

Finally, this plan sends *finish* event.

b GenerateApplicationCode Plan

The **GenerateApplicationCode** plan (Figure 4-23) is activated to generate the application code to implement the defined functions based on the functions linked to the tasks when it received the *GenAppCode* event. The Message base is used as a library for announcing error, the result of performing of the task, confirmation and so on to the end-users.

<<Plan>> GenerateApplicationCode
Event Send Finish handle GenAppCode
Used Belief Message-Base
Method main() { // Handle a <i>GenAppCode</i> event // Generate application code based on // the functions linked to the tasks // Send <i>Finish</i> event }

Figure 4-23: GenerateAppCode plan structure

The Message-Base is created by DB-USE; it is used to provide the generic messages of an application. These messages are used to announce an error, a warning, or the information to the user and to communicate with the user to confirm the tasks which will be performed later. For example: in order to announce to end-user that he/she doesn't have the privilege to work on employee's data, a message "You do not have the privilege to work on this field" is displayed to the user.

The messages used in our process are defined in Table 4-6.

Type of Message	Message
ERROR	Inputted data is not correct
ERROR	Data type is not entered correctly
ERROR	Data is not inserted
ERROR	It is not successful
ERROR	Exception occurred
ERROR	Data type is not valid
ERROR	Database is not connected
ERROR	Data is duplicate
ERROR	An error is occurred
ERROR	Fail
CONFIRMATION	Do you want to save this record ?
CONFIRMATION	Do you want to delete this record ?
CONFIRMATION	Are you sure you want to change the data ?
CONFIRMATION	Do you want to logout ?
CONFIRMATION	Do you want to continue ?
INFORMATION	Data has changed
INFORMATION	A new record is inserted successfully
INFORMATION	Data is deleted successfully
INFORMATION	Connected
WARNING	You have to complete this form
WARNING	This field can not be null
WARNING	Please enter value for this field
WARNING	Can not be found
WARNING	You do not have the privilege to work on this field

Table 4-6: Message table

Here are some important points needed to generate the application code:

- In order to be easily manipulated, the system should load the data for the *Compo-boxes*. This data is derived from a concrete database. For instance, the data of the compo-box *Job title* is selected from the attribute *Job Title* of the object *Jobs*.
- Determine controls that concern each other. These controls are controls that are created from an object (which is not *edited object*). In practice, that is a group composed of a control compo-box and its relating controls (text-field, number-field, check-box). For instance, when the end-user changes the value of the compo-box *Job* then the values of the number-field *Min salary* and *Max salary* are automatically changed. See Example 4-10.

DB-USE: Generating UI from Task, Domain, User Models Combined Together

Example 4-10: The Value Change Of The Data Fields

The process finishes by the sending of a finish event by this plan.

4.2.3 DB-USE Dynamic Diagram

As said before, a plan can be invoked by an event that it handles, and can create new events. The relationships between plans and events are described in Figure 4-24.

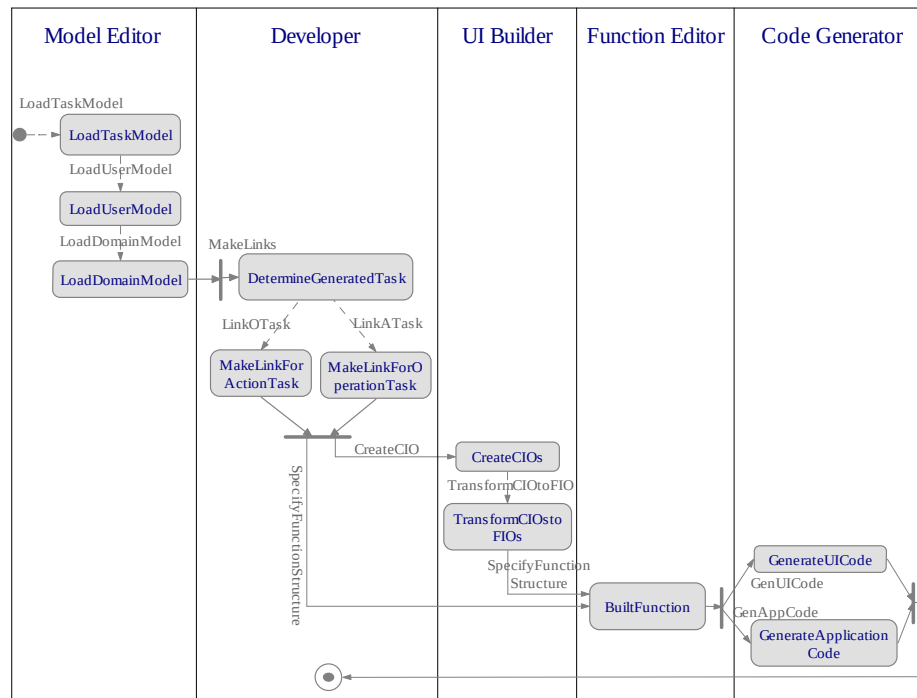


Figure 4-24: Code Generation Workflow

The diagram models the control flow from the **Model Editor** agent defining the basic functions to the **Code Generator** agent creating the user interface. The plans and their events discussed above are summarized below, including some specific points of the workflow, in the Dynamic Diagram.

The process starts with loading the models. Once the **LoadTaskModel** plan received the LoadTaskModel event, it reads the information of the task model from a XML file then loads this model in terms of our format which is represented by only two task types.

After the **LoadUserModel** plan received the LoaduserModel event from the **LoadTaskModel** plan, it analyzes the information of user model to determine the preferences of the user and posts the LoadDomainModel event to load the domain model.

The **LoadDomainModel** plan detects the LoadDomainModel event to load the domain model from a concrete database which is determined by developer.

The **DetermineGeneratedTask** plan is activated when it receives the MakeLinks event from the **LoadDomainModel** plan. The **DetermineGeneratedTask** plan selects a task based on which the user interface will be generated. The MakeALink and MakeOLink events are posted by this plan.

The **MakeLinkForInformationTask** plan is used to make links between the information task and the attributes of the domain model when this plan receives the MakeOLink event.

The **MakeLinkForActionTask** plan is used to make links between the action task and the defined functions when this plan receives the MakeALink event.

The **CreateCIOs** plan is used to create the CIOs based on links between the tasks and the defined functions, the attributes of the domain model when the CreateCIO event is detected by this plan.

These CIOs are transformed to FIOs by the **TransformCIOstoFIOs** plan. The **TransformCIOstoFIOs** plan sends the SpecifyFunctionStructure event to **Function Editor** agent.

The **SpecifyFunctionStructure** plan is activated if it has received the SpecifyFunctionStructure event from both the MakeLinkForActionTask and the **TransformCIOstoFIOs** plans. This plan specifies the structure of the functions linked by the developer. The GenUICode and GenAppCode events are sent to **Code Generator** agent to generate the code.

The **GenerateUICode** plan is triggered once it has handled the GenUICode events to generate the UI code which performs the user interface. The **GenerateApplicationCode** plan will generate the code to perform functions of our application when this plan received GenAppCode event.

The generated code is returned to the **Developer**.

4.4 Chapter Summary

Chapter 4 has presented the core of this thesis. DB-USE process is started with presenting main components in DB-USE's architecture; then methodology of DB-USE is presented to show the reader the steps in this process, the resources used in this process and the participants who are responsible for each steps; and finally, these steps are discussed in detail by using Multi-agent system. In order to apply this process, DB-USE tool will be implemented in chapter 5.

Chapter Summary

Chapter 5

DB-USE: Software and Case Study

In this chapter we present a tool implemented based on the process discussed in Chapter 4. The DB-USE tool is presented in Section 5.1. There are two case studies in Section 5.2 including human resource management and TransLogisTIC. The comparison between DB-USE and Just-UI, Vampire, and patterns of Coad & Yourdon is discussed in Section 5.3.

5.1 DB-USE Tool

Multi-agent system development does not require specific agent-oriented language and can be implemented with language based on other paradigm. Indeed, the concept of *Agent* is not too distant from the concept of *Object* so that object-oriented languages are considered as suitable means. In practice, most proprieties of an agent are the same with the ones of an object such as inheritance, message passing. However, the other proprieties of an agent such as autonomy, flexibility behaviour are not supported by the object.

Today, several software frameworks are available for implementing multi-agent systems such as Jack (Java Agent Compiler and Kernel), Jade (Java Agent DEvelopment framework), 3APL (An Abstract Agent Programming Language). These frameworks have been fully implemented in Java. They provide programming constructs for implementing agents' beliefs, goals, services and basic capabilities (such as belief updates, external actions, or communication actions). Therefore, to implement a multi-agent system in Java instead of using Java language, the developer can use these frameworks.

The agents of DB-USE have been implemented in Java, widely used in the large database-oriented software packages such as Oracle or SAP. Java language has been chosen to implement DB-USE tool for following reasons:

- Java is an object-oriented language.
- DB-USE does not need the advanced feature provided by Jack, Jade and 3APL into its specific developments.
- DB-USE could subsequently slow down the execution process when it does not need to run these frameworks.

Table 5-1 illustrates the mapping between agent components and DB-USE components in Java. Each agent is presented by a class; its plans are the methods used for communication means with other agents.

Agent components	DB-USE components in Java
Agents	Classes
Attributes	Attributes
Belief	XML files, database
Plans	Public methods
Methods	Private methods
Events	Messages

Table 5-1: Agents of DB-USE in Java

The main purpose of DB-USE is to help designers to create the user interface and generate the application code for performing the generic functions of a database application easily and quickly. We summarize below the important goals which DB-USE aims to achieve:

- ***A friendly and simple interface:*** The interface of DB-USE has been designed in such a way that any designer can work with it while benefiting from the important features mentioned above. Similarly to other applications, the main interest of our tool is the visual and/or graphical generated result.
- ***Performing most of the functions discussed in the domain model:*** DB-USE has been developed to perform most of the functions defined in the domain model. These important functions makes the user interface directly specified based on the associations between the task and domain models; and the capability of the application code generation.
- ***Efficient managing existing resources:*** As already discussed, the task, user and domain models used in our process are the existing resources; they are used by the other processes such the analysis process.
- ***Visualizing the process:*** The interface visually supports our process as much as possible. Currently, the tasks are displayed in terms of graphical objects. The designer can review the result in terms of graphics whenever he/she wants during working.
- ***Generating and running the code:*** In the current version, the user interface and application codes generated by DB-USE are implemented in Java. The code generated has been made clear, easy to understand and documented for example: how to name the controls: the controls should be named based on the control type (T for text-field, C for combo-box, B for button ...) and the task name. Typically, the code generated by DB-USE can be immediately compiled by a Java compiler.

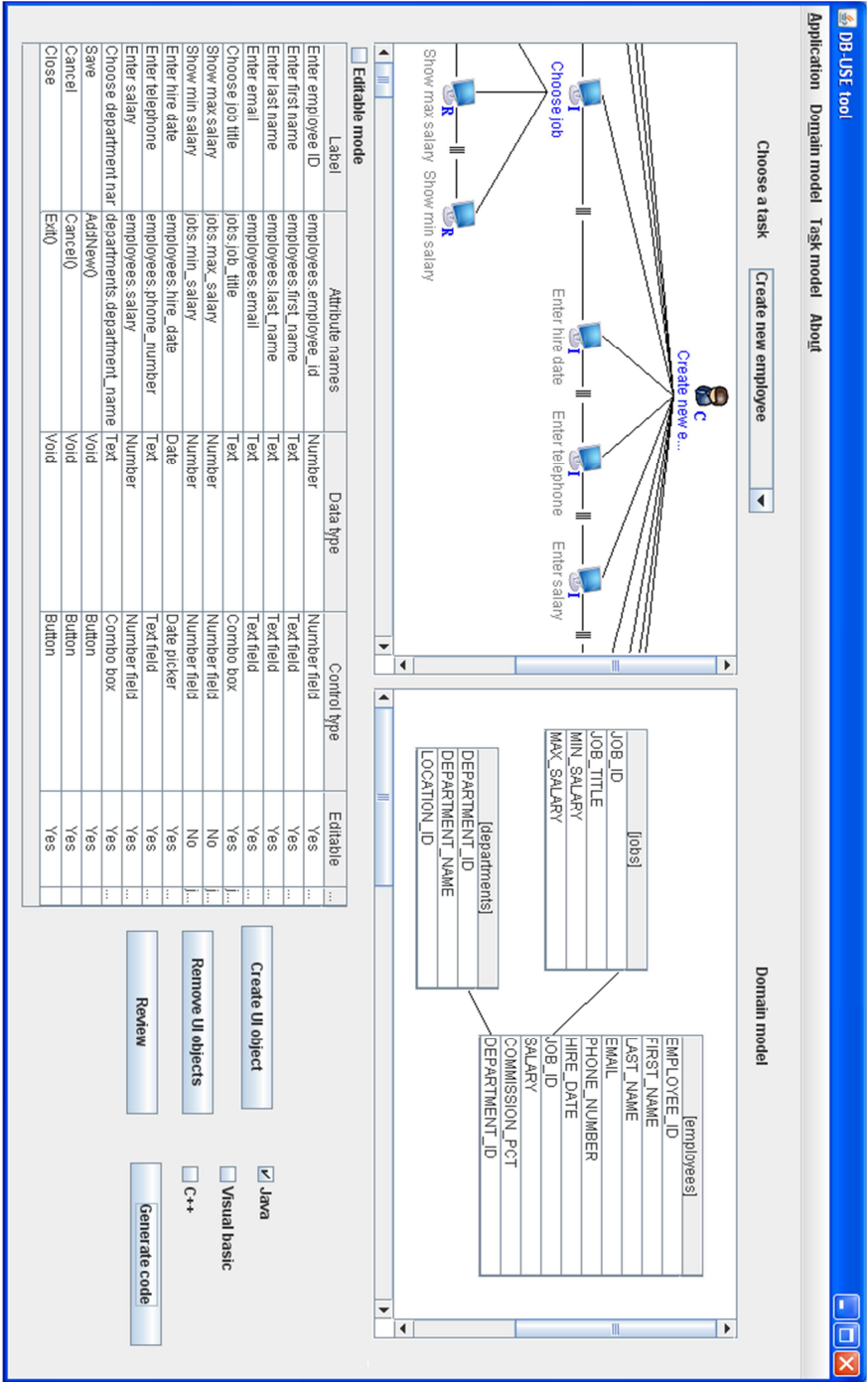


Figure 5-1: DB-USE tool

DB-USE Tool

Figure 5-1 depicts DB-USE tool. The tool is divided into three separate areas. The first area is used to display task model; the second area is used to display the domain model and the last area is used to display the information of the UI objects.

Unlike with the current tools such as Teallach, Goliath, Fuse, the user interface designers do not need to manipulate much to create the interfaces. They can work with the task, user and domain models by using the mouse to select the objects of DB-USE's models and link them together. Besides, the tasks in the task model are described in terms of graphics designed to represent the role of these tasks; making it clear to the designer.

The DB-USE's model can be developed independently.

In this current version, the user interface code and application code generated by DB-USE tool are also implemented in Java. The code generated is very clear and easy to understand; unlike with most of previous tools.

5.1.1 Model Editors

As discussed above, task model and domain model used in DB-USE are the existing resources. Modeling editors of DB-USE have been developed to support the UI designer in displaying a model or a part of the model, selecting the objects, making the associations between the domain's objects and the tasks. With this version of DB-USE, the generated user interfaces aim to the general end-users; so that the user model is not presented. Maybe the user model will be used in the next version of DB-USE tool.

5.1.1.1 Task Model Editor

Task model editor of DB-USE allows the designer:

- Load the task which is created by CTT tool then save it in term of the format of DB-USE task model.
- Create a new DB-USE task model.
- Load DB-USE task model.

The task types defined by CTT are different from the task types defined by DB-USE. So in order to reuse this task model we need to transform the task types of CTT to that of DB-USE (See Table 4-3). In DB-USE, we have two types of tasks: information task and action task. These tasks are presented in Section 3.2.2 and Section 4.2.2.2.i. Information task is represented by icon  and action task is represented by icon . The relationship between the two tasks is represented by a straight line.

Once CTT task model is loaded, the designer can modify it and save it by clicking on *Task model \ Save task model* menu.

The task model editor is depicted in Figure 5-2.

In order to load the DB-USE task model, the designer has to select *Open DB-USE task model* menu.

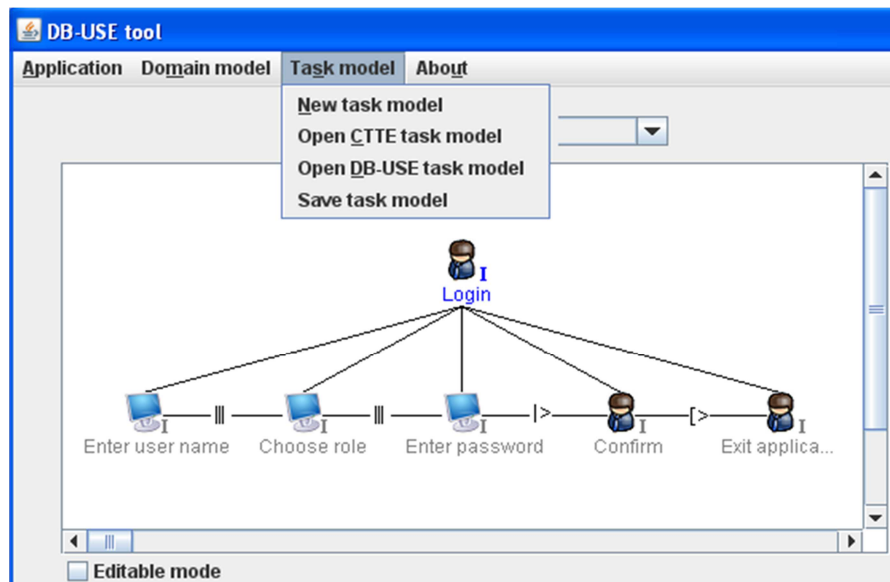


Figure 5-2: Task model editor

Unlike in current applications such as Teallach [Barc99], Trident [Boda95], Goliath [Juli04b], task model editor is displayed in terms of graphics and hierarchical tree structure. The different tasks are represented by different graphical icons; making it easier to interpret the role of the tasks. The hierarchical tree structure is used in DB-USE tool because it allows easy observation of the flow of the tasks. Besides, the action types are also represented to the designer/developer by the I, C, D, U, S, R symbols. The temporal operator between two task at the same level is represented by the $\parallel$, $[>$, $|>$, $>>$, and so on, icons.

Once a task is chosen by the designer/developer, this task and all of its sub tasks are displayed on the task panel.

To increase ease of use, DB-USE tool provides two options: editing mode and liking mode.

a Editing Mode

The editing mode is activated once the check box “*Editable mode*” is checked by the designer/developer. The main goal of this mode is for the designer to see the structure of task tree and modify the tasks.

Based on the task chosen at compo box “*Choose a task*”, DB-USE will display all of the tasks whose root is the task chosen. At this mode, the designer cannot

make the associations between the tasks and the domain components but he/she can modify the attributes of the task, add new a task, and delete an existing one.

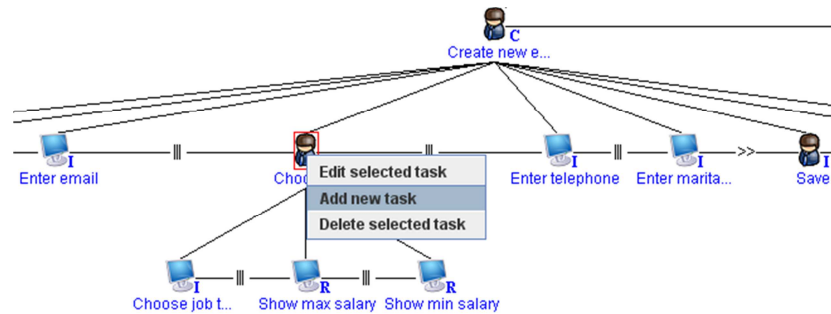
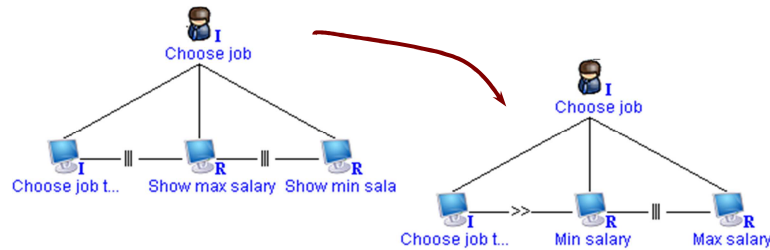


Figure 5-3: Task menu

Figure 5-3 shows you how to edit the task model. Firstly, the designer has to choose the task that he/she wants to edit. Then by clicking on the right button, a task editing menu is displayed. To modify the selected task, one will select the menu “*Edit selected task*”; to add a new task, the menu “*Add new task*”; to delete selected task, the menu “*Delete selected task*”.

Figure 5-4: Task form

The task form depicted in Figure 5-4 is used to create a new task and to modify the task attributes. The designer can decide the task’s position among the tasks which have the same parent task by choosing the right task; but cannot change the parent task. For example: In Example 5-1, the task name of *Show max salary* and *Show min salary* have been modified and their positions have been changed. Besides, the temporal operator of task *Choose job title* is also changed from **Interleaving** to **Enabling**. Once the designer finished the change, he/she can click on the “*Save*” button in order to confirm this change; otherwise, he/she clicks on the “*Cancel*” button.



Example 5-1: task *Choose job* title is modified

Once a task is deleted, all of its sub-tasks are also deleted. For example: if the *Choose job* task in Figure 5-3 is deleted then *Choose job title*, *Show min salary*, *Show max salary* tasks are also deleted.

b Linking Mode

The linking mode is activated once the check box “*Editable mode*” is unchecked. In order to reduce the complexity of task model during creation of the UI objects, when the designer has chosen a task from the compo-box “*Choose a task*”, DB-USE displays only the tasks needed (See Section 4.2.2.2.i) for creating the UI. In this mode, the designer can make the associations between the tasks and the components of domain model including the domain’s attributes, the generic functions. Once the task has been linked to the domain’s component, it is displayed in inactive mode.

c Algorithms

In this Section, we discuss some algorithms for determining the task position in the task tree, deleting the tasks from the task tree, and inserting a new task into the task tree. We suppose that task *T* is located in the task panel by *T.x* and *T.y*; the first position of a task is named *Pmin(x, y)*; the distance between two tasks in the horizontal line is named by *X*; in the vertical line it is named by *Y*.

i Positioning the tasks in the task tree

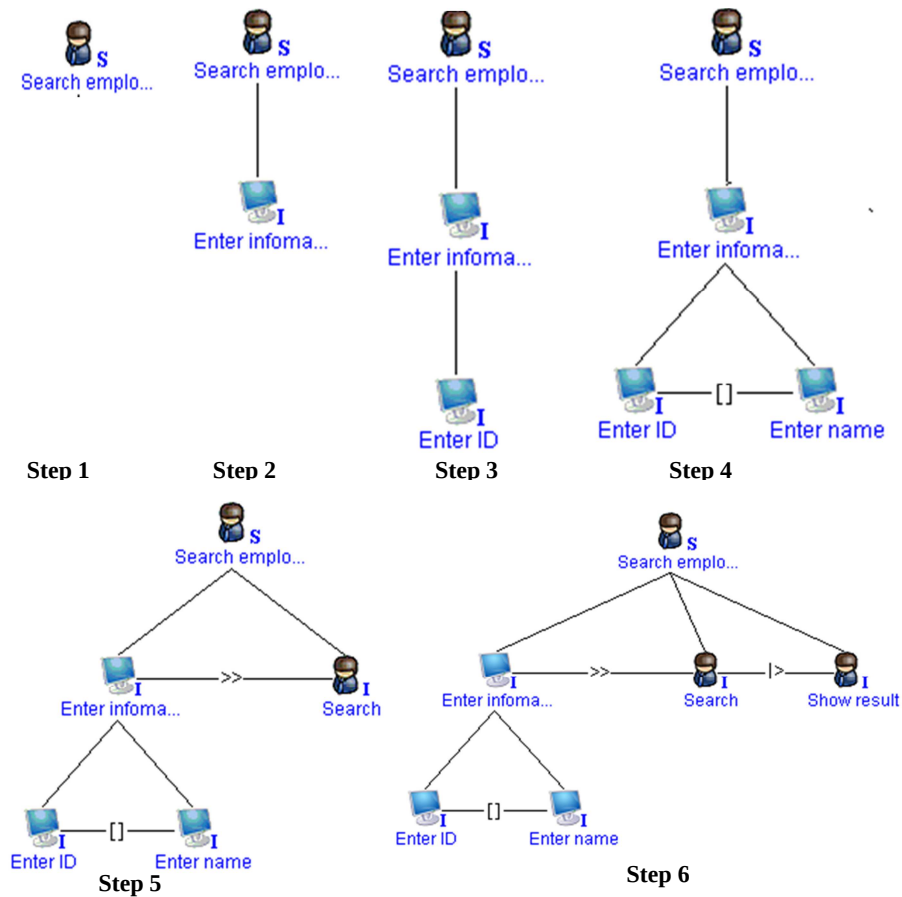
We suppose that *Xmax* is the *T.x* of the last task which was set in position. In order to set the position of the task in the tree we need to build the *SetTaskPostion(Task)* and *recalculatePositionofParentTask(Task)* methods. These methods are recursive. Setting the task position starts with the *SetTaskPostion(T)* method; *T* is the root task.

