



"Reverse Engineering of User Interfaces"

Bouillon, Laurent

Abstract

Reverse Engineering of User Interfaces

Document type : *Thèse (Dissertation)*

Référence bibliographique

Bouillon, Laurent. *Reverse Engineering of User Interfaces*. Prom. : Vanderdonckt, Jean



Reverse Engineering of Declarative User Interfaces

Laurent Bouillon

A dissertation submitted in fulfilment of the requirements
for the degree of

Doctor of Philosophy in
Management Sciences

of the Université catholique de Louvain

Committee in charge:

Prof. Jean Vanderdonckt, Université catholique de Louvain, Advisor

Prof. Bernard Fortz, Université catholique de Louvain, Examiner

Prof. Manuel Kolp, Université catholique de Louvain, Examiner

Prof. Jean-Luc Hainaut, Faculté Universitaire Notre-Dame de la Paix, Reader

Prof. Christophe Kolski, Université de Valenciennes et du Hainaut-Cambrésis, Reader

June 2006

TABLE OF CONTENTS

Chapter 1	The need for reverse engineering.....	13
1.1	Evolution of information systems.....	13
1.2	Reengineering of information systems	14
1.3	Importance of reengineering.....	17
1.4	New needs for reengineering	20
1.5	Objectives and working hypothesis.....	22
1.5.1	Aims and thesis statement.....	22
1.5.2	Working hypothesis and methodological choices.....	23
1.5.3	Reading Map.....	26
Chapter 2	A Conceptual Framework for UI Reverse Engineering	28
2.1	The Multi-context Framework.....	28
2.2	Reengineering Definitions	31
2.3	Retargeting	34
2.3.1	The retargeting concept.....	34
2.3.2	Alternatives during the abstraction process	40
2.4	Conclusion	42
Chapter 3	State of the art in reverse engineering 44	
3.1	Reverse engineering of legacy systems.....	45
3.1.1	Morph.....	46
3.1.2	The AUIDL Environment	49
3.1.3	Cellest - Mathaino.....	50
3.1.4	Cobol UI reengineering for Windows	53
3.1.5	PUC	54
3.1.6	Visual TAP.....	55
3.2	Reverse engineering and reengineering of the Web.....	57
3.2.1	Digestor.....	57
3.2.2	Mobile Transparent access	58
3.2.3	PIMA	60
3.2.4	Webrevenge	61
3.2.5	ReWeb	63
3.2.6	WARE	64
3.2.7	TAMEX	66
3.2.8	XWEB	68
3.2.9	Adaptation of Web Pages.....	69
3.2.10	Transcoding HTML to VoiceXML.....	71
3.2.11	Reverse engineering End-User Web Applications.....	72
3.2.12	Revangie	73

3.2.13	AWT2XIML.....	74
3.3	Comparison of the approaches.....	76
3.4	Location in the Framework.....	79
Chapter 4	Notation for reverse engineering derivation rules.....	85
4.1	Introduction.....	86
4.1.1	Objectives and method.....	86
4.1.2	Mapping and notation definition.....	87
4.1.3	Meta-model definition.....	89
4.2	Mathematical definitions.....	90
4.3	Operations on trees.....	91
4.3.1	Basic operations on T_t	92
4.3.2	Derived operation (on T_H , T_W , T_V and T_T):.....	95
4.3.3	Remarks.....	98
4.3.4	Groups of operations.....	99
4.4	Inter-tree mappings.....	101
4.4.1	Initialization.....	102
4.4.2	Particular to node classes.....	103
4.4.3	General rule defining alignment.....	106
4.4.4	General rules applied for each node.....	106
4.5	Rule implementation.....	111
4.6	Conclusion.....	112
Chapter 5	Reverse Engineering of Web Sites: Vaquita.....	115
5.1	HTML history.....	115
5.2	Working hypotheses.....	116
5.3	HTML and XIML meta-models and derivation tables.....	118
5.4	Tool support and example.....	122
5.4.1	Vaquita.....	122
5.4.2	Example of a complete reengineering thanks to Envir3D.....	125
5.5	Shortcomings of Vaquita.....	129
5.6	Conclusion.....	129
Chapter 6	ReversiXML.....	125
6.1	Evaluation of Vaquita.....	125
6.2	Working hypotheses.....	126
6.3	USIXML meta-model and derivations rules.....	127
6.4	Tools support: ReversiXML.....	132
6.4.1	Description of ReversiXML.....	133
6.4.2	The reverse engineering process.....	136
6.4.2.a	Steps.....	136