The *SetTaskPostion(Task)* method is used to set the position of the task *T* based on its parent task and *Xmax*. If *T* is the root task then its position is *Pmin*. Once there is a task added into the task tree that changes the width of the tree, the *recalculatePositionofParentTask (Task)* method is performed to modify the position of its parent task.

```
setTaskPosition(Task T)
{
    // set position for the root task
    if (T.isRoot())
    {
        T.x = Pmin.x;
        T.y = Pmin.y;
    }
    // get the sub-tasks of task T
    Ts[n] = getSubTasks(T);
    For (int i = 0; i < n; i++)
    {
        if (i == 0)
        {
            Ts[i].x = T.x;
            Ts[i].y = T.y + Y;
        }
        else
        {
            Xmax = Xmax + X ;
            Ts[i].x = Xmax;
            Ts[i].y = T.y + Y;
            // recalculate the position of its parent
            //task
            recalculatePositionofParentTask(Ts[i]) ;
        }
        // set position for the sub-task of task T
        setTaskPosition(Ts[i]);
    }
}
```

The `recalculatePositionofParentTask(Task)` method is used to recalculate the task position.

```
recalculatePositionofParentTask(Task T)
{
    if (T.isRoot())
        return ;
    Tp = getParentTask(T) ;
    Tp.x = Tp.x + X/2 ;
    recalculatePositionofParentTask(Tp) ;
}
```



Example 5-2: Setting task position

Example 5-2 depicts the setting of the task position.

i Adding a new task into the task tree

There are two important points when adding a new task T into the task tree. The first point is to update the tasks around task T and the second point is to recalculate the task position. The `addNewTask(Task)` method is used to add a new task into the task tree.

```
addNewTask(Task T)
{
    // get the right task of task T
    Tr = getRightTask(T);
    if (Tr != null)
    {
        // get left task of the task Tr
    }
}
```

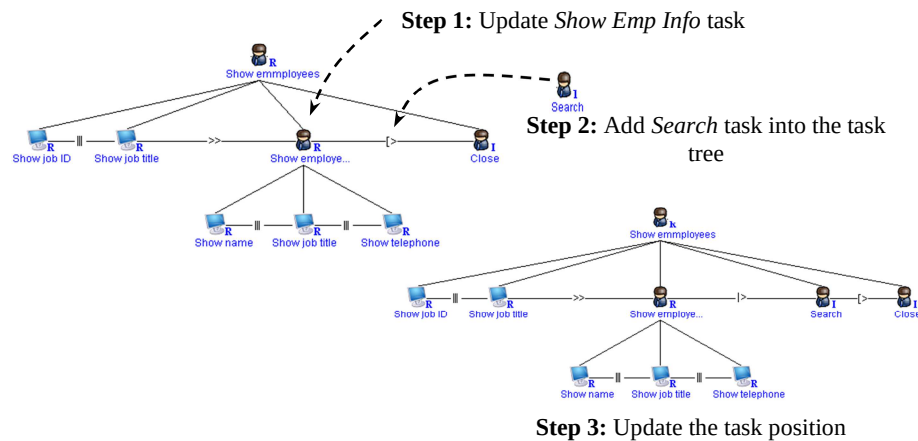
```

    Tf = getLeftTask(Tr);
    if (Tf != null)
    {
        //update right-task attribute for the task Tf
        Tf.setRightTaskID(T.getTaskID());
    }
}

// add task T into the task tree
addTask(T);

// Recalculate the task position;
setTaskPostion(root);
}

```



Example 5-3: Adding a new task into the task tree

Example 5-3 depicts the addition of the *Search* task into the task tree. In order to insert this task between the *Show emp Info* and *Close* tasks, the *Show emp Info* task is automatically updated by DB-USE.

ii Deleting a selected task from the task tree

Before deleting a task, DB-USE has to update the left task of this task by executing the `updateLeftTask(Task)` method. The `deleteTask(Task)` method is used to delete the task *T*; this method is recursive. If the task *T* is deleted that means all of its sub-tasks are also deleted; the deletion of the tasks is started with the leaf tasks. Once the `deleteTask(Task)` method finishes, the `setTaskPostion(root)` method is executed to recalculate the task position.

```

updateLeftTask(Task T)
{
    // get the left task of task T

```

```

Tf = getLeftTask(T);
if (Tf != null)
{
    // update right-task attribute for its left task
    Tf.setRightTaskID(T.getRightTaskID());

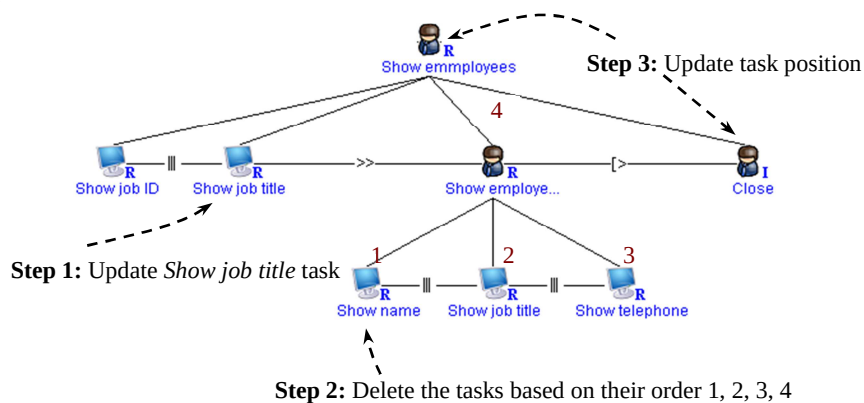
    // update operator attribute for its left task
    Tf.setOperator(T.getOperator());
}

// Call deleteTask(T) method;
deleteTask(T);

// Recalculate the task position;
setTaskPostion(root);
}

deleteTask(Task T)
{
    // get the sub-tasks of task T
    Ts[n] = getSubTasks(T);
    For (int i = 0; i < n; i++)
    {
        // call deleteTask in order to delete the
        // sub-tasks of task T
        deleteTask(Ts[i]);
    }
    // Delete task T if it is a leaf task
    T.delete();
}

```



Example 5-4: Deleting a task from the task tree

Example 5-4 depicts the deletion of the *Show Employee Information* task from the task tree.

5.1.1.2 Domain Model Editor

DB-USE's domain model is defined in Section 3.3. This model is very important for specifying the UI objects. As mentioned, DB-USE's domain model is loaded from a concrete database which is determined by the designer (See Figure 5-5).



Figure 5-5: Login dialogue

As discussed in *LoadDomainModel* plan (See Section 4.2.2.2), the domain's objects are differently stored by the different databases. The current version of DB-USE only supports Oracle system. We hope that in the next version, DB-USE tool can support multiple types of databases such as SQL Server, My SQL.

By using DB-USE you can display all of the objects in the domain model or only some of them by selecting the domain's objects that you want to work on. The characteristic of DB-USE tool is that the designer works on several tasks at a time. So it is better to display only the domain's objects which are used to link to these tasks.

In order to help the designer to correctly and quickly select the objects from domain model based on the selected tasks, we suggest some following important points:

- Designer needs to specify the goal of the selected task root. Based on this goal the edited object is determined.
- Once the edited object has been determined, DB-USE allows the designer choose one in two options including "Display all" (DB-USE displays all of objects in domain model) and "Display objects related to the edited object" (DB-USE displays all tables which relate directly to the edited object).
- Based on the goal of children tasks, the designer selects the other objects.

The designer can choose the needed objects by using domain's object dialog (See Figure 5-6). The domain's object in Edited objects and Selected objects list boxes are selected from Domain's object list box; these objects will be displayed in the panel of domain model if the designer clicks on the Ok button (See Figure 5-7). In practice, the designer only chooses the edited object and the objects related to this object. To reduce the complexity for selecting the objects, he/she can un-check Display all option to see all objects which directly relate to edited table.

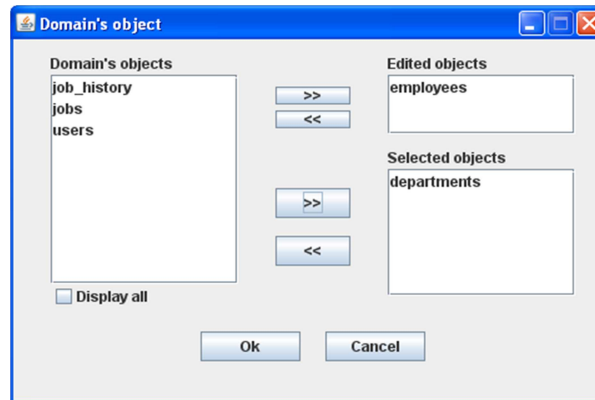


Figure 5-6: Domain's object dialogue

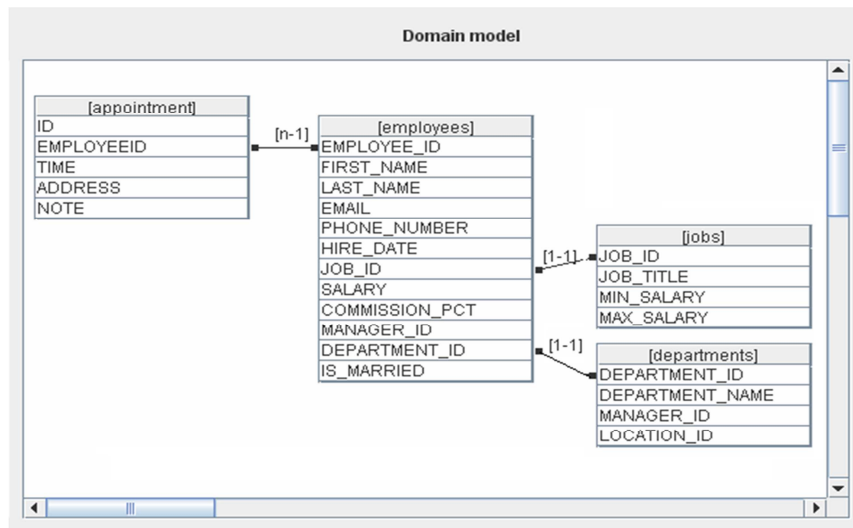


Figure 5-7: Domain's object dialogue

Generic application functions of domain model: a benefit for the designer

As mentioned in previous section, the generic functions of an application are independently defined in DB-USE. These functions are presented in Table 3-9; they are Display(), AddNew(), Update(), Delete(), Search() and Review(). In Teallach and Goliath, the generic application's functions are built by the designer but in DB-USE they are built by DB-USE's system. That is the main advantage of DB-USE over other systems.

DB-USE's generic functions have been designed in terms of the pattern functions to support the designer as much as possible. For instance, in order to display the information of the employees and of the departments, in Teallach and

Goliath, the designer has to build *DisplayEmployees()* and *DisplayDepartments()* functions manually. But in DB-USE, the designer just links the *Display()* function built by the system to *Employees* and *Departments* objects then *DisplayEmployees()* and *DisplayDepartments()* functions are automatically built by the system.

The generic application functions are displayed to the designer in a Function Menu as shown in Figure 5-8. The designer can choose one of them to create a link to the action task. If the designer cannot find the suitable function for the goal of his/her task; he/she can link it to the Default() function.

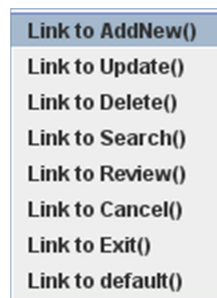


Figure 5-8: The Function Menu

Once a pattern function has been linked to the concrete domain's objects, DB-USE will automatically generate a concrete application function based on the linked function and the relevant objects. This concrete application function will be automatically implemented by Code Generator.

5.1.2 User Interface Creator

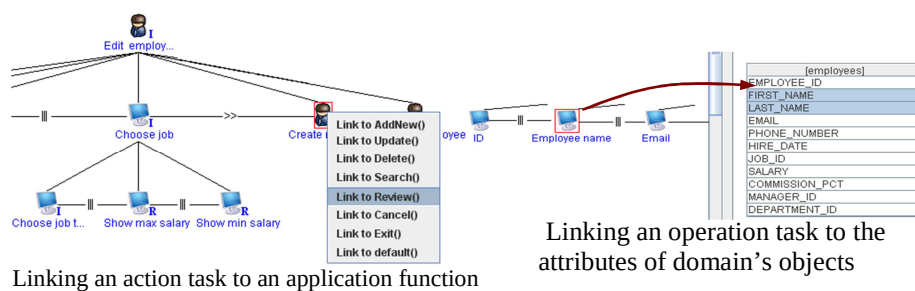


Figure 5-9: Making the associations

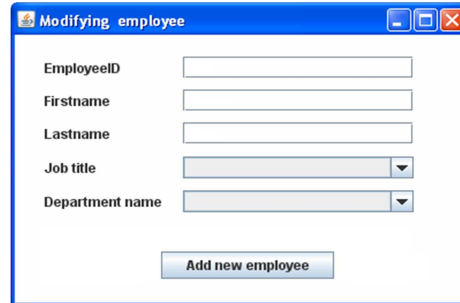
Once the designer has created task and domain models, he/she makes the associations between the information tasks and the attributes of domain's objects and between the action tasks and the defined application functions. The association between the information tasks and the attributes of domain's objects is created by selecting a concrete task and the attributes then clicking the "Create UI object"

Creating the concrete interaction objects

Once a concrete interaction object has been created, the information of this object is displayed in a table as show in Figure 5-10. This information includes the label of control, the data type of the attributes, the control type and the editable attributes of control.

Figure 5-10: User interface objects are displayed in terms of text

In order to help the designer to better figure the user interface which will be generated at the last step of the process, DB-USE supports important function not available in other application. This function is to review the user interface in terms of graphics.



Example 5-5: User interface objects are displayed in terms of graphics

The user interface is displayed in terms of graphics when the designer clicks on the “Review” button. At this level, the designer just views the interface; the system does not make the demo for the application functions. The user interface objects described in Figure 5-10 are displayed in terms of graphics in Example 5-5.

In order to facilitate the operation, the tasks linked to the attributes of the domain’s objects and to the defined functions are not enabled in the task editor. This means that the designer can see them but he/she cannot focus on or select these tasks. The user interface objects can be deleted by the designer by clicking on the “Remove UI objects” button. And of course, the tasks removed from the above links are enabled automatically by the system.

5.1.3 Code Generator

The code is generated when the designer clicks on the “Generate code” button. In this section, the user interface code generation and the application code generation are presented separately.

The code generator component plays an important role in DB-USE tool. It is used to generate the user interface code based on the user interface objects created and on the application code based on the selected function. As mentioned at the opening of this chapter, the code generator produces Java code by using the Swing, AWT libraries. The generated code is a complete code which can be compiled and ran immediately while the code generated by the other applications is prototype or simple code. This is a strong point of DB-USE tool.

There are some current techniques for generating code such as template-based approaches, generative programming, and code insertion:

- *Template-based approaches* involves generating the code based on a data model (an input model that has been specified) using templates [iMat00]. This code generation process is obviously a transformation process. The data model contains information about the system entities that the developer wants to generate, and the template represents the syntax of the

target programming language. The advantages of this technique are that they provide a clearly defined format and a precise semantic. The disadvantage of it is that the developer needs to create the different libraries of templates for the different programming languages.

- *Generative Programming* is the process to generate programs where automated source code creation is done through code generators to improve programmer productivity [Bart01]. The goal of generative programming is to automate the software creation process to some extent, so that human programmers can concentrate on those areas of software design that really require human input, while offloading certain tedious and error-prone tasks to automatic routines. One of the main remarkable advantages of generative programming is about implementing application code automatically after restricting human interventions to the essentials.
- *Code Insertion* involves inserting code into the target system source just before an event is to occur. The inserted code performs the fault injection and then the original statement can execute with the fault present in the system. The main advantages of Code Insertion are that it is easy to implement. Its main disadvantage is that it requires the system source code to be modified making it hard to use this method for system certification [Look05].

Based on advantages and disadvantages of three techniques, DB-USE has decided to apply Template-based to its process to generate code. The generation process is depicted in Figure 5-11.

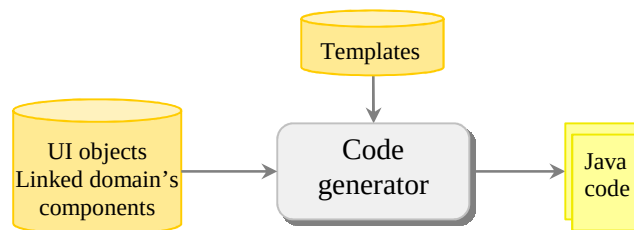


Figure 5-11: The process of code generation

The code generator reads the UI objects and the domain's components which are linked to the tasks then uses a set of templates to build the output code that implements the user interface and the defined functions. These domain's components are the domain objects, domain's attributes and the relationships between the objects.

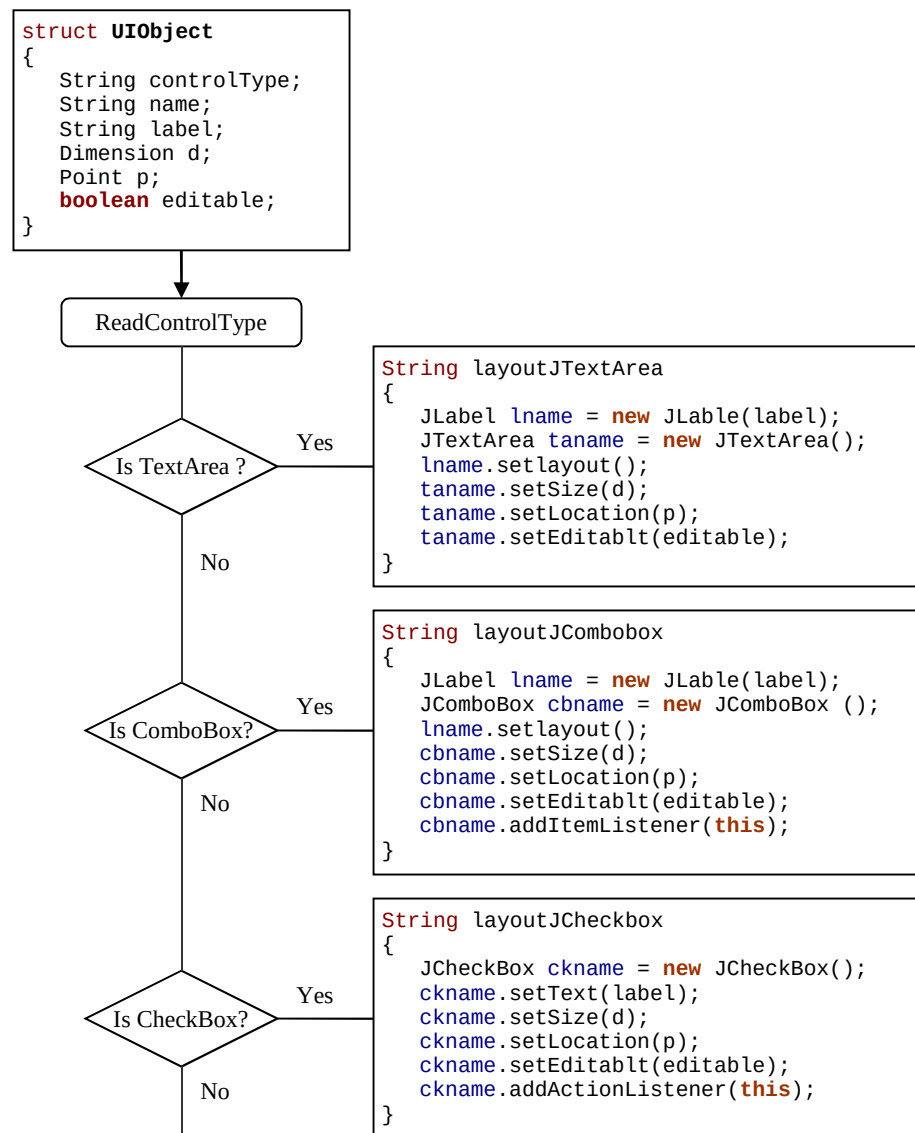
DB-USE is interesting in the representation of the generated code. In order to help the designer to efficiently maintain, enhance and develop the code, the code is generated according to the same structure, thus producing a final code that is consistent. That is one of the reasons why the template-based technique is selected.

In our process, the user interface objects are laid out in a form, a frame or a dialog.

5.1.3.1 User Interface Code Generation

User interface code generation in DB-USE is made by two steps. In first step, DB-USE generates the user interface based the UI objects created in previous steps. In second step, DB-USE generates the code to perform the setting the data for the combo-box or list-box.

a Generating the User Interface



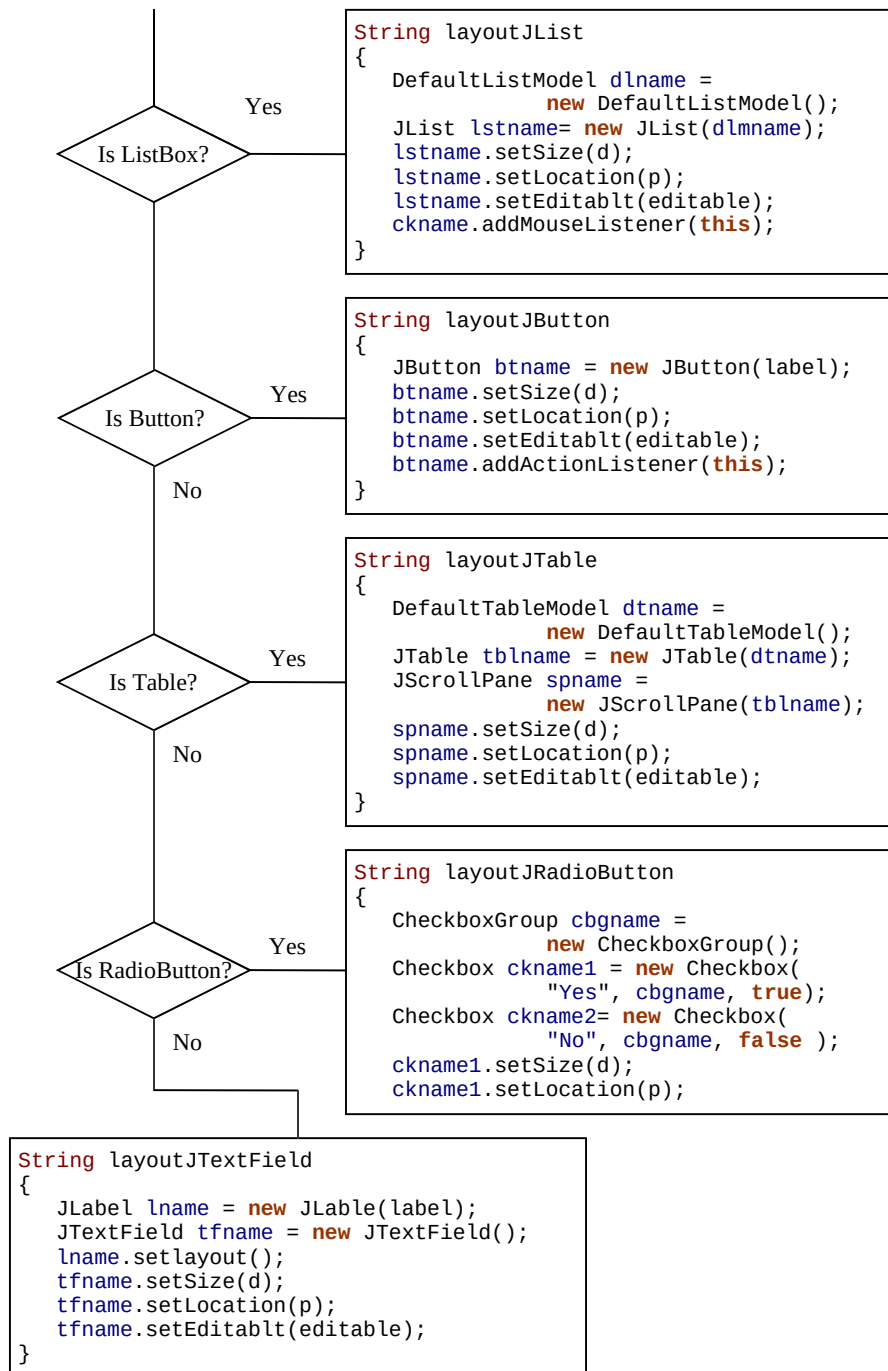


Figure 5-12: Generating the user interface code

The process of user interface generation is depicted in Figure 5-12. The inputs of this process are the UIObjects and the templates of creation of Java controls such as JTextField, Jtable, JtexBox ... The CodeGenerator uses the different templates to produce the Java code based on the different control types of UI object.

b Setting Data for Combo-boxes/List-boxes

Once the combo-box/list-box has been created, DB-USE generates the source code to get the data for it. We only explain how to get data for the combo-box because with the list-box, you can do the same.

Data Type	jDataType
Text	String
Number	int
Yes/No	boolean
Date	Date
Number	Long
Object	Vector

Table 5-2: Data type mapping

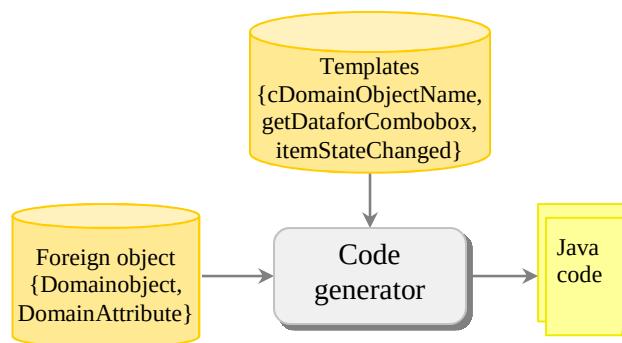


Figure 5-13: Code generation process for setting data of a combo-box

Figure 5-13 depicts code generation process for setting data of a combo-box.

The inputted object of this process is the foreign object which is linked UI object of combo-box. The template of this object is constructed as following:

```

struct DomainObject
{
    String domainObjectName;
    DomainAttribute *selectedAtts;
    String mainAttribute;
}

struct DomainAttribute
{
    String attName;
    String dataType;
    int length;
    boolean isPrimaryKey;
    boolean isForeignKey;
}
  
```

Figure 5-14: Foreign object template

To depict this object in practice, we present *Jobs* object:

```

DomainObject
{
    domainObjectName = "Jobs";
    selectedAtts = {
        {"Job_ID", "String", 20, true, false},
        {"Jobtitle", "String", 30, false, false},
        {"Maxsalary", "int", 15, false, false},
        {"Minsalary", "int", 15, false, false},
    };
    mainAttribute = "Jobtitle";
}

```

Example 5-6: Jobs – foreign object

To store the data records of foreign object, DB-USE uses the following template.

```

class cDomainObjectName
{
    jDataType attName1;
    jDataType attName2;
    ...
    public void setAttName1(jDataType value)
    {
        attName1 = value;
    }
    public jDataType getAttName1()
    {
        Return attName1;
    }
    ...
}

```

Figure 5-15: Data record template

The class name is the combination of the letter “c” and name of the foreign object. jDataType is determined based on the data type of the attribute (See Table 5-2). The attributes of this class are specified based on the linked attributes in this domain’s object; the number of attributes in class is the number of attributes in the foreign object. For each attribute in the class, there are two methods created: one for setting value and one for getting value.

The *Jobs* object is generated as following:

```

public class cjobs{
    private String JOB_ID = "";
    private String Jobtitle = "";
    private int Maxsalary = 0;
    private int Minsalary = 0;

    public void setJOB_ID(String value){
        JOB_ID = value;
    }
    public String getJOB_ID(){
        return JOB_ID;
    }
    public void setJobtitle(String value){
        Jobtitle = value;
    }
}

```

```
public String getJobtitle(){
    return Jobtitle;
}
public void setMaxsalary(int value){
    Maxsalary = value;
}
public int getMaxsalary(){
    return Maxsalary;
}
public void setMinsalary(int value){
    Minsalary = value;
}
public int getMinsalary(){
    return Minsalary;
}
};
```

Example 5-7: Jobs – Data record

getDataforCombobox template is used to get data from the database based on the foreign object (DomainName) and its attributes (attName1, attName2 ...). This template is constructed in Figure 5-16.

```
void getDataforCombobox(JCombobox cb, ResultSet rs, Statement s)
{
    try{
        rs = s.executeQuery("Select attName1, attName2 ...From DomainName");
        while (rs.next()){
            cDomainObjectName temp = new cDomainObjectName ();
            temp.setJOB_ID( rs.getString(attName) );
            vDName.add( temp ); //vDName: Vector used to store the data
            cb.addItem( rs.getString(mainAttribute) );
        }
    }
    catch (Exception e){
        System.out.println(e.toString());
    }
}
```

Figure 5-16: getDataforCombobox template

```
private void getDataforCombobox(JCombobox cb, ResultSet rs, Statement s)
{
    try{
        rs = s.executeQuery("Select JOB_ID, Jobtitle, Minsalary, Maxsalary
                             From jobs");
        while (rs.next()){
            cjobs temp = new cjobs();
            temp.setJOB_ID( rs.getString("JOB_ID") );
            temp.setJobtitle( rs.getString("Jobtitle") );
            temp.setMaxsalary( Integer.parseInt(rs.getString("Maxsalary")) );
            temp.setMinsalary( Integer.parseInt(rs.getString("Minsalary")) );
            vjobs.add( temp );
            cb.addItem( rs.getString("Jobtitle") );
        }
    }
    catch (Exception e){ System.out.println(e.toString());
    }
}
```

Example 5-8: Jobs – getDataforCombobox

Example 5-8 depicts getDataforCombobox template by using Example 5-6 and 5-17.

To automatically change the value of the controls which have a connection with combo-box when the value of this combo-box is changed, DB-USE inserts the source code into itemStateChanged() method to display data correctly (See Figure 5-17).

```
public void itemStateChanged(ItemEvent evt)
{
    if (evt.getSource().equals(comboboxName))
    {
        //Reset value for the other controls which involve with this cb
    }
}
```

Figure 5-17: itemStateChanged template

itemStateChanged example is depicted in Example 5-9.

```
public void itemStateChanged(ItemEvent evt)
{
    if (evt.getSource().equals(cbJobtitle))
    {
        tfMaxsalary.setText(((cjobs)vjobs.elementAt(
            cbJobtitle.getSelectedIndex()).getMaxsalary());
        tfMinsalary.setText(((cjobs)vjobs.elementAt(
            cbJobtitle.getSelectedIndex()).getMinsalary());
    }
}
```

Example 5-9: Jobs – itemStateChanged

5.1.3.2 Application Code Generation

The application code is generated based on the methods of the functions linked to the tasks and on the objects which are associated with these functions (See Figure 4-6). The different methods attend to the different objectives; they are activated in order as described in the functions defined above.

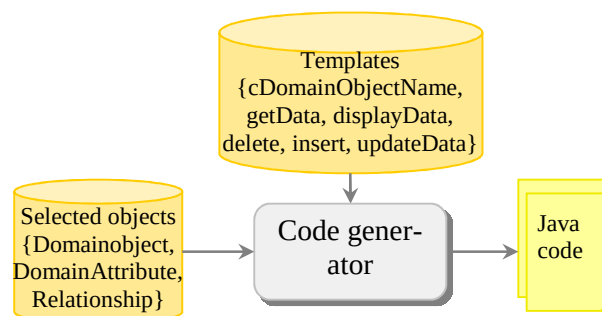


Figure 5-18: Application code generation process

Since the functions are defined, the objects, the attributes and the controls are also defined but we don't know what they are exactly in the domain model. They are only determined when the designer makes the links between the tasks and components of domain model. So in this step, we need to provide the elements which associate with the methods of the function. These elements are the linked attributes, the linked domain's objects, the relationships between these objects. The process of application code generation is depicted in Figure 5-18.

Two points are important for the generation of the application code. The first one is to generate the query strings to select, delete or insert data into the database. These queries are built based on the standard SQL statements. The second one is how to get the values from the controls and to set the values to the controls correctly.

The inputted objects are structured as depicted in Figure 5-19.

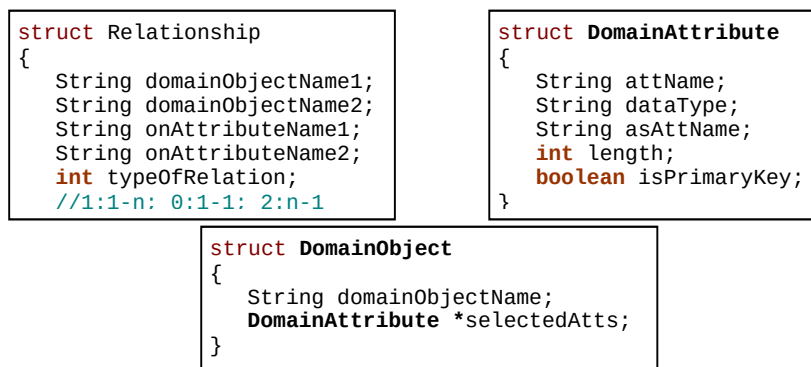


Figure 5-19: Inputted objects

```
DomainObject
{
    domainObjectName = "Jobs";
    selectedAtts = {
        {"Job_ID", "String", 20, true, false},
        {"Jobtitle", "String", 30, false, true},
        {"Maxsalary", "int", 15, false, false},
        {"Minsalary", "int", 15, false, false},
    };
    mainAttribute = "Jobtitle";
}
```

Example 5-10: Domain object- example

As discussed in previous section, in order to store the data records retrieved from the database, a template is defined in Figure 5-20

```

class cDomainObjectName
{
    jDataType asAttName1;
    jDataType asAttName2;
    ...
    public void set asAttName1(jDataType value)
    {
        asAttName1 = value;
    }
    public jDataType get asAttName1()
    {
        Return asAttName1;
    }
    ...
}

```

Figure 5-20: Template for storing the data records

```

public class cjobs{
    private String JOB_ID = "";
    private String Jobtitle = "";
    private int Maxsalary = 0;
    private int Minsalary = 0;

    public void setJOB_ID(String value){
        JOB_ID = value;
    }
    public String getJOB_ID(){
        return JOB_ID;
    }
    public void setJobtitle(String value){
        Jobtitle = value;
    }
    public String getJobtitle(){
        return Jobtitle;
    }
    public void setMaxsalary(int value){
        Maxsalary = value;
    }
    public int getMaxsalary(){
        return Maxsalary;
    }
    public void setMinsalary(int value){
        Minsalary = value;
    }
    public int getMinsalary(){
        return Minsalary;
    }
}

```

Delete function template

The template of *Delete* function is depicted in Figure 5-21.

attName is the primary attribute of the edited object. If this object doesn't define the primary attribute then it should consider all of the linked attributes of the edited object. control1 is the control created from attribute1.

```
private void Delete(String objectName, String attName, String value)
{
    try{
        s.executeQuery("DELETE " + objectName +
            " WHERE " + attName + " = " + value);
        showMessage("Data is deleted successfully");
    }
    catch (Exception e){
        showErrorMessage("An error is occurred ");
        System.out.println(e.toString());
    }
}
```

Figure 5-21: Delete function template

It is important to specify which data records you want to delete based on the primary attribute and its value or the other attributes, otherwise we could delete the wrong data records.

The **executeQuery()** method returns the True value if it is successful, otherwise the False value is returned. The **Delete()** function will display a message dialog to announce the result to the users.

AddNew function template

The template of *AddNew* function is depicted in Figure 5-22.

In this case, *do.getDomainObjectName()* is the name of edited object; the *do.getSelectedAtts().getAttribute1().getName()*, *do.getSelectedAtts().getAttribute2().getName()*... are all of the attributes linked to the tasks of the edited object; the *control1*, *control2*, and so on, are the controls created from the attributes *attribute1*, *attribute2*...

```
private void AddNew(DomainObject do)
{
    if (!checkNullValues())
    {
        showMessage("Can not insert NULL into this object");
        return;
    }
    try{
        s.executeQuery("INSERT INTO " + do.getDomainObjectName() + "(" +
            do.getSelectedAtts().getAttribute1().getName() + "," + ... + ")"
            " VALUES (" + control1.getValue() + "," + ... + ")");
        showMessage("One row is successfully inserted");
    }
    catch (Exception e){
        showErrorMessage("Cannot insert");
        System.out.println(e.toString());
    }
}
```

Figure 5-22: AddNew function template

In order to successfully insert a new record into the database, we need to be sure that the *checkNullValues()* function returns true value. This means that the required attribute are completely entered data and that the value of the primary attributes is not a duplicate of the value of the primary attributes of the existing records.

Update function template

The template of *Update* function is depicted in Figure 5-23.

```
private void Update(DomainObject do)
{
    if (!isModified())
    {
        showMessage("Nothing is changed");
        return;
    }
    try{
        s.executeQuery("UPDATE " + do.getDomainObjectName() + " " +
            "SET " + do.getSelectedAtts().getAttribute1().getName() +
            " = " + control1.getValue() + "," + ...
            "WHERE " + getPrimaryKeyName() = getPrimaryKeyValue();
        showMessage("Data is successfully updated");
    }
    catch (Exception e){
        showErrorMessage("Fail");
        System.out.println(e.toString());
    }
}
```

Figure 5-23: Update function template

As for the parameters of *AddNew* function template, *do.getDomainObjectName()* is the name of edited object; the *do.getSelectedAtts().getAttribute1().getName()*, *do.getSelectedAtts().getAttribute2().getName()*... are all of the attributes linked to the tasks of the edited object; the *control1*, *control2*, and so on, are the controls created from the attributes *attribute1*, *attribute2* ...

Before executing the update SQL, it is necessary to check if the data is modified by calling the *isModified()* method. DB-USE specifies the updated record based on the primary key.

Search - Display - Review function templates

We can discuss the three functions *Search()*, *Display()* and *Review()* at the same time. The difference between these functions is that the data is filtered differently. This means that the *WHERE* clause is generated differently for each function. Besides, these functions have a the same objective, that is to display the record at index *N* to the end-user. So we generate a method *displayData(Vector vRecords, int index)* to display a record at index *N*.

We suppose that *O1*, *O2*, *O3* ...*On* are the linked domain objects; *Att1*, *Att2*, *Att3* ... *Attn* are the linked attributes of these objects; *Control1*, *Control2*, *Control3* ... *Controln* are the control created based on the linked attributes and objects; and

O1 is edited object. Search(), Display() and Review() functions are performed by these following templates.

The *getData* template is used to retrieve data from the database. This template is depicted in Figure 5-24a.

```
private void getData()
{
    vRecords = new Vector();
    try{
        rs = s.execute("SELECT" + O1.Attr1.getName() + "," + O1.Attr2.getName() +
            "," + ... On.Attr1.getName() + "," + On.Attr2.getName()..
            "FROM " + O1.getName() + "," + O2.getName() + "," + ...
            "WHERE " + O1.getForeignKey1() = O2.getPrimarykey () +
            " AND " + O1.getForeignKey2() = O3.getPrimarykey () + ...

        while (rs.next())
        {
            cMainRecords temp = new cMainRecords();
            temp.setAttribute1(rs.getString(Attribute1));
            temp.setAttribute2(rs.getString(Attribute2));
            ...
            vRecords.add( temp ); // store data recosets in vector vRecords
        }
    }
    catch (Exception e){
        showErrorMessage("There is an error occurred");
        System.out.println(e.toString());
    }
}
```

Figure 5-24a: getData template

```
private void displayData(Vector vRecords, int index)
{
    if ((index < 0) || (vRecords.size() == 0)) return;
    if (Control1.controlType == Combobox/ListBox)
        Control1.setSelectedItem(vRecords.elementAt(index).getValue());
    if (Control1.controlType == Testfield)
        Control1.setText(vRecords.elementAt(index).getValue());
    if (Control1.controlType == CheckBox/RadioButton)
        Control1.setState(vRecords.elementAt(index).getValue());
}
```

Figure 5-24b: displayData template

As discussed above, the displayData template is used to display data record N to end-user. (See Figure 5-24b)

Default function template

Default function is a special function. The goal of this function is not specified so that the generated code for this function is the prototype. The name of this function is the name of the linked task. The template of this function is depicted in Figure 5-25.

```
private void functionname()
{
    // The developer has to write the code for this function
}
```

Figure 5-25: displayData template

DB-USE also produces the application messages defined in Table 4-5. Based on this table, we have four types of message; the different message types are represented by different icons. Example 5-11 shows you four different message dialogs which represent four different types of message.



Example 5-11: Message dialog boxes (error, confirmation, warning, and information)

5.1.3.3 How to generate code in C or Visual Basic

As discussed, the disadvantage of template-based technique is that in order to generate code for the different programming languages, the developer has to create the different sets of template. So that if DB-USE wants to generate code in other languages such as C++, Visual Basic then its developer has to create the templates which suit to the syntax and semantic of these languages.

Typically, the number of created templates is the same for programming languages. In DB-USE, the templates are not reused by most different programming languages. For example, to generate code for a text-field control in C++ and Visual Basic, the developer of DB-USE has to create two different templates for two these languages, he/she cannot reuse templates which are created for Java language.

```
String layoutCTextField {
    TextBox lname = gcnw TextBox;
    lname->Name = label;
    lname->Width = w;
    lname->Height = h; }
```

Case study

```
String layoutVBTextField {  
    lname As System.Windows.Forms.TextBox  
    Me.lname.Name = label  
    Me.lname = New System.Windows.Forms.TextBox  
    Me.lname.Location = p;  
    Me.lname.Size = d }  
}
```

Example 5-12: Templates for generating a text-field in C++ and Visual Basic

Example 5-12 depicts the templates which are used to generate code in C++ and Visual Basic for a text-field.

5.2 Case study

In order to validate the DB-USE tool which is built based on the architecture, the methodology, and the agents discussed in Chapter 4, we use two case studies including Human Resource Management and TransLogisTIC [Waute10a, Waute10b]. Human Resource Management case study is not a big one, but most of its tasks can be performed by DB-USE tool. Different from Human Resource Management, TransLogisTIC is a big and complex application. Besides, its tasks are the ones that are implemented in terms of the graphic objects such as line, icons, and rectangle objects (See Figure 5-26); while DB-USE tool doesn't support generation of these objects. This means that we have chosen case studies which are positive, negative, simple, and complex.

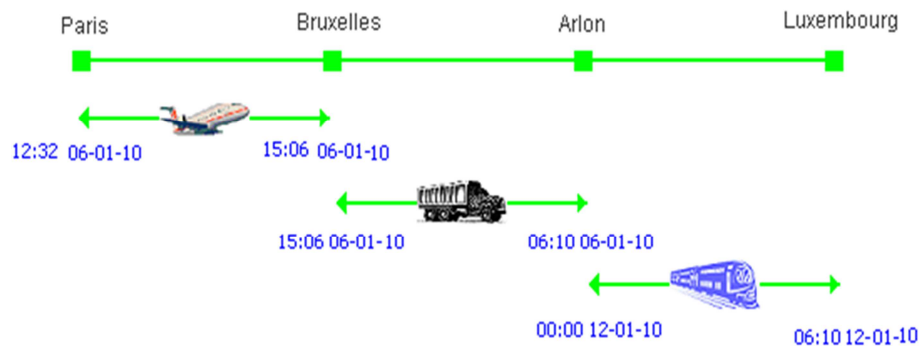


Figure 5-26: Treat Orders Services

Figure 5-26 describes a transportation of merchandise from Paris to Luxembourg. It is executed by three kinds of means of transportation: the cargo is transported from Paris to Bruxelles by plane, from Bruxelles to Arlon by truck, and from Arlon to Luxembourg by train.

In each case study, we start with description of the task and domain models. Then some important tasks in the task model are chosen to generate the user interface code and application code.

5.2.1 Human Resource Management

We have chosen human resource management as a first example for validation of our DB-USE tool because human resource management is used by most large companies; it is easy to understand, and specially it is a database application and has been developed and implemented by using Oracle, SAP.

Human Resource Management (HRM) function includes a variety of activities. These activities are recruitment of employees, management of employees, training of employees and dismissal of employees. Activities also include dealing with issues related to employee such as compensation, hiring, performance management, organization development, safety, wellness, benefits, employee motivation, communication, administration, and training.

HRM is responsible for how employees are treated in an organization. It is responsible for bringing employees into the organization, helping them perform their work, compensating them for labors, and solving problems that arise. HRM works to ensure that employees are able to meet the organization's goals.

We understand that HRM is necessary in every company and employee management plays an important role in HRM. Besides, managing employees is a major job. In this thesis, we specially focus on management of employees.

Computer can help the manager to manage their employees from recruitment to exit. The manager can create a new employee, review or modify the information of an existing employee, delete an existing employee, search employees, and make a report of employees.

The task model and domain model of HRM are discussed in Section 1.1 and 1.2 of appendix. Once the models have been loaded, the designer can start to make the links between the tasks and domain's attributes and the function defined in the domain model in order to generate the user interface code and the application code. *Edit Employees* and *Edit Job Information* tasks are selected to perform in this section.

5.2.1.1 Edit Employees Task

The *Edit Employees* task is depicted in the Figure 5-27. The task's goal is to allow the end-user to review the employee information, create a new employee, delete an existing one, and modify the employee information.

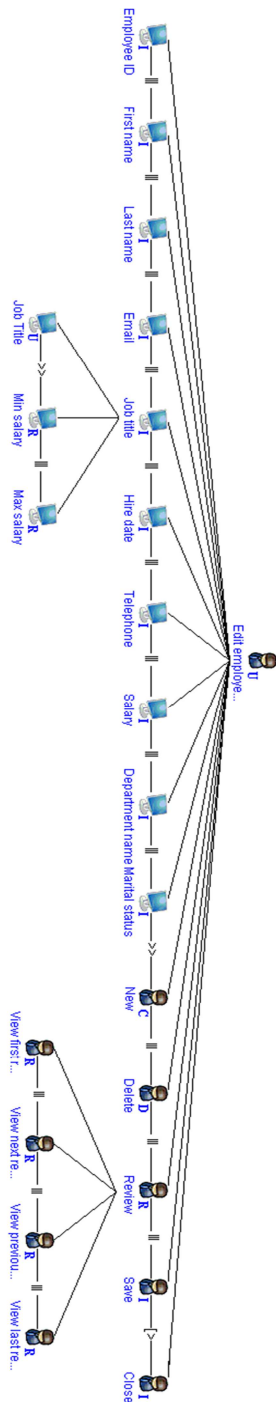


Figure 5-27: Edit employees task

In this case the *Employees* object is also determined to be the edited object.

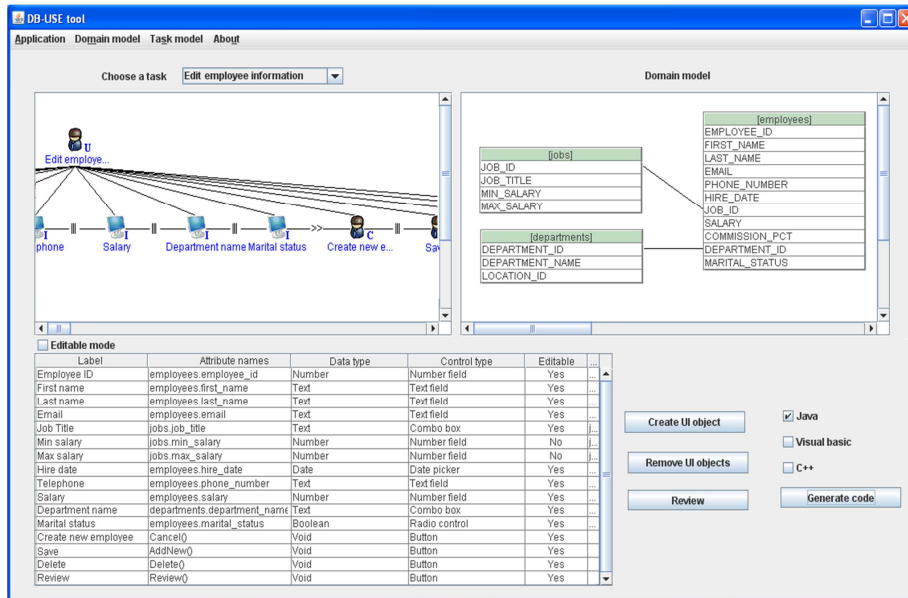


Figure 5-28: Creating the UI for *Edit employee information* Task

The designer can start with linking the tasks to the domain's attributes; these links are depicted in Figure 5-28. The *Employee id*, *First name*, *Last name*, *Email*, *Hire date*, *Telephone*, *Salary*, *Marital status* tasks are linked to *Employee_ID*, *First_Name*, *Last_Name*, *Email*, *Hire_Date*, *Telephone_Number*, *Salary*, *Marital_Status* attributes of the *Employees* object. The *Create new employee* task is linked to *cancel()* function to reset the control to empty; the *Save* task is linked to *AddNew()* function to insert data to the database; the *Delete* task is linked to *Delete()* function to delete the current record; the *Review* task is linked to *Review()* function to review the first, previous, next and last records; the *Close* task is linked to *Exit()* function to close the form.

Once the generated code has been compiled and ran without modifying, the *Edit employee information* form is displayed in Figure 5-29.

With this form, the end-user can review the information of the employees, delete an existing employee, or create a new one. In order to create a new employee, the end-user needs to click on the *New* button to reset the control to empty; then he/she has to enter the data for each control in the form; finally, he/she can insert it into the database by clicking on the *Update* button.

Edit employee information

Employee ID	101
First name	Neena
Last name	Kochhar
Email	NKOCHHAR
Job title	Administration Vice President
Max salary	15000
Min salary	30000
Hire date	1989-09-21 00:00:00.0
Telephone	515.123.4568
Salary	17000
Department name	Executive
Marital status	☒ Yes ☐ No

First Prev Next Last

New Del Update Exit

Figure 5-29: Edit employee information Form

5.2.1.2 Edit job information Task

The goal of this task is to allow the end-user to create a new job, delete an existing one, modify the information of the job, and verify the name, email address, telephone number of the employees of the current displayed job.

The *Edit job information* task is depicted in Figure 5-30.

The *Jobs* object is the edited object. Once the *Edit job information* task has been chosen at combo box *Choose a task* and the *Jobs*, *Employees* objects have been selected from the domain model. The designer can start to make the links between the tasks and the domain's components.

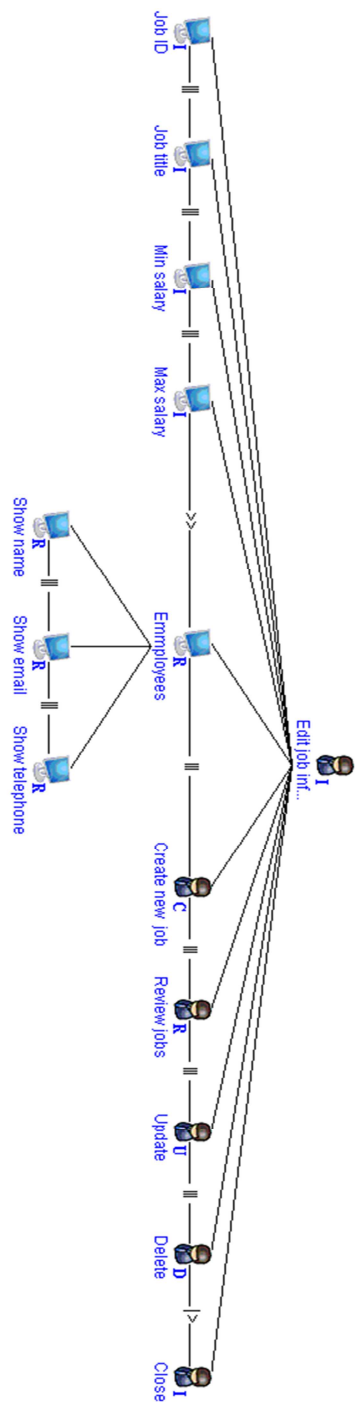


Figure 5-30: Edit job information Task

Case study

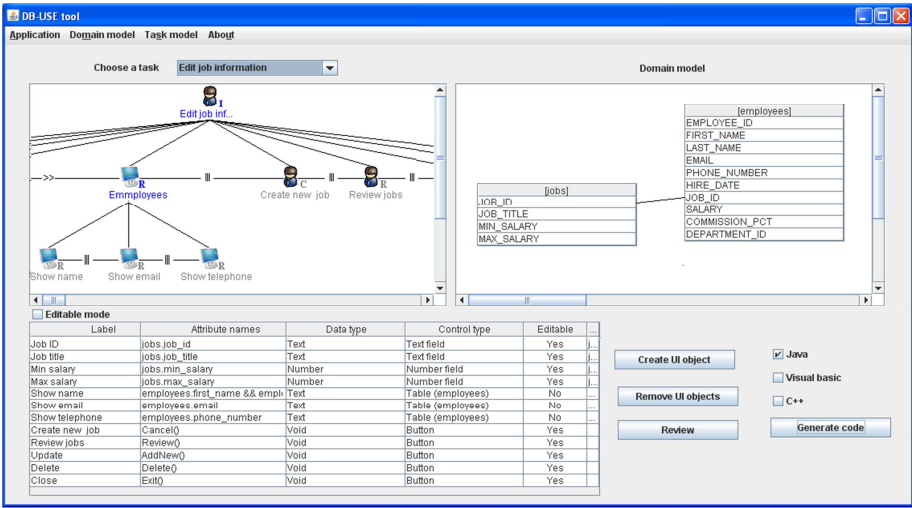


Figure 5-31: Creating the UI for *Edit job information* Task

The *Job ID*, *Job title*, *Min salary*, *Max salary* tasks are linked to the *Job_ID*, *Job_Title*, *Min_Salary*, *Max_Salary* attributes of *Jobs* object; the *Show name* task is linked to *Fist_name* and *Last_Name* attributes of the *Employees* object; The *Show email*, *Show telephone* tasks are linked to *Email* and *Phone_Number* attributes of the *Employees* object; the *Create new job* task is linked to the *cancel()* function; the *Update* task is linked to the *Update()* function; the *Review jobs* task is linked to the *Review()* function; the *Delete* task is linked to the *Delete()* function; finally, the *Close* task is linked to the *Exit()* function. These links are depicted in Figure 5-31. The relationship between *Jobs* and *Employees* objects is *1-N*, so the control type of Table is determined for displaying the data of the *Employees* object.