6.4.2.b	Layout recovery	140
6.4.2.c	Dialog relations.....	142
6.4.3	Evolution of the architecture of the tool	143
6.4.4	Differences between the two tools.....	150
6.4.5	Ongoing issues and future work.....	151
Chapter 7	Reverse engineering of other markup-based UI.....	155
7.1	Reverse engineering of WML.....	156
7.1.1	The WML language.....	156
7.1.2	Working Hypothesis	158
7.1.3	Language meta-model and derivation rules.....	159
7.1.4	Tool support	162
7.2	Reverse engineering of VoiceXML 2.0.....	165
7.2.1	The VoiceXML language	165
7.2.2	Working Hypothesis	166
7.2.3	Language meta-model and derivation rules.....	168
7.2.4	Example	173
7.3	Conclusion	174
Chapter 8	Reverse engineering of resource files	175
8.1	Windows resources files.....	175
8.2	Working hypothesis	179
8.3	Derivation rules and resource meta-model.....	180
8.4	Conclusion	185
Chapter 9	Validation.....	187
9.1	Internal validation.....	188
9.1.1	Coverage of source language	188
9.1.2	Coherence.....	190
9.1.3	Performance.....	191
9.1.4	Correction.....	193
9.1.5	Conclusion for the internal validation.....	199
9.2	External Validation.....	200
9.2.1	Exploratory study of reengineering	201
9.2.1.a	Method.....	201
9.2.1.b	Results	204
9.2.1.c	Discussion	205
9.2.2	Case Study 1: the Sedan-Bouillon Web site.....	206
9.2.2.a	Introduction page.....	207
9.2.2.b	Index page	209
9.2.2.c	Order Documentation.....	211

9.2.2.d	Complete reengineering.....	215
9.2.3	Case study 2: UGC movie booking service.....	219
9.2.4	Case study 3: SNCB homepage.....	223
9.2.5	Conclusion for external validation.....	228
9.3	Comparison with the state of the art.....	229
9.3.1	General comparison.....	229
9.3.2	Comparison with a transcoding approach.....	231
9.3.3	Conclusion of the comparison with the state of the art.....	234
9.4	Conclusion.....	235
Chapter 10	Conclusion.....	237
10.1	Contribution.....	237
10.2	Discussion.....	240
10.2.1	Thesis statement.....	240
10.2.2	The reverse engineering sub-problems.....	241
10.2.3	Results of reengineering.....	243
10.2.4	Limits of the approach.....	244
10.2.5	Compliance with MDA.....	245
10.2.6	Integration with other works.....	247
10.3	Perspectives.....	248
References	253
Appendix A	267
Appendix B	272
Appendix C	282
Appendix D	314
Appendix E	317
Appendix F	321
Appendix G	322
Appendix H	326
Appendix I	344
Appendix J	350
Appendix K	352
Appendix L	355