The screenshot shows the 'Edit job information' form. It has four input fields: 'Job ID' (containing 'FI_ACCOUNT'), 'Job title' (containing 'Accountant'), 'Min salary' (containing '4200'), and 'Max salary' (containing '9000'). Below these is a table titled 'Employee information' with three columns: 'Show name', 'Show email', and 'Show telephone'. The table contains five rows of employee data. At the bottom of the form are two rows of buttons: 'First', 'Prev', 'Next', 'Last' and 'New', 'Del', 'Update', 'Exit'.

Show name	Show email	Show telephone
Daniel Faviat	DFAVIET	515.124.4169
John Chen	JCHEN	515.124.4269
Ismael Sciarra	ISCIARRA	515.124.4369
Jose Manuel Ur...	JMURMAN	515.124.4469
Luis Popp	LPOPP	515.124.4567

Figure 5-32: Edit job information Form

Figure 5-32 depicts the *Edit job information* form. This form is the result of executing the code generated by the previous step; this code is not modified by the developer.

In order to add a new job, the end-user has to provide the job id, job title, max and min salary. Once the information of the current job has been changed; and the end-user moves to another job or creates a new job then this change is automatically updated into the database. The information of the employees cannot be modified in this form.

5.2.2 TransLogisTIC

A supply chain is the set of all actors and relations between them which participate in the process of delivering value to a customer as a product or service. It includes all the processes from raw materials to delivery and is viewed as a network of information, materials and financial flows. In this case study we particularly focus on the application that relates to the movement and storage of products and goods from the supplier to the end-user - with a strong focus on transportation. (See Section 2 of appendix).

To depict the UI creation of TransLogisTIC, we select Create a shipment order and Perform shipment order tasks. These tasks are the ones that are used a lot in this application and their interfaces are also complicated enough.

5.2.2.1 Create a shipment order Task

Once the shipper has received an order from his/her client, he/she creates a shipment order based on this order. A shipment order is represented by shipment ID, order number, description (shipment name), shipping address, delivery address, shipping date, delivery date, volume, status, and transporter who will transport the merchandise. There are four different statuses used to describe a shipment order including wait, sent, accepted, refused. When a shipment order is created its status is wait; if it is sent to the transporter then its status is sent; if it is accepted by the transporter then its status is accepted; if it is refused by the transporter then its status is refused. Once the transporter has been determined, the shipper can send this shipment order to him and wait for his confirmation.

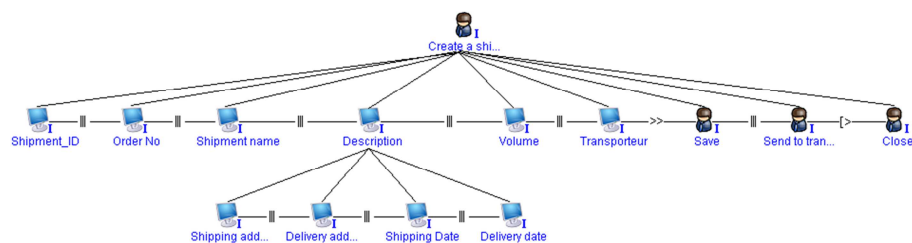


Figure 5-33: Create a shipment order task

Case study

The Create a shipment order task is created in Figure 5-33.

The objects involved in the creation of user interface of Create a shipment order task are T_Appel_A_Transport (is used to describe the shipment orders), T_Noeud (is used to describe the addresses), T_Utilisateur (is used to describe the Transporters), T_Command (is used to describe the orders of the clients) objects. In this case, the T_Appel_A_Transport object is chosen as an edited object.

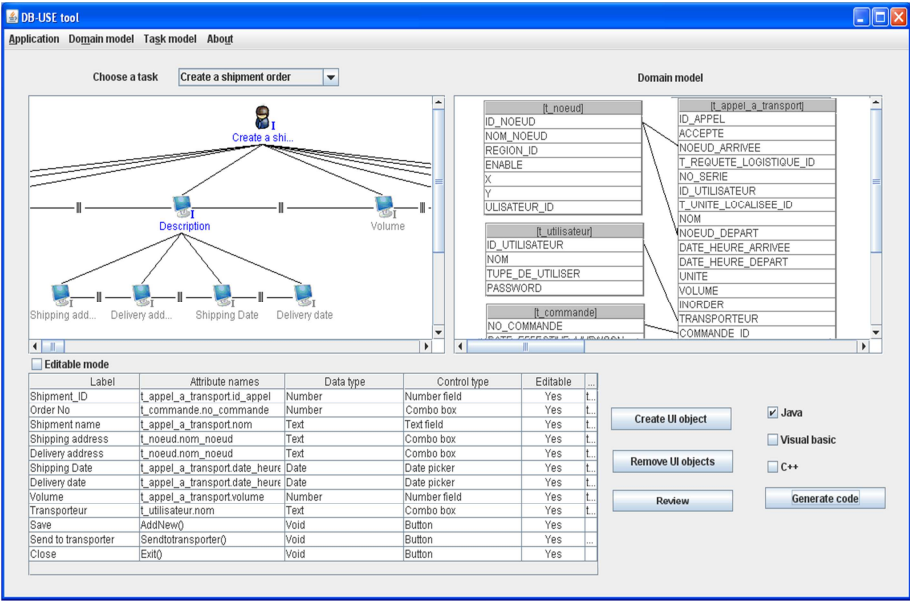


Figure 5-34: Generating the UI for Create a shipment order task

The Shipment_ID, Shipment name, Shipping date, Delivery Date, Volume tasks are linked to ID_Appel, Nom, Date_Heure_Depart, Date_Heure_Arrivee attributes of the T_Appel_A_Transport object; the Shipping address, Delivery address task are linked to the Nom_Noeud attribute of the T_Noeud object; the Order No task is linked to the No_Commande attribute of T_Commande object; the Transporter task is linked to the Nom attribute of the T_Utilisateur object. The Save task is linked to the AddNew() function; the Send to transporter task is linked to Default() function; the Exit task is linked to Exit() function. These links are described in Figure 5-34.

Figure 5-35 depicts the Create a shipment order form generated by DB-USE tool. In this form, the status of the shipment order is not displayed to the end-user but it is specified if the end-user clicked on the Save, Send to transporter button. The developer has to write the code to change the status in the database. The wait status is specified if the end-user clicked on the Save button; the sent status if the end-user clicked on the Send to transporter button. The accepted and refused statuses are determined in the transporter site.

Shipment_ID	No10257
Order No	No27635
Shipment name	Computer-Send to Fnac
Shipping address	Bruxelles
Delivery address	Charleroi
Shipping Date	10/08/2010
Delivery date	10/08/2010
Volume	20 pc
Transporteur	Transporteur 2

Save Send to transporter Exit

Figure 5-35: Create a shipment order form

5.2.2.2 Perform shipment order Task

Once the transporter has received a shipment order from the shipper, he/she selects the needed services among the created services to perform this shipment order. A shipment order may be performed by more than one service. For example, the transportation of computers from Paris to Luxembourg is performed by three services including service 1 (from Paris to Bruxelles by plane), service 2 (from Bruxelles to Arlon by truck), and service 3 (from Arlon to Luxembourg by train).

The Create a shipment order task is created as Figure 5-36.

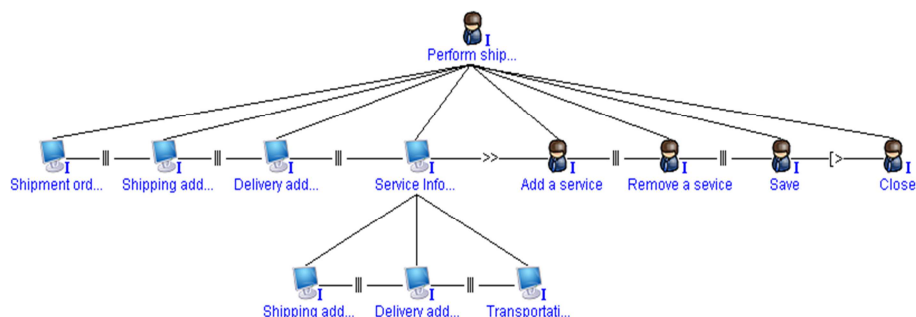


Figure 5-36: Perform shipment order task

The *Shipment Order ID* task is linked to *ID_Appel* attribute of the *T_Appel_A_Transport* object; the *Shipping address*, *Delivery address* task are linked to the *Nom_Noeud* attribute of the *T_Noeud* object; the *Transportation mode* task is

Case study

linked to the *Nom_Mode_Transport* attribute of *T_Mode_Transport* object. The *Add services*, *Remove services*, *Save* tasks are linked to *Default()* functions (the developer has to write the code to perform these functions); the *Exit* task is linked to *Exit()* function. The UI objects are automatically specified based on these links, see Figure 5-37.

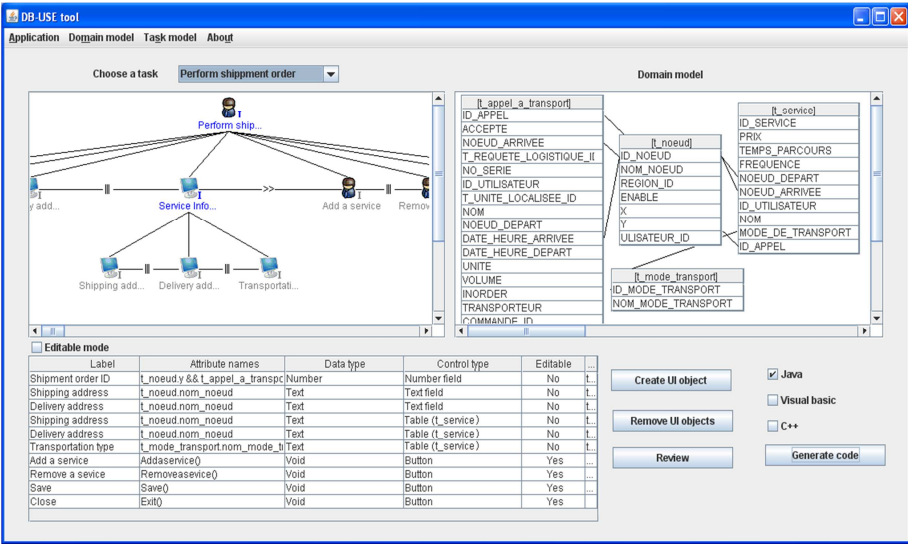


Figure 5-37: Generating the UI for Perform shipment order task

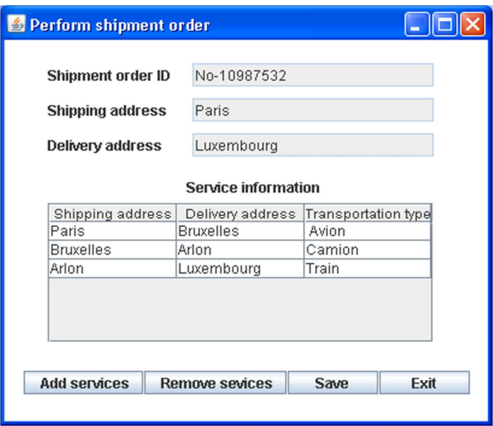


Figure 5-38: Perform shipment order form

In order to perform the functions of *Add services*, *Remove services*, and *Save* buttons, the developer has to write more code to insert or delete the service information into / from database.

5.2.3 Conclusion

Based on the result of the two case studies above, we note that it is more adequate to apply DB-USE for database applications, such as the Human Resource Management application, in which the DB tasks are the display of the information and the reception of the information. In the TranslogisTIC application, some task information can be better understood if they are displayed to the end-user in terms of graphics. For instance, the following ones.

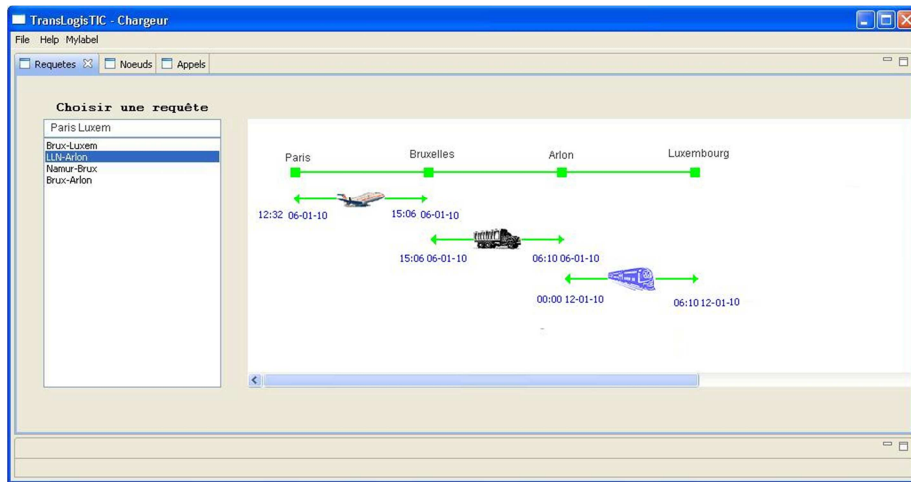


Figure 5-39: Shipment order form – graphic objects

Shipping address	Delivery address	Transportation type
Paris	Bruxelles	Plane
Bruxelles	Arlon	Truck
Arlon	Luxembourg	Train

Figure 5-40: Shipment order form – text object

Case study

The services of a shipment are displayed to the end-user in terms of graphic objects (See Figure 5-39) and of text object (See Figure 5-40).

The screenshot shows the 'TransLogistics - Transporteur' application window. The 'Services et Transports' tab is active, displaying a graphical representation of transport services. On the left, under 'Appel transports', there is a list of transport calls with details like 'Nom', 'Départ', and 'Arrivée'. On the right, under 'Services et Transports', there is a timeline diagram showing the route from LLN to Bruxelles to Wavre, with specific transport services (Transport 1_1, Transport 2, Brx-Wavre) and their respective departure and arrival times.

Figure 5-41: Service and Transport form – graphic objects

The screenshot shows the 'Services and transports' form in a text object view. It contains input fields for 'Transport call', 'From', 'To', 'Departure time', and 'Arrival time'. Below these fields is a table showing the transport services and their details.

From	To	Transport type	Time
Louvain La ...	Bruxelles	Train	10:30 12-10...
Bruxelles	Wavre	Plane	10:30 12-10...
Louvain La ...	Bruxelles	Truck	10:30 12-10...

At the bottom of the form, there are navigation buttons: 'First', 'Prev', 'Next', 'Last', 'New', 'Del', 'Update', and 'Exit'.

Figure 5-42: Service and Transport form – text object

Similar to the shipment information, Figures 5-41 and 5-42 depict information about service and transport in terms of graphic and text objects. With respect to the example shown in Figures 5-39 and 5-41, if the end-user wants to create a new service, transport or modify the existing ones he/she can click on the right mouse button to display a popup-menu then he/she can select options such as Create, Modify, Delete ...

In addition to discuss the two case studies, we would like to compare the advantages and disadvantages of DB-USE with respect to the current other approaches.

5.3 Validation

Section compares DB-USE with other researches including Just-UI [Moli02, Moli03], Vampire [Eise02] and the design patterns of Coad & Yourdon [Coad95] which have the same goal that DB-USE but the inputs they have used are different.

Like DB-USE, the works selected to compare with it aim to the database applications. Just-UI and Vampire aim to generate UIs for database applications while the design patterns of Coad & Yourdon are created to describe the tasks of database application.

The goal of Just-UI and Vampire is to generate UI but they have used the different materials and the different processes. Just-UI generates UIs based on the created patterns; Vampire has used the task model to generate UIs.

The design patterns of Coad & Yourdon are used in this section to value the functions created by DB-USE; we would like to show how DB-USE perform the patterns defined by Coad & Yourdon (which are frequently used in the area of information systems in order to structure data into a domain model with some rigor, thus leading to a UI that appropriately reflects the goal of the pattern) and otherwise how these patterns support generic functions of DB-USE.

The purpose of this section is to compare DB-USE with current works to validate DB-USE based on the input and output of these works.

5.3.1 Just-UI: using pattern to generate user interface

We first discuss the pros and cons of Just-UI [Moli02, Moli03] and DB-USE. The motivation behind both approaches is that the database applications have similar interfaces. But unlike DB-USE, Just-UI uses the pattern technique to generate the user interface. These Just-UI's patterns are specified based on the user's requirements; the interactions of user with an application. The patterns are expressed in terms of abstract interaction objects which will be translated to concrete interaction objects based on mapping rules. The user interface code is generated

Validation

based on the CIOs and the programming language. These patterns are classified as follows:

- *The Service presentation pattern* is created to perform a service specification based on the data inputted by user and maybe in order to return the data to user.
- *The Instance presentation pattern* is created to display data of an object required by user; it is also used to perform services such as modify data, create new data, delete existing one.
- *The Population presentation pattern* like the instance presentation pattern is used to display data of a set of objects, filter data, order data and perform services.
- *The Master-Detail presentation pattern* is created to display information under the form of master-detail objects. For instance, displaying information of a student (master object) and his courses (detail object).

The advantage of Just-UI is that the operations are performed quickly and simply. For instance, if the designer wants to create a form to display the information of the object employee, he/she only makes a link between this object and the *instance presentation* pattern while the designer has to make the links between the attributes of this object and the tasks in DB-USE. However Just-UI is not flexible for user interface creation. For instance, in order to display the information of the employees of a department, the designer has to link department objects to the master pattern and employee objects to the detail pattern. The limitation of Just-UI is that the designer cannot decide which attributes of a department object or of an employee object will be displayed to the end-user. Once the link has been created between a pattern and an object then all of the attributes of this object will be displayed to the end-users. This weakness has been resolved in DB-USE.

Just-UI also proposes translators for Visual Basic, Java, JSP, ASP and ColdFusion.

5.3.2 VAMPIRE: Agents and GUIs from task model

VAMPIRE [Eise02] uses collaborative interface agents to create user interface based on a task model. A collaborative interface agent is a software agent that supports the designer in user interface creation such as proposing control type, managing the links between tasks and user interface components. In VAMPIRE, the task model guides the designer to create the user interface with respect to the task's goal.

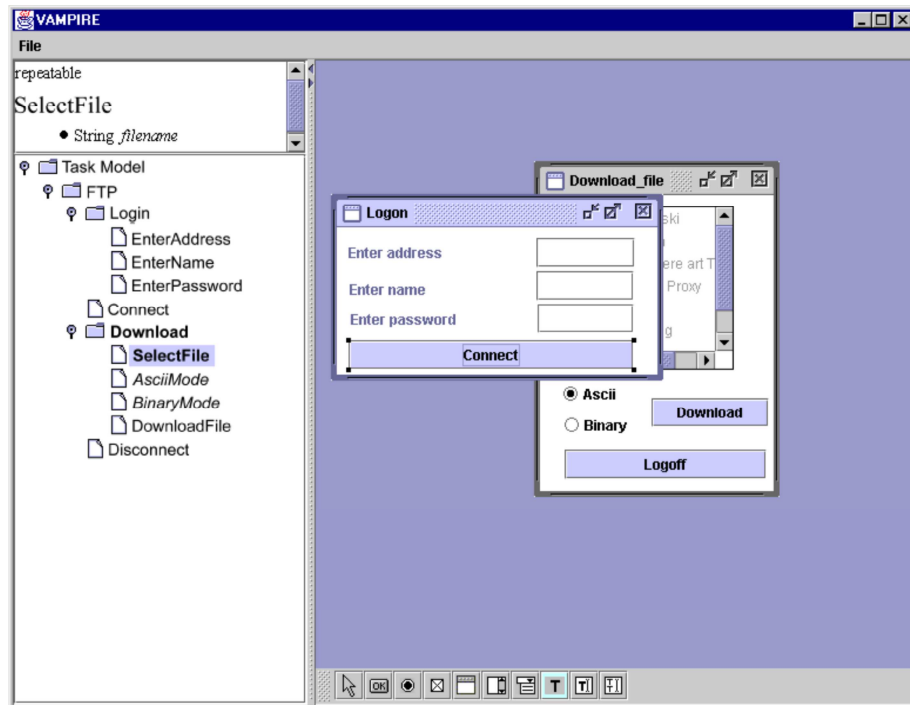


Figure 5-43: VAMPIRE tool (From [Eise02])

VAMPIRE is depicted in Figure 5-43. The process of user interface design starts with the designer selecting an unimplemented task in the task tree (displayed in the left panel); VAMPIRE recommends a control by highlighting this control in the toolbar (displayed at bottom of the screen). If the designer agrees with this recommendation, he/she can add this control to the form by clicking in the layout area where this control will be placed. Otherwise, he/she can ask for a second recommendation by clicking the task again.

In VAMPIRE, the designer has to make the links between tasks and the user interface components (controls) based on the recommendations of the system. While the designer makes the links between tasks and the domain's components and based on these links, the user interface components are automatically generated in DB-USE. Hence, the user interface generated in VAMPIRE depends on the decision of designer. Besides, the absence of the domain model in VAMPIRE implies that its user interface ignores the representation of the data and business described by this model.

5.3.3 Verify the coverage of conceptual Coad & Yourdon patterns by DB-Use

This section discusses of Coad & Yourdon transaction patterns [Coad95] used to build object model.

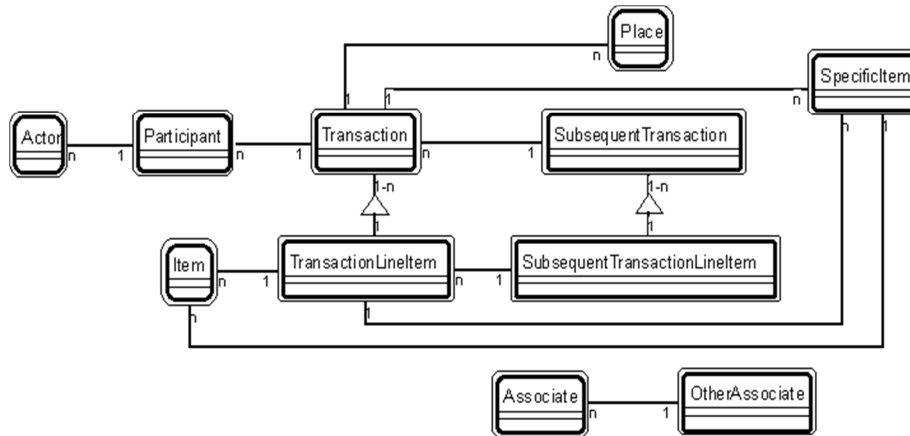
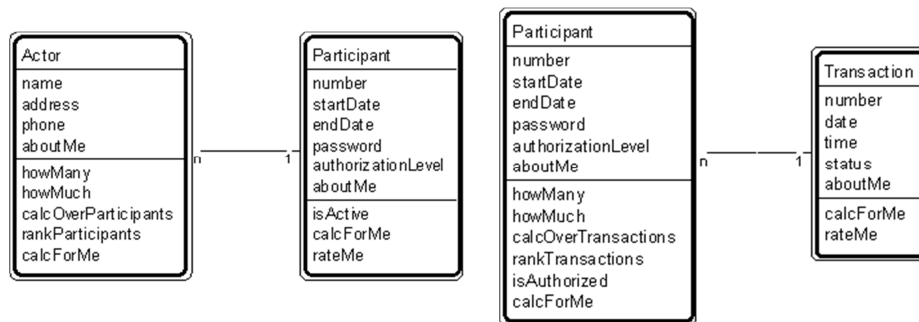


Figure 5-44: Design patterns for Transaction (From [Coad95])

Figure 5-44 overviews the main structure of transaction patterns organized into an object-oriented model. Coad & Yourdon recommend to refer to this structure in order to build a domain model since most of these design patterns are frequently used in the domain of information systems and in interactive systems in general. These transactions are Actor-Participant, Participant-Transaction, Place-Transaction, Specific Item – Transaction, Transaction - Transaction Line Item, Transaction - Subsequent Transaction, Transaction Line Item - Subsequent Transaction Line Item, Item - Line Item, Specific Item - Line Item, Item - Specific Item, Associate - Other Associate. The interconnections between these patterns are represented by the lines.