TABLE OF FIGURES

Figure 1-1 Triggers for evolution of information system	14
Figure 1-2 The waterfall model	15
Figure 1-3 History of langages	18
Figure 1-4 Different languages for UIs development	21
Figure 1-5 History of markup-languages	25
Figure 2-1 The Multi-Context Framework	29
Figure 2-2 Four levels of abstraction	30
Figure 2-3 Reengineering definitions in the framework	32
Figure 2-4 Different representation for a button	34
Figure 2-5 Proliferation of UI descriptions	35
Figure 2-6 Retargeting in the Cameleon Framework	36
Figure 2-7 The transformation of intra-page links into extra-page links	42
Figure 3-1 The reengineering process with Morph	47
Figure 3-2 Morph's reengineering process	47
Figure 3-3 Morph's categorization of concepts	49
Figure 3-4 The reengineering process with AUIDL	49
Figure 3-5 The reengineering process with Cellest	51
Figure 3-6 The architecture of Cellest	52
Figure 3-7 The reengineering with Csaba's approach	54
Figure 3-8 Reengineering process with PUC	55
Figure 3-9 The reengineering with TAP	56
Figure 3-10 Visual TAP	57
Figure 3-11 Widgets recognition in Felsberg's approach	58
Figure 3-12 The transcoding approach with Digestor	59
Figure 3-13 Digested Web page	59
Figure 3-14 The mobile transcoding approach	60
Figure 3-15 The mobile tranparent approach	61
Figure 3-16 Transcoded Web page	62
Figure 3-17 The reengineering process in the PIMA project	63
Figure 3-18 The complete Pima process	64
Figure 3-19 The reverse engineering process of Webrevenge	65
Figure 3-20 Reverse engineering of the task model	66
Figure 3-21 The reweb reengineering process	67

Figure 3-22 The Reweb's Approach	67
Figure 3-23 The reverse engineering process with WARE	69
Figure 3-24 The Ware architecture	69
Figure 3-25 The reengineering process with Tamex	67
Figure 3-26 The Tamex mediation process	71
Figure 3-27 The Lopez's adaptation-transcoding approach	73
Figure 3-28 Adaptation of Web pages	74
Figure 3-29 The voiceXML transcoding approach	75
Figure 3-30 The VoiceXML Transcoder Architecture	75
Figure 3-31 Click's end-application reverse engineering approach	77
Figure 3-32 Revangie reverse engineering process	78
Figure 3-33 AWT2XIML reverse engineering process	79
Figure 4-1 Structure of the chapter	86
Figure 4-2 Subtree representing a graphicalTransition	99
Figure 4-3 Sequence of derivation rules	102
Figure 4-4 Hierarchy detection rules example	108
Figure 5-1 The meta-model of XIML	119
Figure 5-2 Vaquita in the cameleon reference framework	122
Figure 5-3 The complete reengineering process with Vaquita	123
Figure 5-4 Main screen of Vaquita	124
Figure 5-5 Options page of Vaquita	125
Figure 5-6 Creation of a control room with Envir3D	126
Figure 5-7 Control room generated from a XIML specification in Envir3D	126
Figure 5-9 Virtualization process	127
Figure 5-10 The reengineering of a Web Form	128
Figure 6-1 model composition in USIXML	133
Figure 6-2 Start screen of ReversiXML	139
Figure 6-3 Reverse Engineering options	140
Figure 6-4 The resulting abstract or concrete UI models	140
Figure 6-5 ReversiXML in the reference framework	141
Figure 6-6 The entire reverse engineering process	142
Figure 6-7 General layout of a Web page	146
Figure 6-8 Layout recovery	147
Figure 6-9 Example of layout recovery	148
Figure 6-10 Link table	149
Figure 6-11 Coverage and complexity trade-off	154

Figure 7-1 Mobile devices with different capabilities	157
Figure 7-2 The meta-model of WML 1.1	160
Figure 7-3 Command prompt for the WML reverse engineering tool	163
Figure 7-4 Yahoo mobile login page	163
Figure 7-5 Preview UI generated with Grafxml	165
Figure 7-6 The VoiceXML 2.0 meta-model	169
Figure 7-7 Portion of the CUI used to represent vocal UI	170
Figure 8-1 The envisioned reengineering process	176
Figure 8-2 Resource script of a dialog box	177
Figure 8-3 Menu meta-model	180
Figure 8-4 Example of a menu in a resource script	181
Figure 8-5 Find dialog box resource script	185
Figure 9-1 The three web pages of the exploratory study	202
Figure 9-2 Reengineering in the exploratory study	203
Figure 9-3 Platforms and navigators accessing the site	207
Figure 9-4 Division into boxes	207
Figure 9-5 Index page of sedan-bouillon website	210
Figure 9-6 Choose documentation	212
Figure 9-7 Order documentation form	213
Figure 9-8 Results of the forward engineering with Teresa	215
Figure 9-9 First page of case study regenerated with QtXML	216
Figure 9-10 Second page of case study regenerated with QtXML	216
Figure 9-11 Third page of the case study regenerated QtXML	217
Figure 9-12 Fourth page of the case study regenerated QtXML	218
Figure 9-13 City selection page on UGC reservation site	219
Figure 9-14 SNCB Homepage	223
Figure 9-15 AUI model of the case study in IdealXML	227
Figure 9-16 ADAPT's navigation patterns	231
Figure 9-17 Navigation patterns on the CNN Website	232
Figure 10-1 Comparison of the reference framework and MDA	246
Figure 10-2 Integration with other researches	247

TABLES

Table 1-1 Windows PC/ Windows CE sales projection	21
Table 2-1 Comparison between reverse engineering and retargeting	39
Table 2-2 Alternatives in the transformation of images.	41
Table 2-3 Different alternatives for text links.	41
Table 3-1 Comparison of the re(verse)engineering approaches	83
Table 5-1 Derivation table for images	121
Table 5-2 Derivation table for radio buttons	121
Table 5-3 Derivation table for edit boxes	121
Table 6-1 Differences between Vaquita and ReversiXML	151
Table 7-1 Reengineering of Yahoo mobile login page	164
Table 7-2 Tags ignored during reverse engineering	168
Table 9-1 Performance analysis	192
Table 9-2 Error rates for the correction analysis.	196
Table 9-3 Limitations of ADAPT and ReversiXML	233
Table 9-4 Size reduction with ADAPT and ReversiXML	234
Table 10-1 Coverage of the reverse engineering tasks for the different source languages	241
Table 10-2 Complete Reengineering	243

Acknowledgements

I would like to express my thanks to:

- My advisor, Professor Jean Vanderdonckt, for his constant support and enthusiasm regarding my work.
- Professors Bernard Fortz, Jean-Luc Hainaut, Manuel Kolp and Christophe Kolski, for accepting to participate to the jury of this dissertation.
- My colleagues from IAG school of management at Université catholique de Louvain.
- My family and friends.

This dissertation was realized thanks to the support of:

- Cameleon research project (<http://giove.cnuce.cnr.it/cameleon.html>) under the umbrella of the European Fifth Framework Programme (FP5-2000-IST2).
- the SIMILAR network of excellence (<http://www.similar.cc>), the European research task force creating human-machine interfaces similar to human-human communication of the European Sixth Framework Programme (FP6-2002-IST1-507609).

Chapter 1 The need for reverse engineering

1.1 Evolution of information systems

Information systems is “a means of recording and communicating information to satisfy the requirements of all users, the business activities they are engaged in and the objectives established for them”[Olle88]. Information systems should perpetually evolve because of the evolution of three categories of parameters defining their context of use. A context of use can be defined by the user, platform and environment of the information system. These evolving parameters are:

- The evolution of the *user* means modification of parameters of the information system to satisfy user needs. For example, when a new stereotype of users will use the system, it has to be modified according to their level of experience. Another example is the evolution of a system designed for a small number of people in order to be able to manage a great number of users.
- The evolution of *platforms* on which these information systems run (figure 1-3 illustrates the variety of platforms and languages). Here we define a platform by the combination of hardware and software features, thus including the language used to develop the information system. An example is the migration of the back-office system to an Internet version of the information system.

- The evolution of the physical *environment* (including the socio-organizational factors). An example of this type of evolution is the transformation of an information system after a merging between two organizations, by changing its features to cope with the new management procedures. Other examples are law modification (e.g. a tax rate modification) implying an adaptation of the system to take these evolutions into account.

These dimensions have different rates of progression over time (figure 1-1), the change of requirements due to evolution in the environment seldom occurs. Appearance of new platforms has the fastest pace of progression, as several new types or variation of existing platforms appear every year. New development languages or new versions of existing languages appear also frequently. The last incentive for the evolution of information system, the appearance of new user needs appear regularly, more often than evolution of environments, but less frequently than new computing platforms.