We select Actor-Participant and Participant- to verify the performance of the functions covered by these patterns in DB-USE. They are described as follows:



- Actor can be person and organization (agency, company, corporation and department).
- Participant can be agent, client, employee, student and buyer.
- Transaction can be agreement, assignment, contract, delivery, order, payment, report, purchase, refund, shipment.

We separately discuss in one hand the capabilities of DB-USE in performing pattern's functions and in the other hand DB-USE's functions that have not yet been discussed in patterns.

Coad & Yourdon	DB-USE	Note
howMany		DB-USE does not support the counting of the participants of an actor. But the Display() function of DB-USE can show the user all of participants of an actor if these two objects are linked. (Figure 5-44 lists all the transports of a service)
howMuch		Not supported
calcOverParticipants/ calcOverTransactions		Not supported
rankParticipants/ rankTransactions	Display()	To rank the participants/transactions (for example based on attribute startDate), the user can use the Display() function with option <i>order by</i> (startDate).
calcForMe	Display()	With the display() function of DB-USE, the user can sum value of the different attributes (such as salary and bonus attributes of employees objects)
isActive	Search()	To look for the participants who are active, you can use the Search(attribute endDate) function.
rateMe		Not supported
isAuthorized	Search()	To look for the participants who are authorized, you can use the Search(attribute authorizationLevel) function.

Table 5-3: Pattern's functions and performing them in DB-USE

Table 5-3 shows the functions defined in the Actor-Participant and Participant-Transaction patterns and which functions of DB-USE are used to perform them. Most of the calculation functions such as howMuch, rateMe are not supported by DB-USE because they consist of calculate functions.

Table 5-4 lists the functions of DB-USE and related pattern. It shows that most functions in DB-USE have not been discussed in Coad & Yourdon's design patterns. Unlike the functions of Coad & Yourdon's patterns, DB-USE's ones are usually used to retrieve or modify data in database.

Chapter Summary

DB-USE	Coad & Yourdon	Note
Display()	rankParticipants/ rankTransactions	rankParticipants / rankTransactions is used to display data in descending/ascending order.
Search()	isActive/ isAuthor- ized	isActive / isAuthorized is used to search the active/authorized objects.
AddNew()		Not supported
Delete()		Not supported
Update()		Not supported
Review()		Not supported

Table 5-4: Functions of DB-USE and related pattern's functions

5.4 Chapter Summary

This chapter has presented a DB-USE tool. This tool has applied for two case studies, including Human Resource Management and TransLogisTIC. Besides, to validate the abilities of DB-USE we also compare it with Just-UI, VAMPIRE and patterns of Coad & Yourdon. It explained that most of the design patterns from Coad & Yourdon could be indeed incorporated into a domain model made by DB-Use, that is in turn used together with other models, to semi-automatically generate corresponding UIs. Coad & Yourdon do not provide any guidance on how to produce such UIs, while DB-Use provide some starting point.

Chapter 6

Conclusion

This conclusion presents a summary of DB-USE and its benefits. Section 6.1 summarizes the main ideas of our work. The expected benefits are emphasized in Section 6.2. Section 6.3 lists some future works.

6.1 Original contribution of this thesis

To be efficient, data-intensive systems that are an important component of today's software applications need effective human-computer interaction. User interfaces for such data systems have been a recurrent research issue and nowadays these UIs have to support automatic generation to adequately be dealt with.

We have proposed in this thesis a methodological framework called DB-USE for developing user interfaces to database applications from the task, domain and user models combined together. More specifically, this framework consists of:

- A meta-model that governs the semantics of the task, domain and user models (Section 3.1);
- A methodological process that supports the UI designer/developer to create the UIs semi-automatically from the above models (Section 4.1.4);
- A software system supporting the methodological process in order to generate both the user interface code and the application code that performs the basic functions of the database application (Chapter 5).

Our framework is aimed at offering a low cost, short time-to-implementation and efficient development environment from producing a user interface to this database system. Indeed, the objective is not to only generate the UI code but also the application code for performing generic functions of a database application such as AddNew(), Search(), Delete(), Update().

The methodological process of DB-USE (Section 4.1.4) has been established based on lessons learnt from existing researches such as Teallach, Goliath, and Wold (Section 2.2). The advantages come from the contribution of declarative models in UI generation (Section 1.1.2), the general way to create the UI objects, the usefulness coming from UI generation, the relation between task and domain models. The disadvantages come from the limit of mapping rules, the missing of end-user model which is used to create the UIs which consider the preference of the end-user (Section 2.4). In order to improve this process, task, domain, and user models have been considered together as opposed to separately in existing work.

Original contribution of this thesis

The methodological process of DB-USE (Section 4.1.4) is divided into five phases depending on the role of participants and tasks that they perform.

- *Model analysis phase:* The Model analyst is responsible for this phase. DB-USE's models including task, domain and user models are loaded from the existing resources.
- *Relation making phase:* Once these models have been loaded, the UI Designer makes links between the tasks in task model and the elements in domain model.
- *UI design phase:* Based on these links, the CIOs (Concrete Interaction Object) are automatically created by DB-USE system. Since the programming language has been determined by the UI Designer, the CIOs are translated to FIOs (Final Interaction Object).
- *Application function design phase:* Once Phase 3 has been completed, the functions linked in phase 2 are implemented in more detail based on the created FIOs by DB-USE system. The Developer is responsible this phase.
- *Code generation phase:* Finally, the UI code and application code are automatically generated based on the result of UI design and application function design phases. The role of the Developer in this phase is to verify generated code to be sure that it is successfully compiled. Typically, the generated code is clear and complete enough to be compiled and executed immediately.

The methodological process of DB-USE has been detailed using a multi-agent approach (Section 4.2) in order to support flexibility, but also to ensure clear separation of responsibilities of these agents, as opposed to monolithic process as in existing work. A multi-agent framework has been applied to the DB-USE process since MAS provide the required characteristics to describe it. Each component in the DB-USE architecture (Section 2.1.2) plays a role of one agent in multi-agent system (Section 4.2).

This process has been discussed step by step, in each step, besides providing the technique for loading DB-USE's models or generating UIs, rules for translating the task model from CTT to DB-USE or for determining UI objects, we have also provided concrete examples to illustrate the result of the work done in the considered step. The task, domain and user models have been used to create the UIs for the following reasons:

- The task model prescribes specifications that drive the production of the abstract user interface.
- The domain model provides the special features needed for creating a UI manipulating the information contained in the domain model and for specifying the methods to be performed on this information.
- The user model supports the creation of user interfaces which consider the preferences of the users.

In this research, we have tried to capitalize on existing work and resources as much as possible. Typically, the DB-USE task model (Section 3.1.1.2) can be loaded from an existing CTT task model (Section 4.2.2.1_i) while the Domain model (Section 3.1.2) is created based on a database schema (Section 4.2.2.1_ii). DB-USE's task model has been particularly defined differently from the other ones since we have considered it to be used for generating UIs at the implementation. In addition, to support the designer to understand the task's goal clearly, the action type of tasks have been also specified (See Table 3-5).

Our research work has also contributed to define rules for creating the UI objects based on the associations between tasks in task model and elements in domains model. They are the rules RTMs, RSTs, RLIs, RLAs, RSDs, RGAs, and RDCs (Section 4.2). These rules have been used to translate the task model from CTT version to DB-USE version, or to create the UI objects. Especially, RDCs have been summarized in Table 4-3.

The DB-USE tool has been implemented in Java (Section 5.1). It allows the designer to perform most of the goals discussed in our methodological process. DB-USE has been specifically designed to help the designer/developer to work with it easily and effectively. Task and Domain models are displayed in terms of graphics; and most operations can be performed by using the mouse. In addition, the DB-USE's task model cannot only be loaded from the CTT's task model but can also be newly created by the designer (Section 5.1.1.1). In this context, we have also provided algorithms for editing task trees such as how to position tasks in the tree when a task is deleted or inserted (Section Algorithms).

The result of this research work has been applied through two real-world case studies of two different sizes: one about Human Resources Management (Section 5.2.1), the other about the TransLogisTIC project (Section 5.2.2). The lessons learnt from applying this methodological process on these two case studies as well as the comparison between DB-USE with respect to significant related work (e.g. such as VAMPIRE or Just_UI) are provided in Section 5.3.

6.2 Benefits and shortcomings

In this section, we resume some benefits and shortcomings of DB-USE.

6.2.1 Potential Benefits

The potential benefits of DB-USE come from the following aspects:

- The task model of DB-USE is specifically defined for generating UIs. Although this model is defined based on representative task models found in the literature - for example it can be loaded from the CTT's task model -, the generated UI highly depends on the expressiveness of this task mode. The current version of the tool proposes two types of tasks, but the

Benefits and shortcomings

designer/developer can define new task types that reflect the flexibility of the DB-USE's task model. But this task model does not support multiple users or multiple contexts of use.

- The domain objects of DB-USE's Domain model are compatible with the ones of existing domain models. However, the functions defined in DB-USE are not the same. They are provided by the DB-USE system itself while in the other current domain models, such functions have to be created by the designer/developer. The number of the functions can be decided by the DB-USE's developer. The current version supports nine typical functions including Display(), AddNew(), Update(), Delete(), Search(), Review(), Cancel(), Exit(), and Default() which are necessary for a database application. This function set can be augmented in the future with generic functions depending on requirements. In addition, based on concrete contexts, the DB-USE system can generate detailed functions; for example, if the Display() function is selected for displaying the information of the employees, DB-USE can generate automatically the DisplayEmployees() function which describes the attributes of an employee.
- The methodology of DB-USE has provided a new process for generating both the user interface code and application code together based on declarative models. With this new process, the amount of work of the designer/developer is decreased significantly and the complexity of linking the UI code to the functional core is reduced. Moreover, the generated code is complete in that it can be run immediately without any further modification.
- In order to reuse the generated UIs in different platforms and programming languages, DB-USE allows the designer/developer to store them in terms of UsiXML specifications.
- DB-USE has also provided the model-based user interface development environment with a set of mapping rules which are used to specify the concrete interaction objects based on the associations between the task and domain models. The algorithms used for working with the task tree have been also provided in this thesis.
- Our research work has provided a complete tool for generating UIs from task and domain models. The goal of DB-USE tool is not only to validate the result of research but also to be applied to actual working environments. It then results to be very helpful for implementing database applications.

6.2.2 Shortcomings

Based on the case studies and comparisons between the DB-USE approach and the other ones, we can point out the following shortcomings:

Conclusion

- DB-USE has only allowed the designer/developer to create the user interfaces which relate to the domain model. That is a limitation of DB-USE in UI creation since in practice, the information does not only come from the database but also from the system or the network e.g., as the information of printer driver. Therefore, there is a need for improvement here.
- In the current version of DB-USE, the generated user interfaces are the dialogues and they are generated separately. These dialogues are conceptually related to each other. In order to do that, the designer/developer has to link them together.
- The user model has been discussed on how it is considered in the methodological process, but requires further investigation on how to determine the influence of user parameters (such as preference versus performance) on the generated UIs. Today, we do not have conducted any experimental
- The user interfaces generated by DB-USE are typically for database applications. This research was not focused on applications for other types of applications, hardware or devices yet such as hand phone or palm while these devices are being used more and more in our modern world.
- Most functions supported by DB-USE have been used for displaying information to the end-user and receiving the information from the end-users. Calculation functions have not yet been discussed such as sum(), multiple(), subtract() while these are necessary for implementing the database applications such as calculating salary, prices, rates, ...
- The domain model contains a number of domain objects while the designer/developer just needs some of them for generating the user interface at once. The question to help him/her to select the corrected domain objects is until now still open. However, in this version, if the edited object (Section 4.2.2.3) is specified by the designer/developer then DB-USE system will show him/her the related objects.

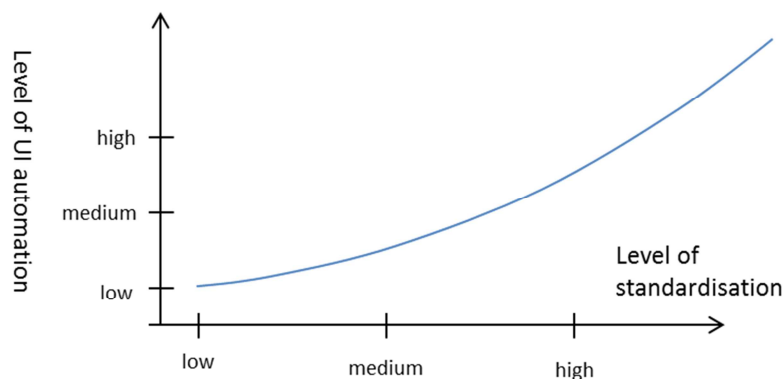


Figure 6-1: Level of automation vs. Level of standardization

Future work

It has been observed (Figure 6-1) that the more standard the generic functions are defined in the domain model, such as, but not limited to, Display(), Read(), etc., the more automatic the corresponding UI generation can take place. This is an advantage, but also a shortcoming: a good UI generation process may take place when the function is described with a level of expressiveness that is high enough to generate a usable UI. Without this level of description, only a very basic UI could be instead generated. Therefore, it is possible to expand the expressiveness of functions in order to improve the quality of the UI generation process, but this is implemented by the raise of complexity of the generation engine as well as the size of the knowledge usability base required to be used. For instance, in OlivaNova, the domain model could incorporate description of functions thanks to a first-order predicate logic that goes far beyond simple functions like in CRUDS pattern (since a full language could be used for this purpose). However, the UI generation process becomes much more complex. Its automation can be improved, but will require a significant amount of usability knowledge to be incorporated in the environment to become acceptable.

6.3 Future work

The future work derives from the aforementioned shortcomings:

- *Specifying model dialogue:* In the current version, the generated dialogues are not yet linked to each other. In the future, these dialogues will be automatically linked based on the temporal operation between the tasks (See Figure 6-2).

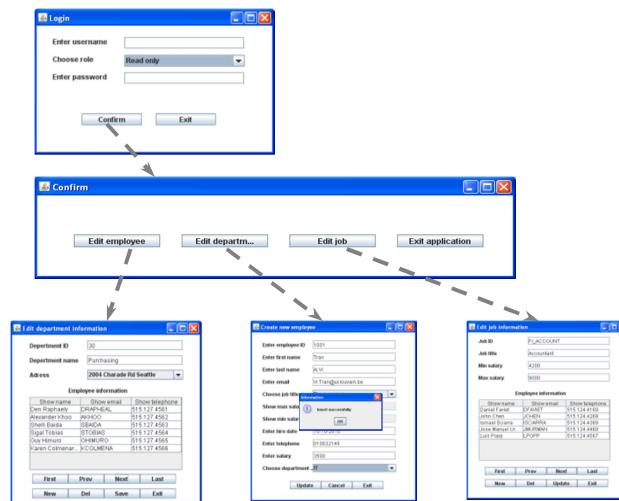


Figure 6-2: Linking Perform shipment order forms

- *Enhancing the set of functions:* These additional functions should be created to better meet the development needs of the database application

such as Print(), Report(), ImportToImage(), ... These generic functions should be generic enough in order to cover a wide range of database applications but also expressive enough to feed a reasonable generation process.

- *Allowing the designer to generate user interfaces for non-database application:* Based on the actual needs, most information does not come from the database, for instance, information from systems such as calendar, device's information, environments, guides. These sources should be at one point taken into account.
- *Supporting calculation formula:* Based on patterns from Coad & Yourdon which are not partially supported by DB-USE, designers and users need formula such as HowMuch() and RateMe(). In order to perform these methods, DB-USE should define the generic operations such as sum, average, multiple, plus, subtract, statistical operations. For instance, Figure 6-3 depicts the pattern "Item-lineItem" from Coad & Yourdon. In this pattern, the calcForMe method can be understood as a "sumPrice" method. This method is performed by multiplying the reference value by its quantity.

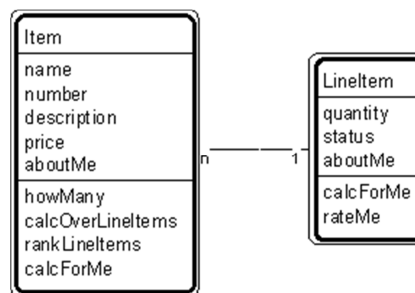


Figure 6-3: "Item - Line Item" Pattern

- *Improving the DB-USE tool scope:* It is necessary to add the user model into DB-USE tool. Indeed, the different UIs should be generated for different types of end-user depending on the information contained in the user model. For this purpose, new usability guidelines should be determined that provide us with some guidance on how to influence the UI based on user properties. This could also go beyond the scope of this thesis by considering alternate interaction styles depending on user's preferences (See Figure 6-4). For instance, a command language could be added to a form filling when the user experience with the task is moderate to rich and the user experience of the system is rich and the task motivation is high, while user experience with interaction devices is moderate (first line of Figure 6-4). This could give for instance a form-based UI combined with a command line for the command language, or even with function keys (See Figure 6-5). In addition to code generation in Java, DB-USE should allow the designer/developer to target other programming languages such as C++, Visual Basic.

Future work

Interaction style	Task experience	System experience	Task motivation	Experience with modern interaction devices
Command Language	moderate to rich	rich	High	moderate
Programming language	Rich	rich	Rich	moderate
Natural Language	Rich	moderate	Low	high
Query language	Rich	moderate	Moderate	high
Questions/Answers	Elementary	elementary to moderate	Low	moderate to rich
Function keys	moderate to rich	elementary	Low	low
Menu selection	elementary	elementary	Low	low
Form filling	elementary to rich	elementary to rich	low to moderate	elementary to moderate
Multi-windowing	elementary	elementary	Low	low
Direct manipulation	elementary	moderate	Low	low
Iconic interaction	elementary to moderate	moderate	low to moderate	low
Graphic interaction	elementary	moderate	low to moderate	moderate
Multimedia interaction	elementary	moderate	Low	moderate
3D User Interfaces	elementary	moderate	Low	high

Figure 6-4: Possible interaction styles depending on user preferences

Figure 6-5: A UI combining different interaction styles: form filling, command language, and function keys.

- Extending relationships between objects:** The discussed relationships between objects are first order relationships. The first order relationships of an object A (employees object) are the ones that directly connect to A; the second order relations of A are the ones that connect to A through another object (e.g., department object). (See Figure 6-6). DB-USE should take into account relationships at a higher level than first order ones.

- Conducting an experimental study:** DB-USE has been validated on two real-world case studies, which is a reasonable starting point. The next step should consist in defining and conducting an experimental study in order to determine the value of the methodological process. Ideally speaking, one should identify two or three projects having different characteristics, recruit a significant amount of designers and developers with different levels of expertise and group them into two groups: one group using DB-Use and another one without. Then, the results could be compared based on the same case studies based on both qualitative (e.g., designers satisfaction) and quantitative (e.g., time to design) metrics. Consequently to this, this thesis did not perform any evaluation of the UI issued by this methodological process. Therefore, it should be also identified when and how such generated UIs are different from UIs produced by a typical user interface development life cycle. Such two studies are certainly desirable, but represent a considerable amount of work, without speaking about the availability and the cost of the persons involved in such an experiment.

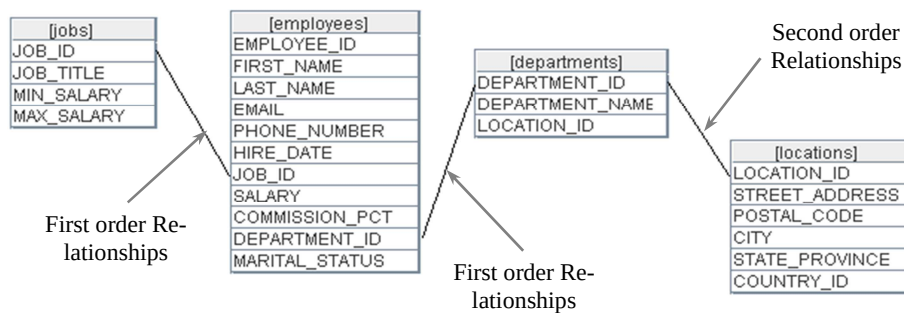


Figure 6-6: First order Relationship - example

- Applying “From Domain to Task models” tool to DB-USE tool:** In order to reduce the work of the model analyst if he wants to create a new task model. We can apply “From Domain to Task models” [Prib07] tool to DB-USE tool. “From Domain to Task models” tool is depicted in Figure 6-7. This tool allows the model analyst to generate tasks from the domain objects.

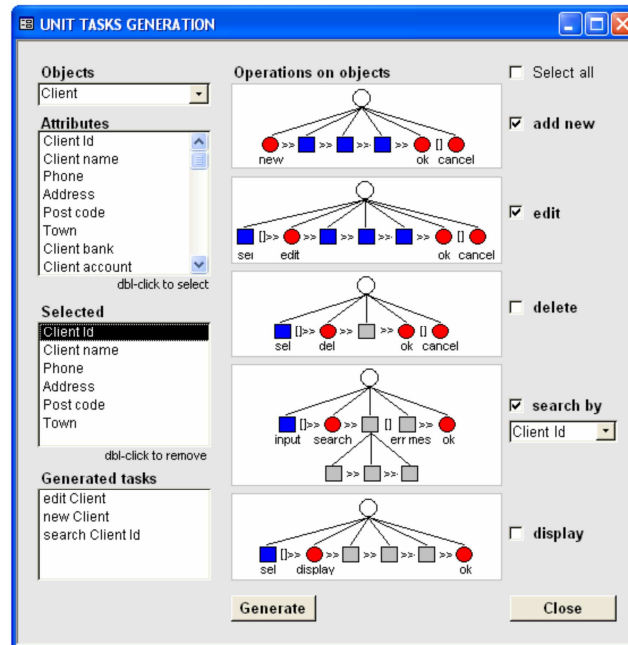


Figure 6-7: “From Domain to Task models” tool

REFERENCES

- [Baar92] de Baar, D., Foley, J.D. and Mullet, K.E., *Coupling Application Design and User Interface Design*, Proc. of ACM Conf. on Human Aspects in Computing Systems CHI'92 (Monterey, May 1992), ACM Press, New York, 1992, pp. 259-266.
- [Baja06] Bajaj, A., *Inferring the User Interface from an EER Data Schema*, Proceedings of the Americas Conference on Information Systems AMCIS'2006 (Acapulco, Mexico, August 2006).
- [Balz95] Balzert, H., *From OOA to GUI - The JANUS-System*, Proceedings of the 5th IFIP TC13 Conference on Human-Computer Interaction INTERACT'95 (Lillehammer, 25-29 June 1995), K. Nordbyn, P.H. Helmersen, D.J. Gilmore and S.A. Arnesen (Eds.), Chapman & Hall, London, 1995, pp. 319-324.
- [Barc00] Barclay, P., Kennedy, J., *Teallach's Presentation Model*, Proc. Of ACM Working Conference on Advanced Visual Interfaces AVI'2000 (Palermo, 23- 26 May 2000), ACM Press, New York, 2000, pp. 151-154.
- [Barc99] Barclay, P.J., Griffiths, T., McKirdy, J., Paton, N.W., Cooper, R. and Kennedy, J., *The Teallach Tool: Using Models for Flexible User Interface Design*, Proc. 3rd International Conference on Computer-Aided Design of User Interfaces (CADUI'99), 21-23 October 1999.
- [Bart01] Barth, B., Butler, G., Czarnecki, K., and Eisenecker, U., *Generative programming*. In ECOOP 2001 Workshops, Panel and Posters, Budapest, Hungary, June 2001, volume 2323 of Lecture Notes in Computer Science. Springer Verlag, 2001.
- [Boda93] Bodart, F., Hennebert, A.M., Leheureux, J.M., Sacre, I., and Vanderdonckt, J., *Architecture elements for highly-interactive business-oriented applications*. Lecture Notes in Computer Science (volume 753 of LNCS), Springer, Verlag, 1993.
- [Boda94] Bodart, F., Hennebert, A.M., Leheureux, J.M., Provot, I., and Vanderdonckt, J., *A Model Based Approach to Presentation: A Continuum from Task Analysis to Prototype*, Proc. of 1st Eurographics Workshop on Design Specification and Verification of Interactive System (DSVIS '94), Eurographics, Carrara, Italy, 1994, pp. 25-39.
- [Boda95a] Bodart, F., Hennebert, A.-M., Leheureux, Provot, J.-M., I., Sacre, B., Vanderdonckt, J., *Towards a Systematic Building of Software*

REFERENCES

- Architectures: the TRIDENT Methodological Guide*, Proc. of 2st Eurographics Workshop on Design Specification and Verification of Interactive System (DSVIS'95), Vienna, Springer, 1995, pp. 237-25.
- [Boda95b] Bodart, F., Hennebert, A.M., Leheureux, J.M., and Vanderdonckt, J., *Computer-Aided Window Identification in TRIDENT*, Proc. of the 5 IFIP TC13 Conf. on Human Computer Interaction Interact'95 ,Chapman & Hall, London, 1995, pp. 331-336.
- [Budi96] Budinsky, F.J., Finnie, M.A., Vlissides, J.M., Yu, P.S., *Automatic code generation from design patterns*, IBM Systems Journal, v.35 n.2, 1996, pp. 151-171.
- [Burm06] Burmeister, R., Pohl, C., Bublit, S., Hugues, P., *SNOW: A Multimodal Approach for Mobile Maintenance Applications*. Proc. of the 15th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (2006).
- [Coad95] Coad, P., North, D. and Mayfield M., *Object Models: Strategies, Patterns, and Applications*, Yourdon Press, Englewood Cliffs, NJ, 1995.
- [Do05] Do, T., A Framework for Multi-Agent Systems Detailed Design. Ph.D. Thesis, Université Catholique de Louvain, Institut d'Administration et de Gestion (IAG), Louvain La Neuve, Belgique, July 2005
- [Eise00] Eisenstein, J., Puerta, A., *Adaptation in Automated User-Interface Design*, Proc. of the 5th international conference on Intelligent user interfaces, January 09-12, 2000, New Orleans, Louisiana, United States, pp. 74-81.
- [Eise02] Eisenstein, J., Rich, C., *Agents and GUIs from Task Models*, Proc. of the 7th international conference on Intelligent user interfaces, January 13-16, 2002, San Francisco, California, USA
- [Elwe95] Elwert, T. and Schlunbaum, E., *Modelling and Generation of Graphical User Interfaces in the TADEUS Approach*, in Designing, Specication and Verication of Interactive Systems, Vienna, 1995, pp 193-208.
- [Espa06] Espana, S., Pederiva, I., Panach, J.I., Pastor, Ó., *Integrating Model-Based and Task-Based Approaches to User Interface Generation*, in: Calvary, Pribeanu C, Santucci G, Vanderdonckt J (eds) Computer-Aided Design of User Interfaces VI. Kluwer, Dordrecht, 2006, pp 255–263.
- [Forb00] Forbrig, P. and Lämmel, R., *Programming with Patterns*, Proc. of TOOLS-USA 2000. IEEE, 2000. (<http://citeseer.ist.psu.edu/312457.html>)
- [Gajo04] Gajos, K., Weld, D.S., *SUPPLE: Automatically Generating User Interfaces*, Proc. of the 9th international conference on Intelligent user interface, January 13-16, 2004, Funchal, Madeira, Portugal

REFERENCES

- [Grif01] Griffiths, T., Barclay, P.J., Paton, N.W., McKirdy, J., Kennedy, J., Gray, P.D., Cooper, R., Goble, C.A., Pinheiro da Silva, P., *Teallach: a Model-Based User Interface Development Environment for Object Databases*, Proc of Interacting with Computers, Vol. 4, No. 1, December 2001, pp. 31-68.
- [Grif98] Griffiths, T., McKirdy, J., Paton, N., Kennedy, J., Cooper, R., Barclay, B., Goble, C., Gray, P., Smyth, M., West, A., and Dinn, A., *An Open Model-Based Interface Development System: The Teallach Approach*, Proc. of DSV-IS, Eurographics, June 1998, pp. 32-49.
- [Gonz09] González-Calleros, J.M., Guerrero-García, J., Vanderdonckt, J., Muñoz-Arteaga, J., *Towards Canonical Task Types for User Interface Design*, Proc. of Joint 4th Latin American Conference on Human-Computer Interaction-7th Latin American Web Congress LA-Web/CLIHIC'2009 (Merida, November 9-11, 2009), E. Chavez, E. Furtado, A. Moran (Eds.), IEEE Computer Society Press, Los Alamitos, 2009, pp. 63-70.
- [Hart92] Hartson, R., Gray, P., *Temporal Aspects of Tasks in the User Action Notation*, in Human Computer Interaction, Vol.7, 1992, pp. 1-45.
- [Hein07] Heinrich, M., Winkler, M., Steidelmuller, H., Zabelt, M., Behring, A., Neumerkel, R., and Strunk, A., *MDA applied: A task-model driven tool chain for multimodal applications*, Proc. of the 6th International workshop on Task Models and Diagrams (TAMODIA 2007), 2007.
- [INRIA] <http://caml.inria.fr/>
- [iMat00] <http://legacy.imatix.com/pub/info/codegen.pdf>
- [Jans93] Janssen, C., Weisbecker, A., Ziegler, J.: *Generating User Interfaces from Data Models and Dialogue Net Specifications*. In Ashlund S., Mullet K., Henderson A., Hollnagel E., White T. (eds.): Proc. of INTERCHI'93. New York: ACM Press 1993, pp. 418-423.
- [Juli04a] Julien, D., Ziane, M., and Guessoum, Z., *GOLIATH: An Extensible Model-Based Environment to Develop User Interfaces*, Proc. of the Fourth International Conference on Computer Aided Design for User Interfaces (CADUI'2004), sponsored by ACM and jointly organized with IUI'2004, pp. 95–106. Kluwer Academics Publishers, January 2004.
- [Juli04b] Julien, D., Guessoum, Z. and Ziane, M., *Building User Interfaces Thanks to Eco-Resolution*, in International Conference on Advances in Intelligent Systems - Theory and Applications (AISTA'2004), in cooperation with IEEE Computer Society, November 2004.