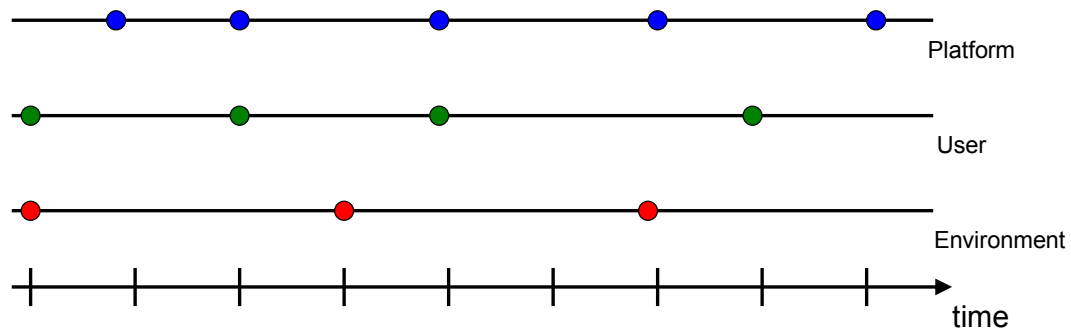


Figure 1-1 Triggers for evolution of information system

This constant need for evolution exists since the birth of information systems, and will ever be present, thus there will always be 'old' systems. To cope with this inherent problem, reengineering can be applied to redesigning the information system and in some cases, recovering the structure and some parts of the system that has to be modified rather than start the implementation of a new system from scratch, and so lower the costs and the efforts devoted to the evolution of the information system.

1.2 Reengineering of information systems

Reengineering is a crucial problem for information systems, as estimates of the resources devoted to maintenance in software development range from 50 to 80% [Boeh81].

Chapter 1 The need for reverse engineering

Software maintenance is the modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment [Sem03]. It is estimated that 40 to 60% of maintenance effort is devoted to the understanding of the software to be modified. The automation (or semi-automation) of the abstraction of the parts of a system could thus save lots of resources.

A software can be divided into two parts: the functional core which contains the various functionalities of the application and the user interface which gives access to these functionalities, presents the results obtained with the software, etc. The user interface (UI) code is generally extremely dependent on the platform and language on which it has been coded, and has, most of times, to be rewritten completely when the software needs to be transformed for another platform. Moreover, the part of the code dedicated to the UI can reach in average 50% of the total code size of a software [Myer92]. For a classical information system, this portion is rather representative. For other systems, this portion can largely vary from 10% (e.g. in a scientific calculation intensive application) to 80% (e.g. for a word processor).

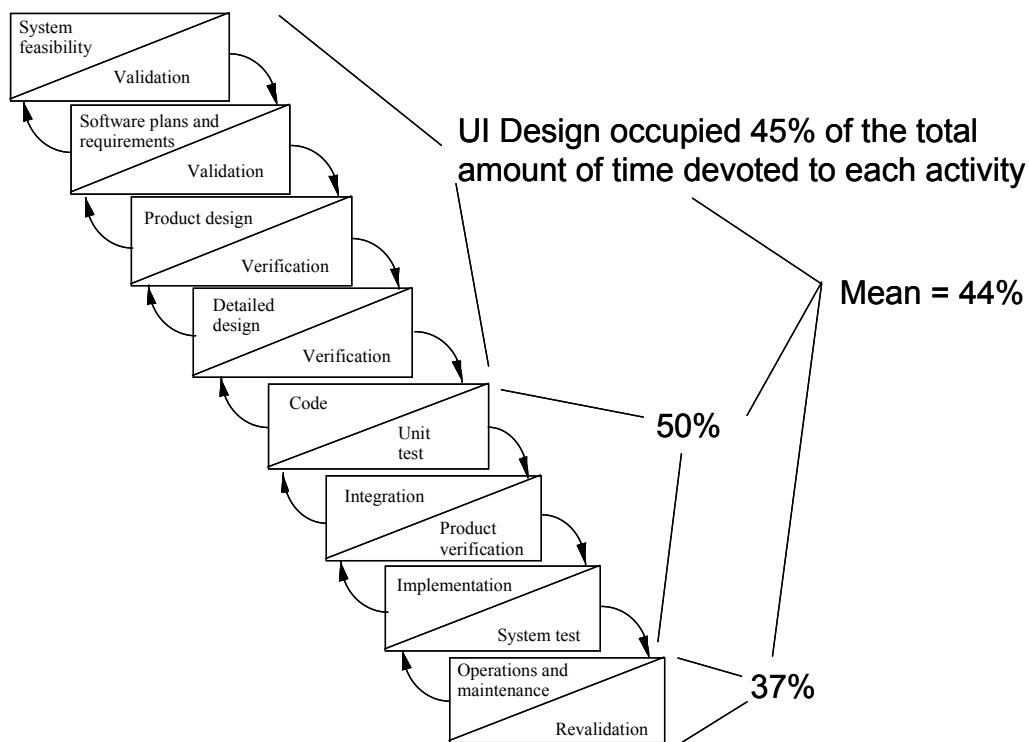


Figure 1-2 The waterfall model

Chapter 1 The need for reverse engineering

Another study [Boeh88] demonstrated that the UI design and implementation occupied 44 % of the total time devoted to each activity in the development cycle of a software (figure 1-2). 45% percent of the time needed to make the preliminary analysis (feasibility, requirements, design) was devoted to UI, 50 % of the coding time was needed to develop the UI, and 37% of the total time for the testing and maintenance. As the resources needed for UI development are often underestimated, this shows the importance of the development of techniques for reengineering UI as the conception of this part of the system consumes almost the half of the time to develop an entire information system.

We consider that reengineering of a software (and consequently of its two parts, the functional core and the UI) can be decomposed into two successive steps: the reverse engineering and the forward engineering. The reverse engineering of a UI consists in the examination of the current implementation in order to extract an abstract representation and the forward engineering step is the generation of a new UI based on this abstraction.

Reengineering thus involves the reverse engineering of source code, which is the subject of this thesis. Chikofsky and Cross [Chik90] give the following definition: “Reverse engineering is the process of analyzing a subject system to identify the system’s components and their interrelationships and create representations of the system in another form or at a higher level of abstraction”.

Reverse engineering can have different objectives and thus produces different outputs. We can distinguish:

1. Reverse engineering of the domain and data. Several works have already been conducted on this subject [Hain95, Embl98, Estie03, Cres04, Gott04, Astr05]. The aim is to reconstruct the domain model, data structure or/and databases used by an information system. The points of interest here is not to reconstruct the information system as it exists, but only to keep the data, which is the true asset of this type of system, and from this, either document the system for future enhancements or build a new system based on the “old” data structure.
2. Reverse engineering of the task model [Paga02]. In this case, only the sequence of tasks is recovered. This is a reverse engineering at a very high level of abstraction as no information about the composition of the UI or

data processed in the information system/UI are recovered at this level of abstraction.

3. Reverse engineering of the UI (see chapter 3). In this case, the presentation (components and structure of the UI) and dialog (behaviour of the components) are captured in a model.

The reverse engineering of the domain and the task models has been already addressed in a significant manner: this will be one of the reasons explaining why we will focus on reverse engineering of UIs. There are several benefits for reengineering:

- **Practical benefits:** Software development productivity is improved if programs can be enhanced instead of being rebuilt. It is then possible to modify an information system for previous stages in the development process. Major pieces of existing systems can also be reused with reduced efforts. It can also enable tools to debug or improve older systems and so develop information system quality [Rug94].
- **Economical benefits:** Costs related to the complete reconstruction of huge legacy systems can be unaffordable. A generalization and automation of the process of reengineering would save lots of resources. J. Salisin showed that the techniques of reengineering get up to 35 % of savings on the total of the costs related to maintenance [Sali92].
- **Theoretical benefits:** Reengineering has often to be used with old systems that have not been developed following a structured approach. Reengineering helps to restructure the application, by using models and producing documentation, and so reinstates the system in a more comprehensible, controllable and structured implementation.