REFERENCES

- [Lau09] Laudon, K.C., Laudon, J.P., *Management Information Systems*, Pearson Education (US), 2009.
- [Lece98] Lecerof, A. and Paternò, F., *Automatic Support for Usability Evaluation*, IEEE Trans. Software Eng., vol. 26, no. 10, Oct. 1998, pp. 863-888.
- [Limb09] Limbourg, Q., Vanderdonckt, J., *Multi-Path Transformational Development of User Interfaces with Graph Transformations*, in Seffah, A., J. Vanderdonckt, M. Desmarais (eds.), “Human-Centered Software Engineering”, Chapter 6, HCI Series, Springer, London, 2009, pp. 109-140.
- [Lonc96] Lonczewski, F., Schreiber, S., *The FUSE-System: An Integrated User Interface Design Environment*, in J. Vanderdonckt (ed.), Computer-Aided Design of User Interfaces, Namur University Press, Namur, 1996, pp. 37-56.
- [Lonc97] Lonczewski, F. (1997). *Providing user support for interactive applications with FUSE*. In Moore et al. (1997), pp. 253–254.
- [Look05] Looker, N., Munro, M., and Xu, J., *A Comparison of Network Level Fault Injection with Code Insertion*, presented at 29th Annual International Computer Software and Applications Conference (COMPSAC'05), Edinburgh, Scotland, 2005.
- [Luyt06] Luyten, K., Van den Bergh, J., Coninx, K., Vandervelpen, C., *Designing Distributed User Interfaces for Ambient Intelligent Environments Using Models and Simulations*. Computers and Graphics 30, 2006, pp. 702–713.
- [Lyna03] Lynagh, I., *Template haskell: A report from the field*, <http://www.comlab.ox.ac.uk/oucl/work/ian.lynagh/papers/>, 2003.
- [Mahf05] Mahfoudhi, A., Abed, M., Abid, M., *Towards a User Interface Generation Approach Based on Object Oriented Design and Task Model*, TAMODIA'2005 : 4th International Workshop on TAsk MOdels and DIAGrams for user interface design For Work and Beyond Gdansk, Poland September 26-27, 2005.
- [Moro06] Moroney, L. and MacDonald, M., *ASP.NET Applications*, in Pro ASP.NET 1.1 in VB .NET From Professional to Expert, Apress, 2006, pp. 183- 230.
- [Myer00] Myers, B., Hudson, S.E., Pausch, R., *Past, present, and future of user interface software tools*, ACM Transactions on Computer-Human Interaction (TOCHI), March 2000, pp. 3-28.

REFERENCES

- [Nich05] Nichols, J., Faulring, A., *Automatic Interface Generation and Future User Interface Tools*, In CHI. Proceedings of the Workshop on the Future of User Interface Design Tools, 2005.
- [Olse93] Olsen, D.R., Foley, J.D., Hudson, S.E., Miller, J., Myers, B.A., *Research Directions for User Interface Software Tools*, Behaviour & Technology, Vol. 12, No. 2, 1993, pp. 81-97.
- [Pate01a] Paternò, F., Mancini, C., and Meniconi, S., *Concursertrees: A Diagrammatic Notation for Specifying Task Models*, in S. Howard, J. Hammond, and G. Lindgaard, editors, Human-Computer Interaction INTERACT, Chapman and Hall, 1997, pp. 362–369.
- [Pate01b] Paternò, F., Mori, G., and Galiberti, R., *CTTE: An Environment for Analysis and Development of Task Models of Cooperative Applications*, in CHI '01 Extended Abstracts on Human Factors in Computer Systems. Seattle, Mar., ACM Press, 2001, pp. 21–22.
- [Pate01c] Paternò, F., *Task Models in Interactive Software Systems*, in Chang, S.K. (Ed.), Handbook of Software Engineering Knowledge Engineering, World Scientific Publishing Co. (ISBN: 981-02-4973-X), 2001.
- [Moli02] Molina, P.J., Melia, S., and Pastor, O., *JUST-UI: A User Interface Specification Model*, Proc. of CADUI'2002 Conference, Valenciennes, Francia, May, Kluwer Academics, 2002, pp. 323-334. Extended version to appear in Journal of Human-Computer Interaction (RIHM).
- [Moli03] Molina, P., Belenguer, J., and Pastor, O., *Describing Just-UI Concepts Using a Task Notation*, Proc. of DSV-IS'2003, Springer, Berlin, 2003.
- [Pinh00] Pinheiro da Silva, P., *User Interface Declarative Models and Development Environments: A Survey*, Proc. of DSV-IS2000, volume 1946 of LNCS, Limerick, Ireland, June 2000, pp 207-226.
- [Prib02] Pribeanu, C., Vanderdonckt, J., *A Methodological Approach to Task-Based Design of User Interfaces*, Studies in Informatics and Control 11, 2002, pp. 145–158.
- [Prib07] Pribeanu, C., *Tool Support for Handling Mapping Rules from Domain to Task Models*, Coninx, K., Luyten, K., Schneider, K. (Eds.): Proc. of TAMODIA 2006, Hasselt, Belgium, 23 – 24 October. Lecture Notes in Computer Science - LNCS 4385, Springer 2007, pp. 16-23.
- [Puer94a] Puerta, A.R., Henrik Eriksson, Gennari, J.H., Musen, M.A., *Beyond Data Models for Automated User Interface Generation*, Proc. of HCI'94: People and Computers. Glasgow, UK, August 1994, pp. 353–366.

REFERENCES

- [Puer94b] Puerta, A., *Model-Based Automated Generation of User Interfaces*, Proc. of Nat'l Conf. Artificial Intelligence, MIT Press, Cambridge, Mass., 1994, pp. 471-477.
- [Puer96] Puerta, A., *The MECANO Project: Comprehensive and Integrated Support for Model-Based Interface Development*, Proc. of CADUI 96, J. Vanderdonckt, ed., <http://www.info.fundp.ac.be/~jvd/dsvis/cadui-96.html>.
- [Puer99] Puerta, A., Cheng, E., Ou, T., and Min, J., *MOBILE: User-Centred Interface Building*, Proc. of Computer-Human Interaction (CHI '99) , 1999, pp. 426-433.
- [Rade06] Radeke, F., Forbrig, P., Seffah, A., Sinnig, D., *PIM Tool: Support for Pattern-driven and Model-based UI Development*. In Proc. TAMODIA 2006, LNCS, Vol. 4385, pp. 82-96.
- [Rade07] Radeke, F., Forbrig, P., *Patterns in Task-Based Modeling of User Interfaces*, in Task Models and Diagrams for User Interface Design, Springer, Berlin, 2007, pp. 184-197.
- [Reic08] Reichart, D., Dittmar, A., Forbrig, P., Wurdel, M., *Tool Support for Representing Task Models, Dialog Models and User-Interface Specifications*, Interactive Systems: Design, Specification, and Verification, Kingston, Canada, July 16-18, 2008 Proceedings, LNCS Vol.5136, pp.92-95, Springer, ISBN:978-3-540-70568-0
- [Rosa07] Rosado da Cruz, A.M., Pascoal de Faria, J., *Automatic Generation of User Interfaces from Domain and Use Case Models*, Quality of Information and Communications Technology (QUATIC2007), 2007.
- [Schl96] Schlunbaum, E. and Elwert, T., *Automatic User Interface Generation from Declarative Models*, In: J. Vanderdonckt, Editor, 1996. Proc. of Computer Aided Design of User Interfaces (CADUI'96), pp. 3-18.
- [Silv00] Da Silva, P.P., Griffiths, T., Paton, N., *Generating user interface code in a model based user interface development environment*, Proc. of Advanced Visual Interfaces (AVI'00). ACM Press, New York, pp. 155-160.
- [Stoc04a] Stocq, J., Vanderdonckt, J., *WOLD: A Mixed-Initiative Wizard for producing Multi-Platform User Interfaces*, Proc. of the 9th international conference on Intelligent user interface, 2004, pp. 331-333.
- [Stoc04b] Stocq, J., Vanderdonckt, J., *A domain model-driven approach for producing user interfaces to multi-platform information systems*, in AVI '04: Proc. of the working conference on Advanced visual interfaces, ACM Press, 2004, pp 395-398.

REFERENCES

- [Szek95] Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., Salcher, E., *Declarative Interface Models for User Interface Construction Tools: The Mastermind Approach*, in L. Bass C. Unger (eds.): Engineering for Human-Computer Interaction, Proc. of EHCI'95. London: Chapman & Hall 1995, pp. 120-150).
- [Vand93] Vanderdonckt, J., Bodart, F., *Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection*, Proc. of the ACM Conf. on Human Factors in Computing Systems (INTERCHI'93), Amsterdam, 1993, ACM Press, New York , pp. 424-429.
- [Vand95] Vanderdonckt, J., *Accessing Guidelines Information with SIERRA*, Proc. of the Fifth IFIP TC13 Conference on Human-Computer Interaction (INTERACT' 95), S.A. Arnesen & D. Gilmore (Eds.), Chapman & Hall, London, 1995.
- [Vand99] Vanderdonckt, J., *Advice-Giving Systems for Selecting Interaction Objects*, Proc. of the 1999 User Interfaces to Data Intensive Systems, September 05-06, 1999.
- [Waute10a] Wautelet, Y., Achbany, Y., Lange, JC. and Tran, V., *Developing a Collaborative Supply Chain Management Platform a Service Oriented Approach*, in Intelligent Systems in Operations: Models, Methods, Applications, Information Science Publishing, 2010.
- [Waute10b] Wautelet, Y., Kiv, S., Lange, JC. and Tran, V., *A Software Package for Actors E-Collaboration in Outbound Logistics using a Multi-Agent Architecture: The TransLogisTIC Project*, in Supply-Chain Management: Theories, Activities/ Functions and Problems, Nova Science Publishers, New York, 2010.
- [Wils96] Wilson, S., Johnson, P., *Bridging the Generation Gap: from Work Tasks to User Interface Designs*, in Computer-Aided Design of User Interfaces. Namur University Press, 1996, pp. 77-94.
- [Wood95] Wooldridge, M. and Jennings, N.R., *Intelligent Agents: Theory and Practice*, The knowledge Engineering Review, Volume 10, Number 2, Cambridge University Press, 1995, pp. 115-152.
- [Wood96] Wooldridge, M., Jennings, N.R., *Special Issue on Intelligent Agents and Multi-Agent Systems*, Applied Artificial Intelligence Journal, Volume 9, Numero 4, 1996, pp. 74-86.
- [Wurd07] Wurdel, M., Forbrig, P., Radhakrishnan, T., Sinnig, D., *Patterns for Task- and Dialog-Modeling*. In Jacko, J.A. (ed.) HCI 2007. LNCS, vol. 4550, pp. 1226-1235. Springer, Heidelberg, 2007.

REFERENCES

- [Yu95] Yu, E., *Modelling Strategic Relationships for Process Reengineering*, Ph.D Thesis, Dept of Computer Science, University of Toronto, Canada, 1995.
- [Zamb00] Zambonelli, F., Jennings, N.R., Wooldridge, M., *Organizational Abstractions for the Analysis and Design of Multi-Agent Systems*, Proc. of the 1st International Workshop on Agent-Oriented Software Engineering, volume 1957 of LNCS, pp 243-252.

Appendix

1. Human Resource Management

1.1 Task Model

The designer can create the task model of *Employee management* by using DB-USE tool or load it from a XML file created by CTT tool. In this case, the task model is loaded from a XML file created by CTT tool then the designer modifies it if he/she wants. Figure 1 shows how to load a CTT task model in DB-USE tool.

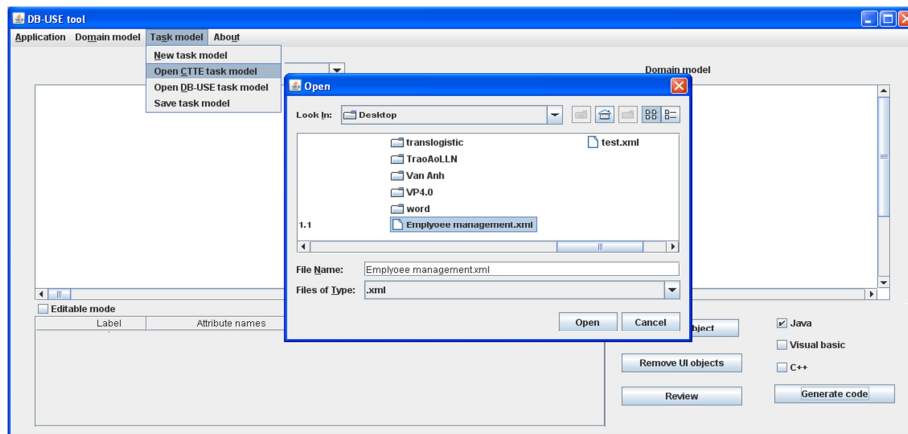


Figure 1: Load task model

We cannot show the task model of *Employee management* on one or two pages; it is very large. So the description of the task model is started with the principal tasks; then these tasks will be presented in details later. Figure 2 depicts the main tasks of the employee management. Once the end-user has logged into the application, he/she can edit the employee information, the department information, and the job information or exit the application. In order to edit the employee information, the end-user can create new employee, review employee information, modify an existing employee, delete existing one, and search employee. If the end-user wants to work on the department information, he/she can review it, modify it, delete an existing department, or make the department report. Besides, the end-user can review, modify or delete the job information.

In order to log into the application, the end-user has to choose the role from a list of roles and to enter the user name and password. Once he/she has logged into the application; he/she can choose the task that he/she want to work on. If he/she wants to work on the employee information he/she has to choose *Edit employee* task; if he/she wants to work on the department information he/she has to choose *Edit department* task; and finally, if he/she wants to work on the job information he/she

has to choose *Edit job* task. The end-user can exit the application whenever he/she wants by choosing the *Exit application* task.

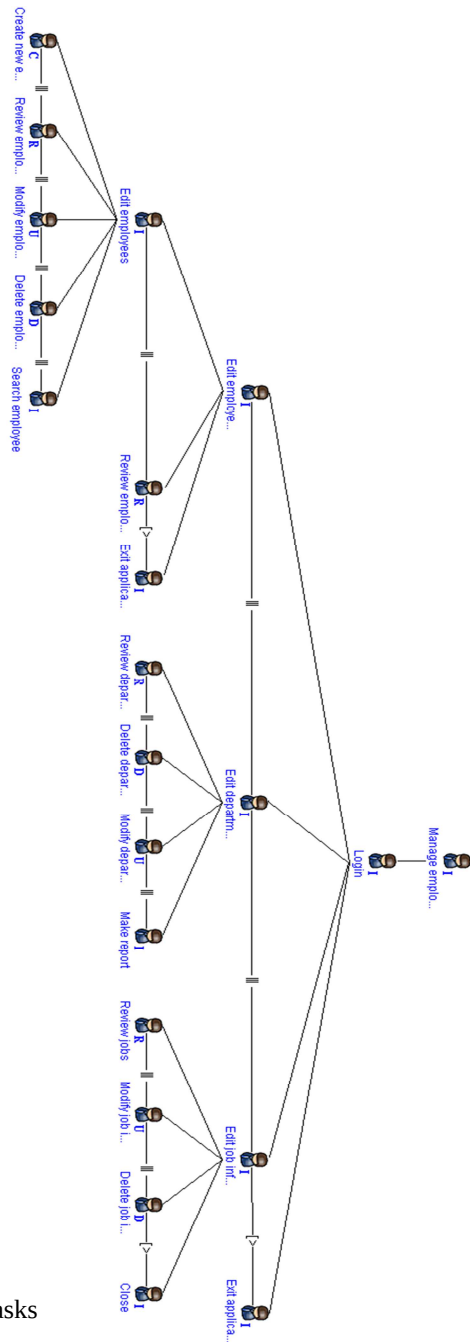


Figure 2: Principal tasks

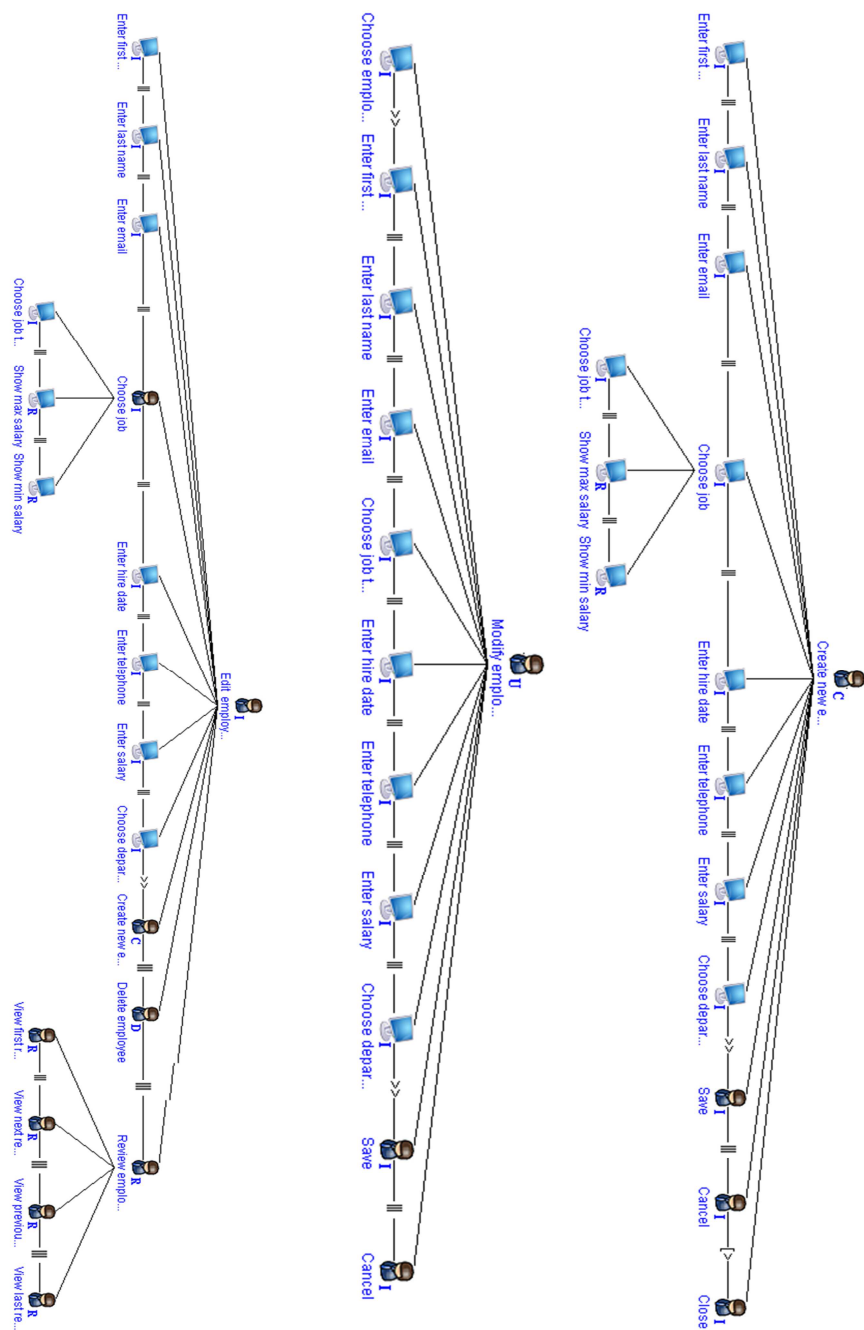


Figure 3: “Create new employee”, “Modify employee information”, and “Edit employee information” tasks

We chose to present three main tasks among the sub-tasks of the *Edit employee* task. They are

Create new employee, Modify employee information, and Edit employee information tasks (See Figure 3). The creation or modification of the employee information is performed by the end-user. Once the first name, the last name, the email address, the telephone, the job title, the hire date, the department name and the salary have been modified or entered, the end-user has two choices: save or cancel these changes. Specially, in the *Edit employee information* task, beside of modification of employee information, the end-user can create a new employee, delete an existing one, or review the information of the next employee, the previous employee, the first employee, or the last employee.

1.2 Domain Model

The domain model is created from Oracle Human Resources database by using a data type mapping table (Table 4-4). This database is provided by Oracle system and it is automatically installed when you install the oracle database software on your computer. We chose Human Resources database of Oracle because it is a simple Human Resource Employee database system; besides, employee data is organized and simple to use.

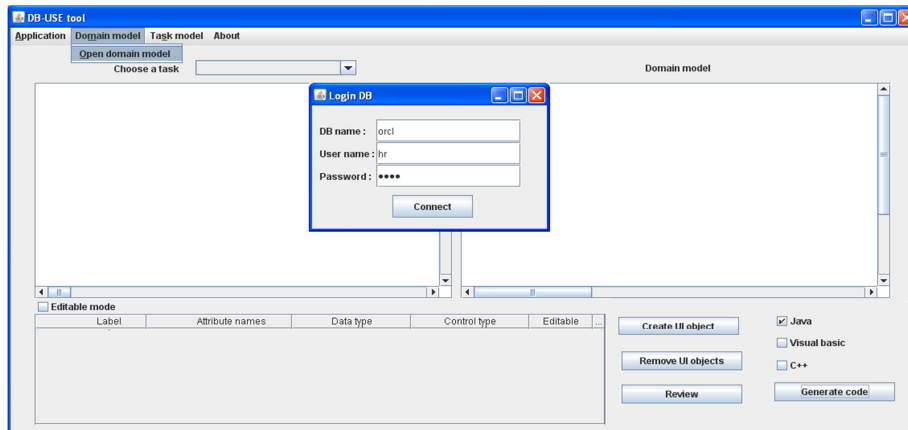


Figure 4: Load domain model

Figure 4 shows how to load domain model from database Oracle.

In domain model of employee management application, we have six domain's objects including Jobs, Employees, Users, Roles, Departments, Locations, and Countries objects (See Figure 5):

- **Employees** object describes the information of the employee such as the employee's name, the telephone number, the email address, the job title, the department that he/she works in, the date that he/she began his/her work, and the salary. Specially, an employee is identified by a unique employee number named *Employee_ID*.

- **Users** object describe the user name, the password and the role id of the user.
- **Roles** object describes the role of the user.
- **Departments** object describes the department information. This information is the department name, the location, and the department ID.
- **Jobs** object stores the job descriptions. A job is described by the job title, the job ID, the max salary, and the min salary.
- **Locations** object describes the address of a department including the street address, the postal code, the city name, the state province, and the country ID.
- **Countries** object describes the country name and the country ID.

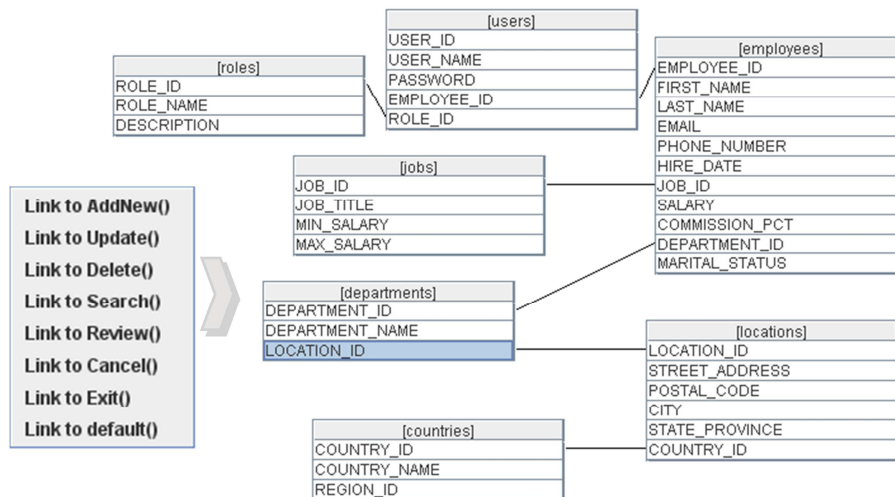


Figure 5: Employee management - Domain model is described by DB-USE

1.3 Generating the User Interface and Application Codes

1.3.1 Login Task

The goal of this task is to allow the end-user to log into the application by providing the user name, the password and the role to the system.

The designer can start with the *Login* task by choosing its name from the *Choose a task* combo-box and selecting the *Roles* and *Users* objects to from domain model.

In this case, the object *Users* is the edited object. The edited object is chosen by the designer.

In order to create the UI for performing this task, the designer has to make the links between *Enter user name* task and attribute *User_name* of object *Users*, *Choose role* task and attribute *Role name* of object *Roles*, *Enter password* task and attribute *PassWord* of object *Users*, *Confirm* task and *default()* function and *Exit* task and *Exit()* function.

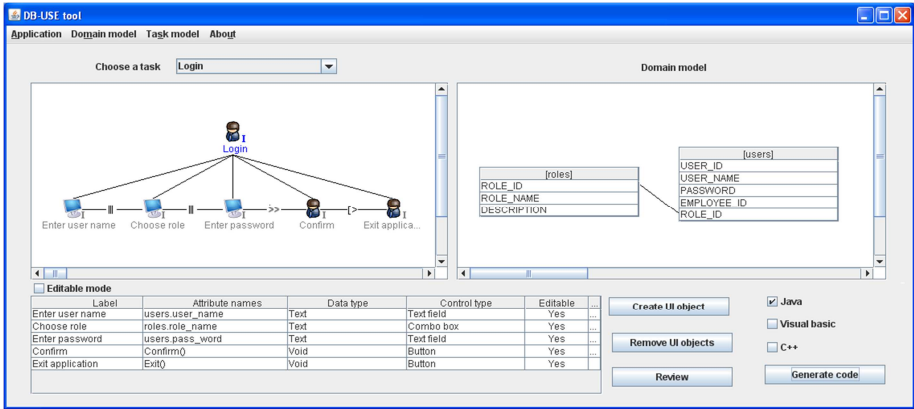


Figure 6: Create the UI for Login Task

Figure 6 depicts the creation of the UI for the *Login* task. The UI objects are automatically created by DB-USE based on the linked components once the designer has clicked on the *Create UI object* button. These objects are described in the table at the bottom of window.

Once the designer has clicked on the *Generate code* button; a message dialogue is displayed to announce to the designer the file name; this file is used to contain the code laid out by DB-USE.



Figure 7: Login Form

Figure 7 shows the login form which is created by DB-USE. In order to complete this form, the developer has to write a paragraph of code for logging into the application when the end-user clicks on the *Confirm* button. DB-USE generates only the code for performing the UI and getting the data for the *Choose role* combo-box.

1.3.2 Confirm Task

The goal of the *Confirm* task is to allow the end-user to choose the objects that he/she wants to work on. These objects are *Employees*, *Departments* and *Jobs* objects. Besides, the end-user can exit the application if he/she wants.

This task is a special one. In order to specify the UI for this task; we don't need to use the domain's objects. Its sub-tasks will be linked to the functions in the domain model. And the created controls are buttons or menus. The three tasks *Edit employee*, *Edit department*, *Edit job* are linked to the *default()* functions and the *Exit application* task is linked to the *Exit()* function (See Figure 8).

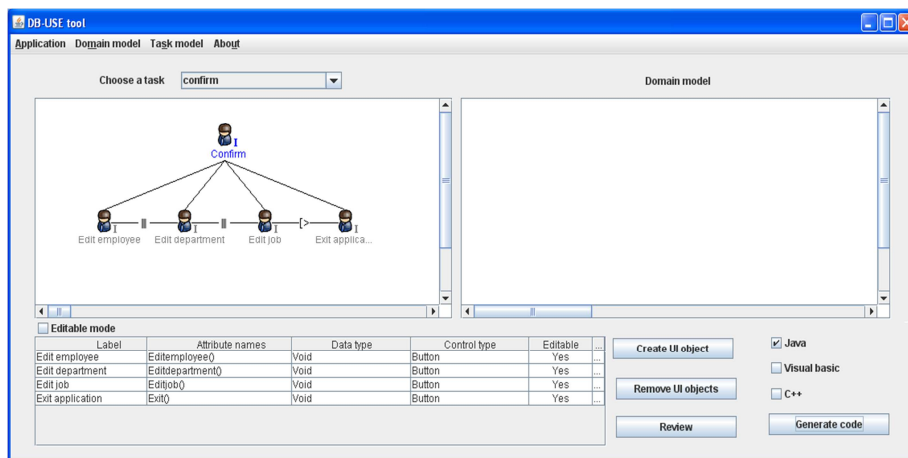


Figure 8: Create the UI for *Confirm* Task

Once the tasks have been linked to the functions, the code is generated by clicking on the *Generate code* button. The generated form is depicted in Figure 9. The developer can change the form title from *Confirm* to *Main form*.

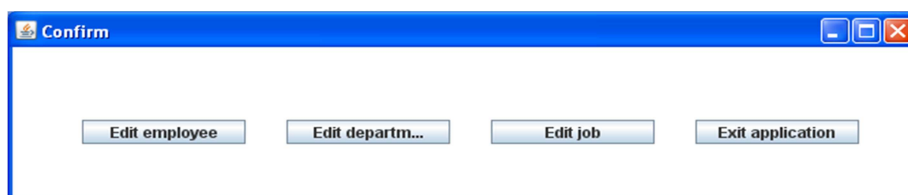


Figure 9: Main form

In order to complete this form, the developer has to write the code for opening the *Edit employee*, *Edit department*, *Edit job* forms when the end-user clicks on the *Edit employee*, *Edit department*, *Edit job* buttons.

1.3.3 Create new employee Task

The goal of this task is to allow the end-user create a new employee. Once *Create new employee* task has been selected in the *Choose a task* combo-box, this task is displayed to the designer as Figure 10.

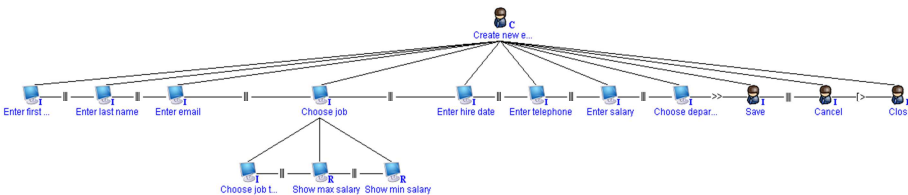


Figure 10: Create new employee task

The object *Employees* is determined to be an edited object. Besides, in order to create a new employee, the end-user has to give the employee id of this new employee. So the designer has to add a new task “*Enter employee ID*” into the *Create new employee* task.

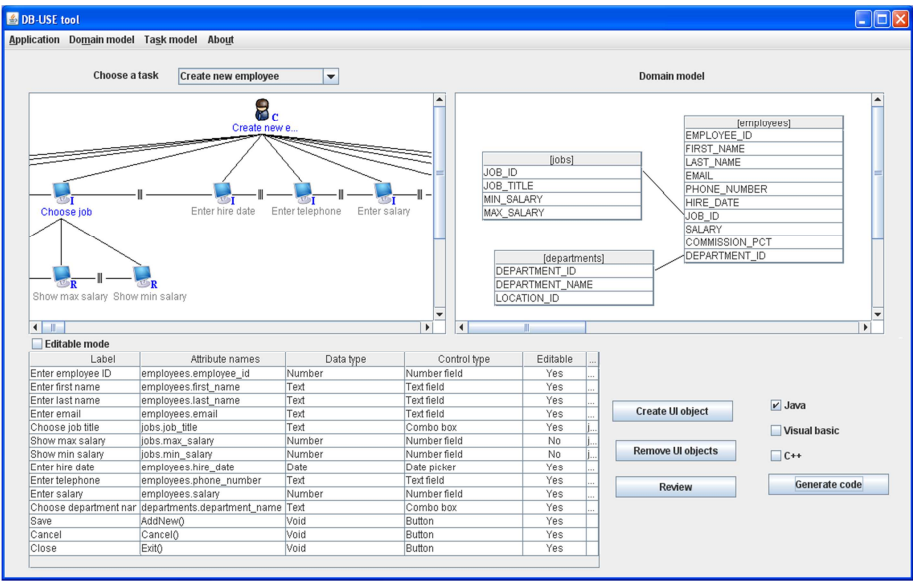


Figure 11: Creating the UI for Create New Employee Task

Now, the designer links its sub-tasks to the domain’s attributes and to the functions in the domain: the *Enter employee ID*, *Enter first name*, *Enter last name*, *Enter email*, *Enter hire date*, *Enter telephone*, *Enter salary* tasks are linked in the same order to the attributes *Employee_ID*, *First_Name*, *Last_Name*, *Email*, *Telephone_Number*, *Salary* of the object *Employees*; the *Choose job title*, *Show max salary*, *Show min salary* tasks are linked in the same order to the attributes *Job_Title*, *Max_Salary*, *Min_Salary* of the object *Jobs*; and finally, the *Choose*

department name is linked to the attribute *Department_Name* of the object *Departments*; the *Save* task is linked to the function *AddNew()*; the *Cancel* task is linked to the function *Cancel()*; the *Exit* task is linked to the function *Exit()*, see Figure 11.

Once the links have been created, the designer can click on the *Generate code* button to generate the user interface code and the application code for this task. The code is laid out in file *cCreatenewemployee.java*.

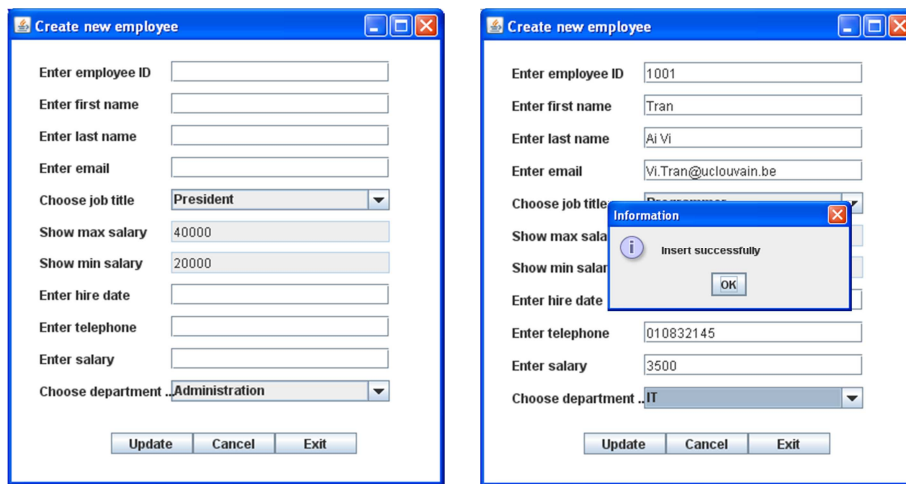


Figure 12: *Create new employee* form

Since the generated code is compiled and ran without modification by the developer; the *Create new employee* form is displayed as the left form in Figure 12.

The max salary and min salary are automatically changed when the end-user changes the job title. Once the end-user has entered the value for the controls in the form, he/she can click on the *Update* button to insert the new employee into the database; or click on the *Cancel* button to reset the value of the controls; or click on the *Exit* button to close this form.

There are two types of message displayed to the end-user when the *Update* button is clicked. The information message is displayed if the data is successfully inserted (See the right form in Figure 12); otherwise, the error message is displayed. Besides, the application also displays the warning messages if the data is not entered, or is incorrectly entered.

1.3.4 Review Employees Task

The goal of the *Review employees* task is to allow the end-user to verify the information of the current, first, previous, next and last employee, and print it. This task is depicted in Figure 13.

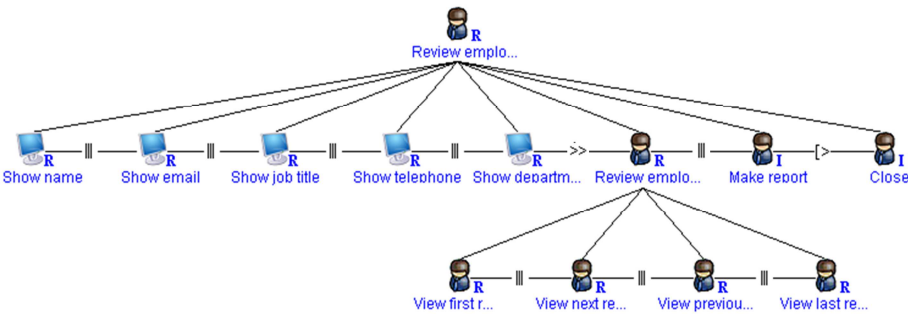


Figure 13: Review employees task

Before making the links between the tasks and the domain’s components, we want to modify the name of the tasks: “Show name” is modified to “Employee name” and the word “Show” is removed from the name of the tasks.

The domain’s objects interacting with this task are *Jobs*, *Departments* and *Employees*. The *Employees* object is determined as an edited object.

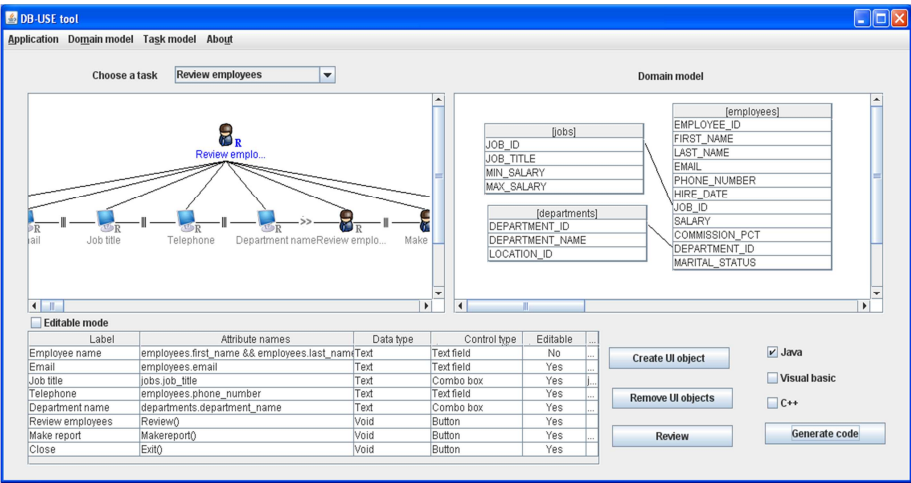


Figure 14: Creating the UI for Review employees task

The *Email*, *Telephone* tasks are linked to the attributes *Email* and *Phone_number*; but The *Employee name* task is linked to the attributes *First_Name* and *Last_Name*; the *Job title* task is linked to the *Job_Title* attribute of the *Jobs* object; the *Department name* task is linked to the *Department_Name* attribute of the

Department object. The *Review employees*, *Make report*, *Close* tasks are linked to the *Review()*, *Default()* and *Exit()* functions.

In this case, there is no linked function that is used to change the data such as *AddNew()*, *Update()*. So a message dialogue is displayed to the designer to ask him/her if he/she wants to change the control type of combo box to the control type of text field. If the answer is Yes then the value Yes of *Editable* attribute is changed to the value No.

The figure shows two versions of the 'Review employees' form. The left version has text input fields for 'Employee name', 'Email', 'Job title', 'Telephone', and 'Department name'. The right version has dropdown menus for 'Job title' and 'Department name'. Both forms include navigation buttons ('First', 'Prev', 'Next', 'Last') and action buttons ('Exit', 'Make report').

Figure 15: *Review employees* form

Figure 15 depicts the *Review employees* form generated based on the previous steps. If the designer has decided that the compo boxes are replaced by the text fields then the form generated is the left form; otherwise, it is the right form.

With this form, the end-user can review the employee information by clicking on the *First*, *Next*, *Previous*, *Last* buttons; in order to close this form he/she clicks on the *Exit* button. The code used for getting data from the database and performing the functions of these buttons is automatically generated by DB-USE.

In order to complete this form, the developer has to write the code for performing the function of the *Make report* button.

1.3.5 Link the generated forms each other

Once the code has been generated, the developer needs to take a look on this code then write more code to link the forms to each other and perform the functions that are not implemented in DB-USE yet. In this case, we are talking about the function *Make reports*.

Figure 16 depicts the forms generated by DB-USE. The application starts with the *Login* form. Once the end-user has logged into the application, he/she can decide which object he/she wants to work on by clicking on the *Edit employees* button if he/she wants to edit the employee information, on the *Edit departments* button if he/she wants to modify the department information, and finally on the *Edit Jobs* if he/she wants to edit job information. With the *Employees* form, the end-user can click on the different buttons to open the different forms.

The *Edit department information* and *Employees* forms were not discussed in the previous steps but they are also generated by DB-USE.

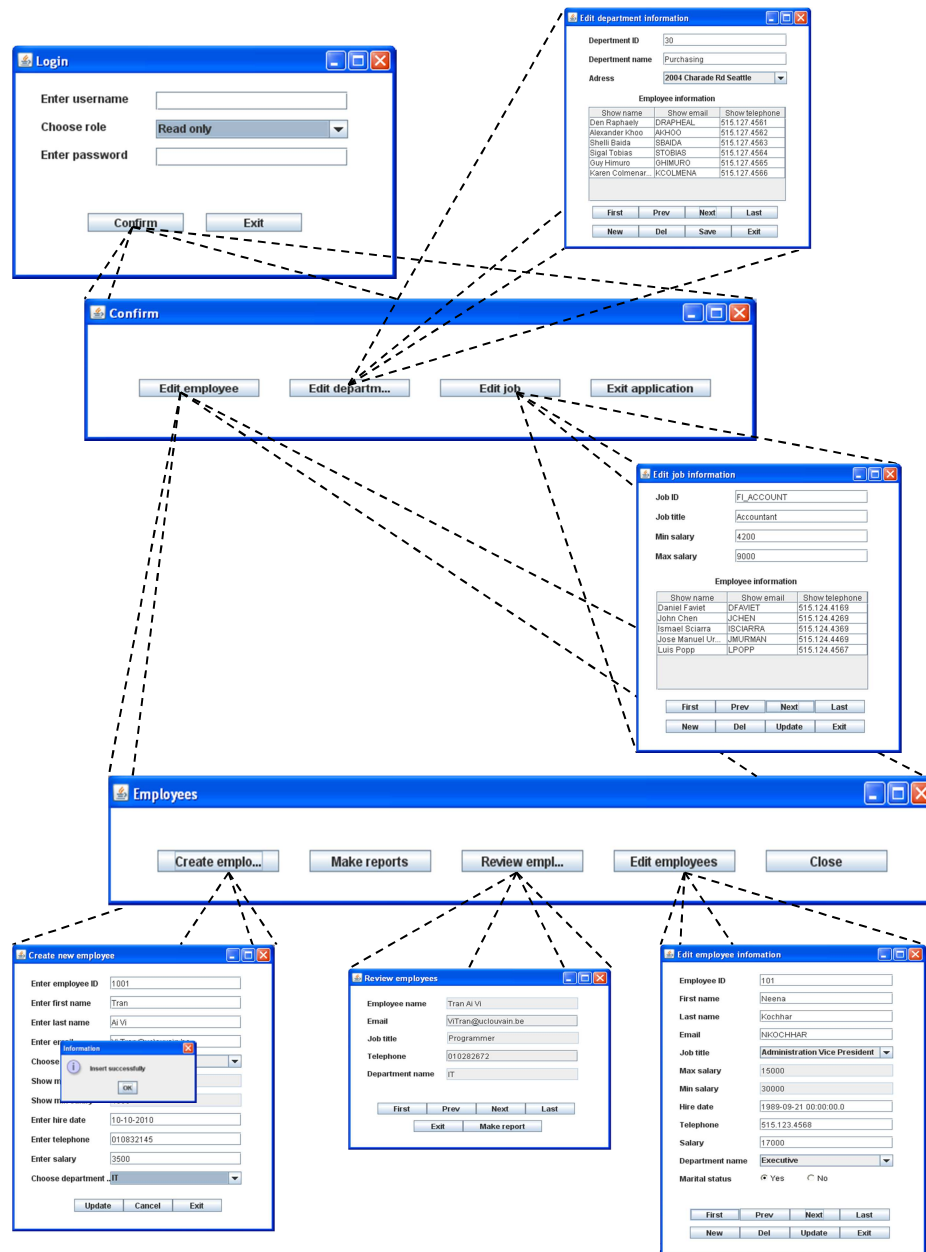


Figure 16: Linking the generated forms together

2. TransLogisTIC

The model presented hereafter is a process and interaction view of a supply chain with respect to outbound logistics. The model incorporates the processes realized at the actor's level and the interaction between the different actors. The individual actors, however, can play different roles. For example, a producer will sometimes be a shipper, when he/she fulfills orders from its clients, and sometimes be a final client, when he/she has ordered shipments from its own suppliers. Therefore, the model is presented in terms of roles played by actors.

In the processes involved in outbound logistics, four major roles can be identified:

- **The shipper:** This role is taken by an actor who needs transportation for a shipment. He has received an order from a client and has to deliver this order. The shipment has its own characteristic in terms of volume, transportation constraints, pickup and delivery dates, transportation mode and the shipper has its own expectation on the price he/she should be charged. The shipper will turn towards one or several transporters to realize that shipment.
- **The final client:** This role is taken by an actor who has ordered a shipment from a supplier and awaits its delivery. The client is generally not involved in the choice of the transporter, but is impacted by the planned delivery date and the delays that could arise in the realization of the transportation.
- **The transporter:** This role is taken by an actor owning transportation capacity who receives transportation orders from shippers for the realization of shipments to final clients. The core business of the actor having this role is the transportation. As such, he/she must realize these orders while maximizing his/her profit.
- **The logistic infrastructure manager:** This role is taken by an actor who owns the logistic infrastructure which is needed by transporters such as hubs, rail tracks, airports, docks... The utilization of this infrastructure is rented to the transporters while other services could be offered (transshipments, cross-docking, storage...).

Our generic process and interaction diagram focuses on collaboration. Collaboration consists of two mechanisms.

First, information sharing allows visibility between the actors in the supply chain. This information can generate value for each actor since it allows them to better plan their own operations and optimize their own decisions. Information sharing can appear at every stage. At strategic level, announcing its own planned requirements and developments will allow the other actors to position themselves better in their own strategic decisions. Tactically, it raises the opportunity to better fit the needs of clients and suppliers. Operationally, it allows all actors to be kept informed of the realization of one's operations and adapt more rapidly to problems encountered by one actor that could affect the others.

A second and more subtle aspect of collaboration is coordination. If information sharing has been a current practice in a supply chain for a long period of time, and trust has emerged between the actors, they can consider moving further to coordinate decisions. In this case, decisions are not only taken based on internal and shared information in order to improve one's own efficiency, but are discussed and taken in order to optimize some processes more globally. The goal is not only to take the decisions that are favorable for one actor but also decisions that are favorable for its clients and suppliers. In this case, it is the efficiency of the chain itself that can be improved.

For each role, the processes involved in outbound logistic can be categorized into different levels. At strategic level, the actors take long term decisions determining their relations to the other actors, fixing their long term objectives, aiming at matching their capacity to the market requirements and defining adequate corporate policies. At tactical level, the actors take medium term decisions which aim at achieving the strategic goals. Those decisions define the structure in which operations will take place. At operational level, day-to-day decisions are taken which direct the logistic flows. At each level, interactions occur between the actors playing the different roles.

On the generic process and interaction diagram, the processes and interactions are defined in a generic way, which could be extended further by defining for each process the tasks that are involved. This task definition gives rise to an extended model which is beyond the scope of this chapter. The processes are categorized by roles and levels.

At the strategic level, each player is involved with his/her own strategic analysis and defines his/her strategic goals. He/She determines his/her offer, his/her markets, his/her objectives and future investments, and his/her corporate strategies. He/She realizes global supply and demand planning and turns to long term business partners.

The shipper will evaluate his/her future transportation needs. He will determine the type of transportation he/she will require, the characteristics in terms of volume, specific constraints, reliability, quality, schedules, frequency and expected fares. He will scan the market for appropriate transporters to take care of his/her transportation activities and may eventually enter into long term contracts with those transporters.

The transporter will decide on which market he/she will present and with which offer. In this model, we use the term « presence » to refer to a strategic offer made by a transporter. A presence is the expression of this offer. It can define the destinations he/she serves, the capacity he/she makes available on specific destinations pairs or in specific regions, the transportation mode that is involved, the type of clients he/she is willing to work with and possibly the tariffs he/she offers. The presence allows his/her offer to be visible by the market. He can find shippers reacting to his/her presence definition with whom he/she may enter into long term contracts.