1.3 Importance of reengineering

A *legacy system* is an information system or application program, which continues to be used because of the prohibitive cost of replacing or redesigning it and despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic and difficult to modify [Fold98].

The number of applications written in obsolete languages or systems is incredibly high: it is estimated that 60% of all the code in the world is legacy code. Therefore,

Chapter 1 The need for reverse engineering

maintenance is the major source of revenue of development/maintenance agencies. For example, let us consider the COBOL language. The archaism of the COBOL language is shown on the diagram of programming languages history (figure 1-3). Despite this fact, it is still extremely used in organizations [Kapo03]:

- 2.4 millions COBOL programmers maintain 9.3 million COBOL application.
- There are about 200 billion “active” lines of COBOL code in the world.
- The increase rate of COBOL lines is about 5 billion lines/ year.

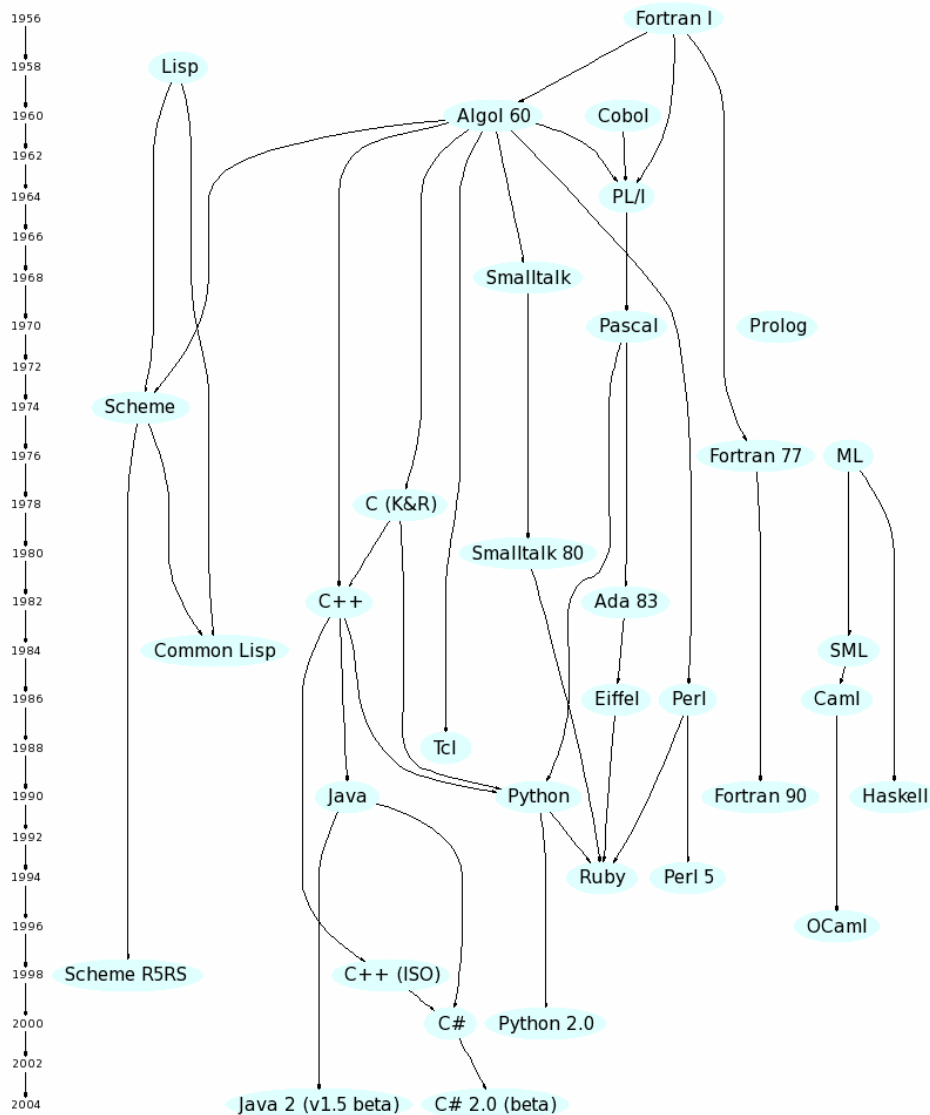


Figure 1-3 History of languages [Pixe]