The infrastructure manager is involved in a similar strategic process. He evaluates the capacity he/she plans to offer to the transporters, and the policies for

accepting transporters on its infrastructure. He determines how the transporters will use this capacity, under what conditions, and how this will be priced. He determines the development of his/her infrastructure.

The final client has generally little impact on strategic outbound logistics decisions. In our model, he/she does not choose the transporters that will be selected, or the paths that will be used by the shipments, although, in practice, he/she could be involved by his/her suppliers in those decisions.

At the tactical level, more concrete decisions are taken which aim at implementing the strategy defined at the corporate level. The business units adapt this strategy in terms of policies under which the operations will happen. For example, the shipper may determine rules for allocating concrete requests for shipment to specific transporters, or to realize possible trade-off between reliability, speed and price of the transportation for each kind of shipment. The transporter will define how he/she will implement his/her announced presence, by defining connections and paths. Connections are links between two locations on which he/she plans to have some transports. Defining connections structures his/her network and allows him to reserve the required capacity on those connections. He will also define paths, which are decisions about how to route the shipments from the origin to the destinations by using a series of connections. The paths can be dependent on the type of shipments.

The operational level deals with concrete shipments that have to be realized. The operational level itself can be decomposed into the scheduling phase and the monitoring phase.

In the scheduling phase, the players plan the short term realization of the outbound logistics. The shipper receiving orders from his/her clients will determine his/her day-to-day transportation needs. In our model, those needs are defined as « logistic requests ». Such requests identify the shipment, the origin and the destination, the requested pickup and delivery dates and other constraints specific to the shipment. Once the logistic requests have been identified for a scheduling horizon, the shipper tries to match those requests to a set of transporters, based on the tactical policies of the business unit. The shipper builds, for each request, one or several « transportation calls », which are calls that will be addressed to the transporters to realize a part of the journey of the shipment. The shipper then submits these calls to the relevant transporter.

The transporter receives the transportation calls, and will accept or refuse them. If he/she accepts them, he/she may want to reserve the required capacity on the path that will be used by the shipment. According to his/her tactical policies, he/she may already allocate the shipment to a series of connections. This allocation allows him to evaluate the delivery date which he/she can confirm to the shipper. As a new scheduling horizon begins, the transporter's scheduler lists the shipments allocated, or to be allocated, to each specific connection. He verifies the feasibility of the allocation, and modifies it if needed. He creates « transports » to realize the shipments. In our model, a transport is a specific movement of a logistic unit (truck, train, barge...) on a connection. It has a capacity, a schedule, a list of stops and the

logistic unit is identified. The shipments are then allocated to specific transports. The path of each shipment is verified and required terminal operations, such as transshipments, are identified and communicated to the involved infrastructure managers.

The infrastructure manager receives an infrastructure load from the transporters, and schedules his/her operations and teams to meet such load. In an efficient supply chain, infeasibilities of one player are communicated to the others for schedule adaptations. When communication is absent, infeasibilities generally result in delays.

The monitoring phase consists of checking if everything goes as planned, which is rarely the case. The shipper wants to check where his/her shipments are, if the delivery dates are still confirmed, if the shipments encounter problems on their route to be aware as soon as possible of any delay or loss of the shipments. The final client should have the same visibility on the shipments he/she is awaiting.

The logistic players, i.e. the transporters and the infrastructure managers, can react by taking correctives measures in case of problems in the realization of the schedules. However, this requires, first, a great visibility on the realization of the transports, second, a great flexibility in rescheduling, and, third, good communication with the other logistic partners.

2.1 Task Model

The task model of TransLogisTIC is created by using the DB-USE tool for the following reasons:

- There was no task model yet so it is easier and faster if the designer uses the DB-USE tool to create the TransLogisTIC's task model directly.
- There are lots of tasks in TransLogisTIC whose user interfaces cannot be automatically generated by using DB-USE tool. These tasks are the ones that are performed by using the graphical objects such as the icons, the lines, the rectangles ... So that the designer only creates the needed tasks for generating the user interface; he/she doesn't need to create all of the tasks of TransLogisTIC in its task model.
- Finally, the designer can create the tasks then use them to generate the user interfaces in the same time.

Figure 17 depicts the task mode of TransLogisTIC. In order to reduce the complexity of the task model, it is represented by its mains tasks. These tasks were discussed in more details at the previous steps. When the end-user logs into the application, he/she will play one of four different roles including the client role, shipping role, transporter role, and logistic infrastructure manager role.

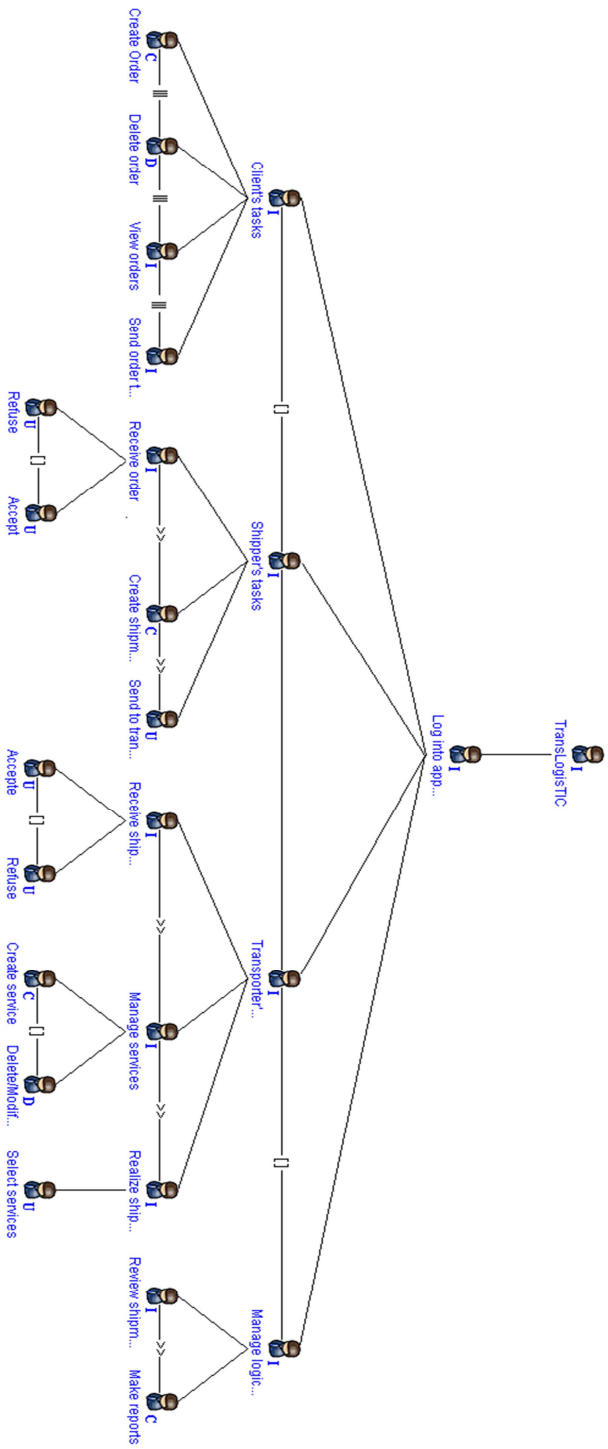


Figure 17 : TransLogisTIC's Task model

The main tasks of a client are the *Create order*, *Delete order*, *Review order*, and *Send an order to shipper* tasks.

Once the shipper has received an order from client, he/she changes the status of the order to *accepted* if he/she can meet its command, otherwise, to *refused*. Based on the order received from the client, the shipper will create the shipment order which will be sent to the transporter.

In the transporter site, the shipment orders are censored one by one to change their status. Besides, the shipper also creates or deletes the services which are used to perform the shipment order. Once a shipment order has been accepted by the shipper, he/she selects the needed services to perform it.

The main tasks of logistic infrastructure manager are only to review the shipment orders in order to issue warnings to the clients, transporters and shippers, or make the reports.

2.2 Domain Model

The domain model of TransLogisTIC is represented by more than 50 objects. Figure 18 only depicts a part of the domain model. These objects will be used in the next steps.

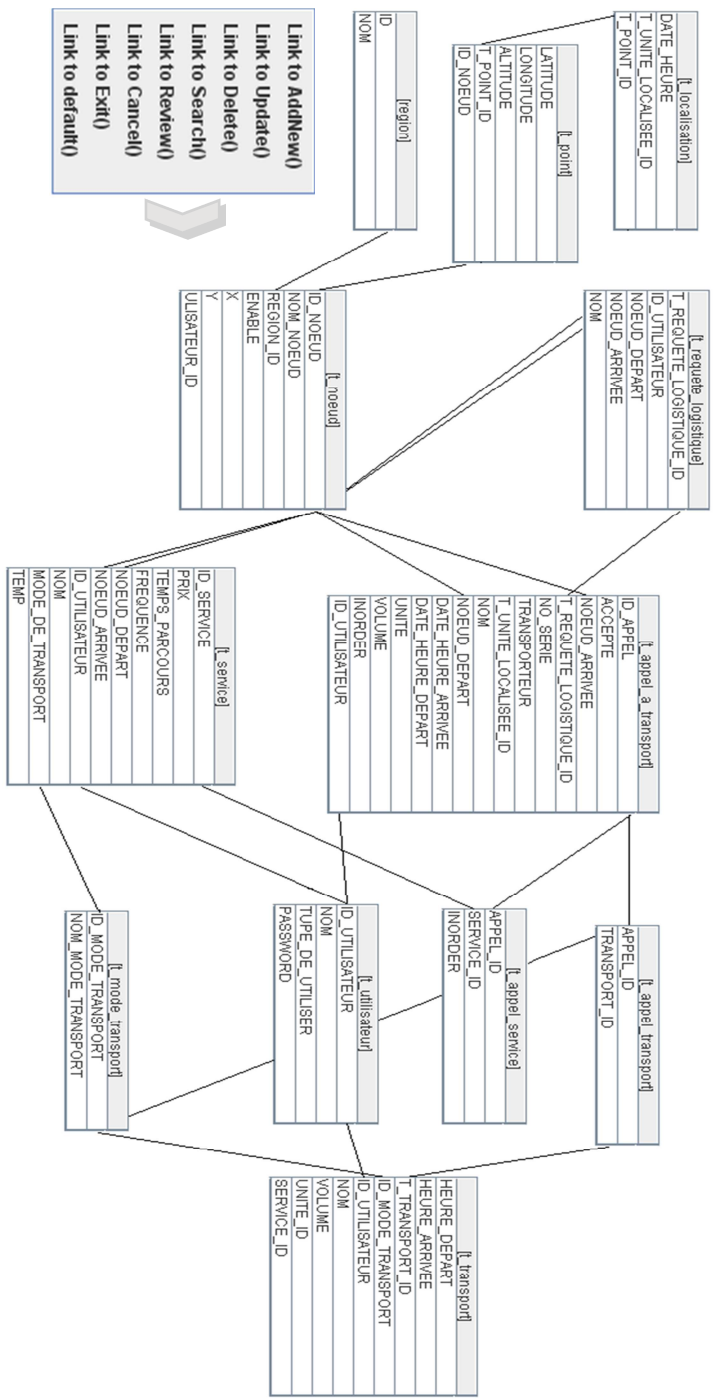


Figure 18: TransLogistic's Domain model

2.3 Generating the User Interface and Application Codes

2.3.1 Log into the application Task

In order to use the application, the end-user provides the user name, password, and the user role; there are four different roles including client role, shipping role, transportation's role, and role of logistic management. Based on the provided user role, the application will specify the user site; the different sites are performed by the different interfaces. The *Login to application* task is depicted as follow:

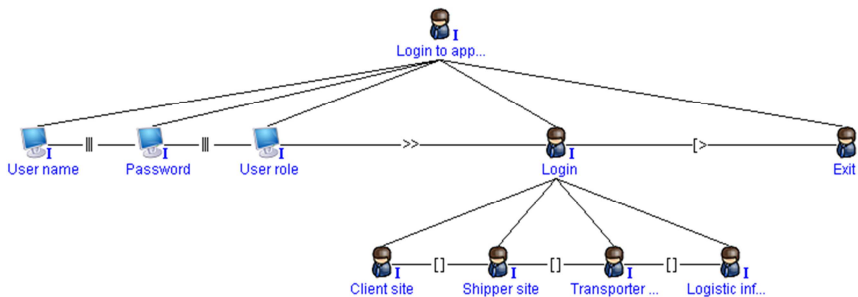


Figure 19: Login to application task

Based on the created task *Login to application* and the edited object *T_Utilisateur*, the designer makes links between the sub-tasks and the domain's components.

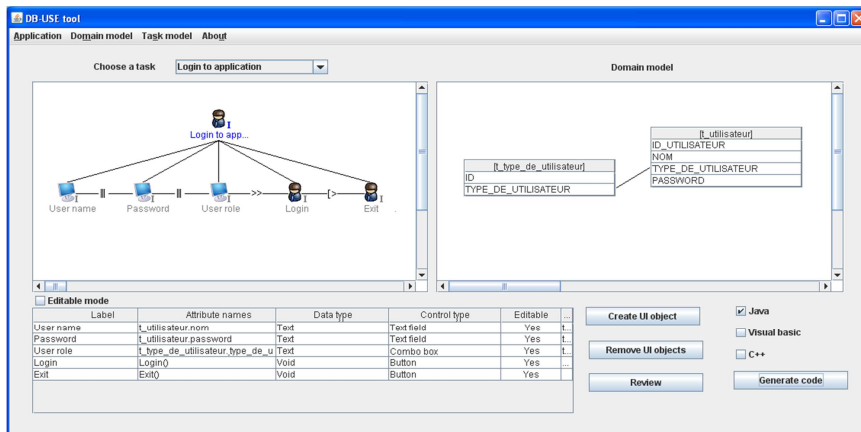


Figure 20: Generating the UI for Login to application task

The *User name*, *Password* tasks are linked to the *Nom*, *Password* attributes of the *T_Utilisateur* object. The *User role* task is linked to *Type_De_Utilisateur* attribute of the *t_Type_De_Utilisateur* object. The *Login* task is linked to *default()* function. The *Exit* task is linked to *Exit()* function. See Figure 20.

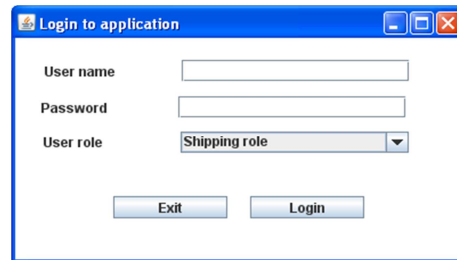


Figure 21: Login to application form

Figure 21 depicts the *Login to application* form generated by DB-USE tool.

2.3.2 Create a new service Task

A service in TransLogisTIC is the one that is used to describe the transportation of the merchandise from the shipping node (shipping address) to delivery node (delivery address) by a mean of transportation. The services are created and deleted in the transporter site. A service is represented by the service ID, service name, shipping address, delivery address, transporter (transport company), transportation mode (plane, train, truck, ship and so on), and price.

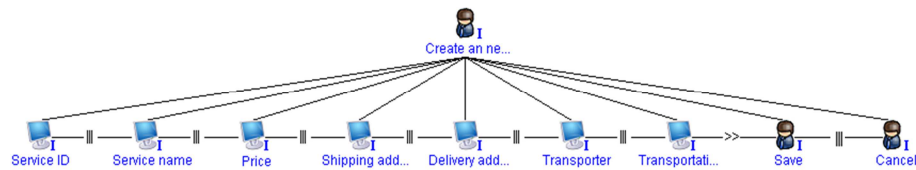


Figure 22: Create a new service task

Figure 22 depicts the *Create a new service* task.

The objects involved in the creation of user interface of *Create a shipment order* task are *T_Service* (is used to describe the services), *T_Noed* (is used to describe the addresses), *T_Utilisateur* (is used to describe the Transporters), *T_Mode_Transport* (is used to describe the transportation type) objects. The *T_Service* object is chosen as an edited object.

Once *Create a new service* task has been created and the domain objects have been determined; the designer will make the links between the tasks and the domain's components. The *Service ID*, *Service name*, *Price* tasks are linked to *ID_Service*, *Nom*, *Prix* attributes of the *T-Service* object; the *Shipping address*, *Delivery address* tasks are linked to the *Nom_Noed* attribute of the *T-noeud* object; the *Transporter* task is linked to the *Nom* of the *T_Utilisateur* object; the *Transportation mode* task is linked to *Nom_Mode_Transport* attribute of the *T_Mode_Transport* object; and finally, the *Save*, *Cancel* tasks are linked to the *AddNew()* and *Exit()* functions, see Figure 23.

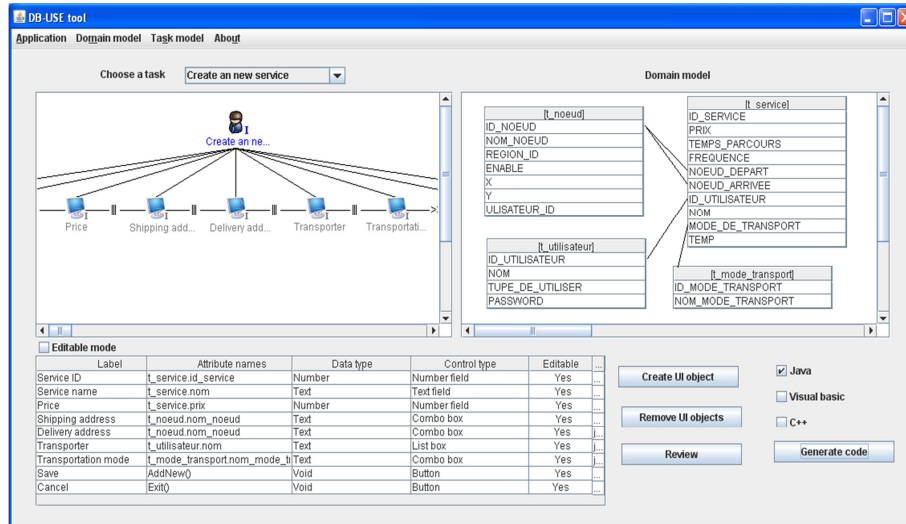


Figure 23: Generating the UI for *Create a new service* task

The screenshot shows the 'Create a new service' form. It has a title bar with the text 'Create a new service'. The form contains several input fields and dropdown menus. The 'Service ID' field has the value '32'. The 'Service name' field has the value 'new servie'. The 'Price' field has the value '100'. The 'Shipping address' dropdown menu has the value 'Ottignies'. The 'Delivery address' dropdown menu has the value 'Liege'. The 'Transporter' dropdown menu has the value 'Transporteur 2'. The 'Transportation mode' dropdown menu has the value 'Avion'. At the bottom of the form, there are two buttons: 'Save' and 'Exit'.

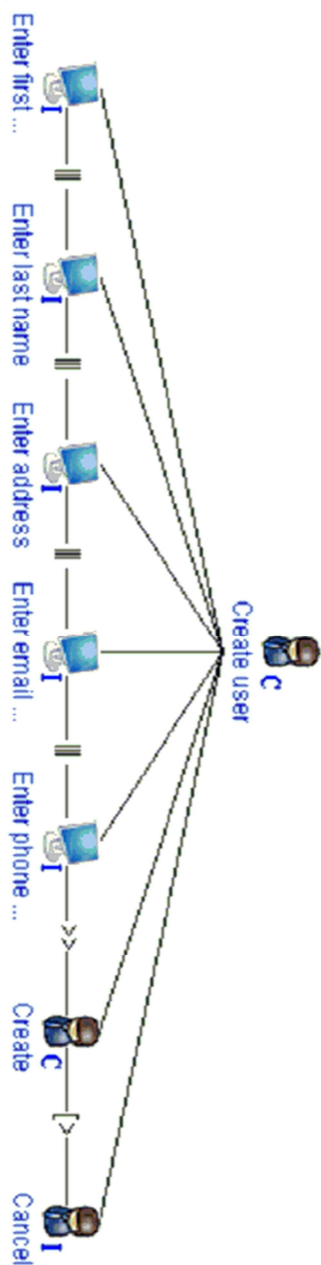
Figure 24: *Create a new service* form

Figure 24 depicts the *Create a new service* form. This is the result of the execution of the generated code by DB-USE tool without any modification. Once the data is completely entered by the end-user; he/she can click on the *Save* button to insert a new service into the database or on the *Exit* button to close this form.

3. Task patterns

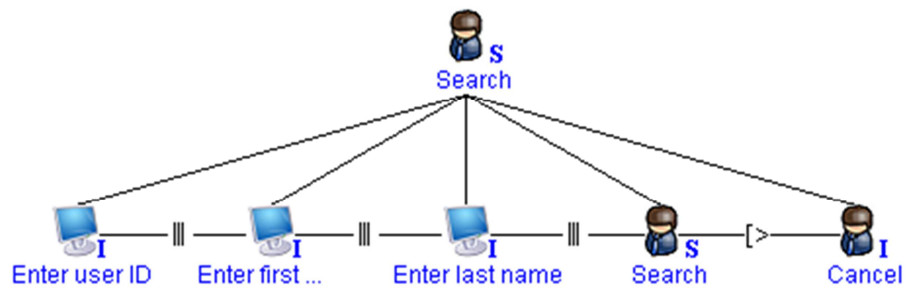
This section provides some tasks discussed in Section “Task library” in terms of DB-USE task tree.

Provide personnel information

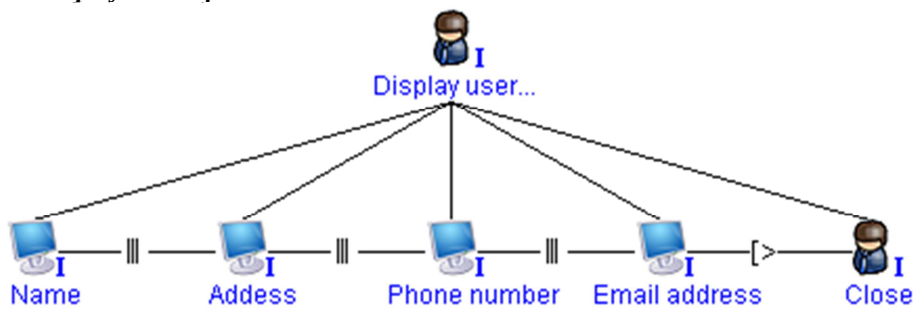


Task patterns

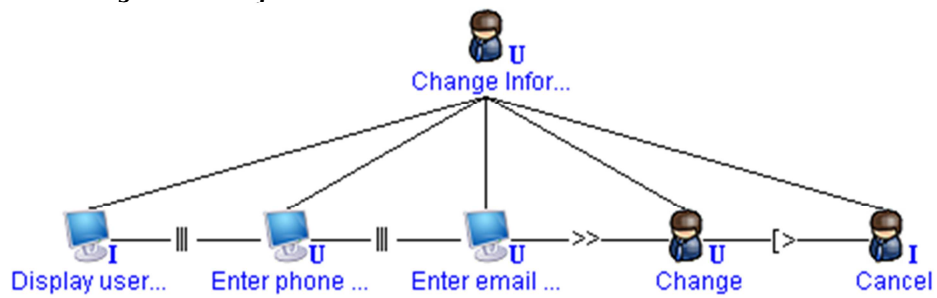
Search user



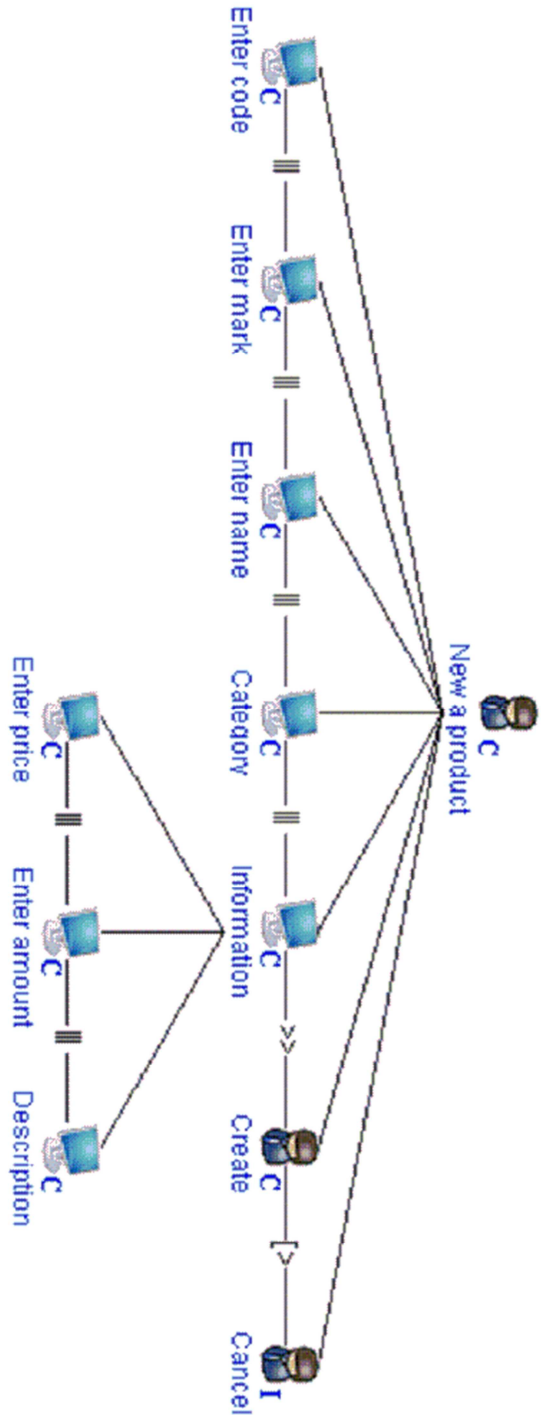
Display user information



Change contact information



Create a new product



Task patterns

Make an order