Chapter 1 The need for reverse engineering

Legacy systems often encapsulate implicitly years of business-process experience and evolution. They contain vital business data and thus represent valuable assets for organizations. Therefore, organizations usually prefer to apply “patches” to the existing system instead of changing the system completely (4 out of 7 programmers are engaged in maintaining existing software systems). Maintenance is often preferred to a complete reconstruction for two reasons.

1. Firstly, the organization fears to lose value during the conversion as systems are loaded with diverse business policies and corporate decisions [ElRa00].
2. Secondly, it is easier to take the decision to gradually spend some parts of the Information Technology budget for the maintenance of the system rather than to engage considerable funds for a complete reconstruction. Moreover, it could happen that the cost related to a complete reconstruction of the legacy system is unaffordable, or simply technically impossible.

Maintained systems have several shortcomings. Most of them are undocumented, they use legacy languages and skills (unsatisfactory performance) and their user interfaces are not of good quality (usability problems) or they are implemented in an old fashion (e.g. character-based UI). When they are documented, the documentation is limited or too partial to allow being reused. There are also indirect costs related to those poor and archaic UIs, as new users have to be trained longer to use the legacy system. As they do not have any formal architecture, the evolution, extension or modification of these systems is very hard, if not impossible. Furthermore, these kinds of applications are unable to benefit from the “Internet/Mobile technology”, which is currently a handicap compared to modern systems, but will become a true burden in the future. Finally, the quality of these maintained systems is being deteriorated continually over time, and when accumulated during long periods, requires enormous human resources [Kapo03].

All these factors stress the importance of developing methods for reengineering legacy systems. We can distinguish two main aims for the reengineering of old systems:

1. Modernizes a legacy application by adding features to benefit from the new possibilities of the target system (e.g., moving the application to a newer and faster platform, re-purposing the system, integrating the system with other legacy system, enabling web access to the system).
2. Transform existing (modern) applications to make them available on a variety of computing platforms.

The major focus of this document is on this second aim of reengineering (i.e. to transform an information system to another platform), but applied only to the UI part of an information system. Due to the focus on the platform, the next section will examine significant evolution of the platform to be considered.

1.4 New needs for reengineering

The technological push of newly marketed access devices (figure 1-4) - such as Personal Digital Assistants (PDAs) and Wireless Access Protocol (WAP) enabled cellular phones - has exacerbated the need to access the same web site, from different access devices [Boui02a]. With the demand for Internet access increasing for e-mail, electronic commerce, current events, and quick information, the need for Internet-capable access devices has extended beyond the professional desktop to the home office, to other rooms of the house or between office and home. Moreover, the same need appears for other types of UIs in order to take the benefits from a mobile access to information systems.

On this subject, some evocative figures can be quoted:

- Worldwide mobile devices growth is more than 45% between 2003 and 2004[Cana04].
- The market of the mobile devices with an Internet access was estimated at 1,77 billions US\$ in 2002 and converge with fixed terminal[Etfo03].
- Microsoft « mobile and embedded » annual revenue surges 58% for 2004[Can a04].
- Internet-Connected PDA will outnumber traditional PDAs by 2006[Cana04].
- 85% year-over-year worldwide growth for smartphones (400% for US) [Cana04].

The market for mobile platforms (e.g., PalmPilot, Pocket PC, laptop, mobile phone) is growing twice faster than the market for traditional desktop computers [Ber98]. Microsoft's sales projection (table 1-1) suggests that the mobile version of their Operating System (OS) should outnumber the classical PC version in 2008 [EtFo03].

As the expansion rate of mobile platforms is becoming greater and their capabilities have considerably improved, there is now a need to adapt the huge amount of UIs rapidly in order to make them accessible to a variety of contexts of use (e.g., different devices in different environments)[Boui02c].

Chapter 1 The need for reverse engineering

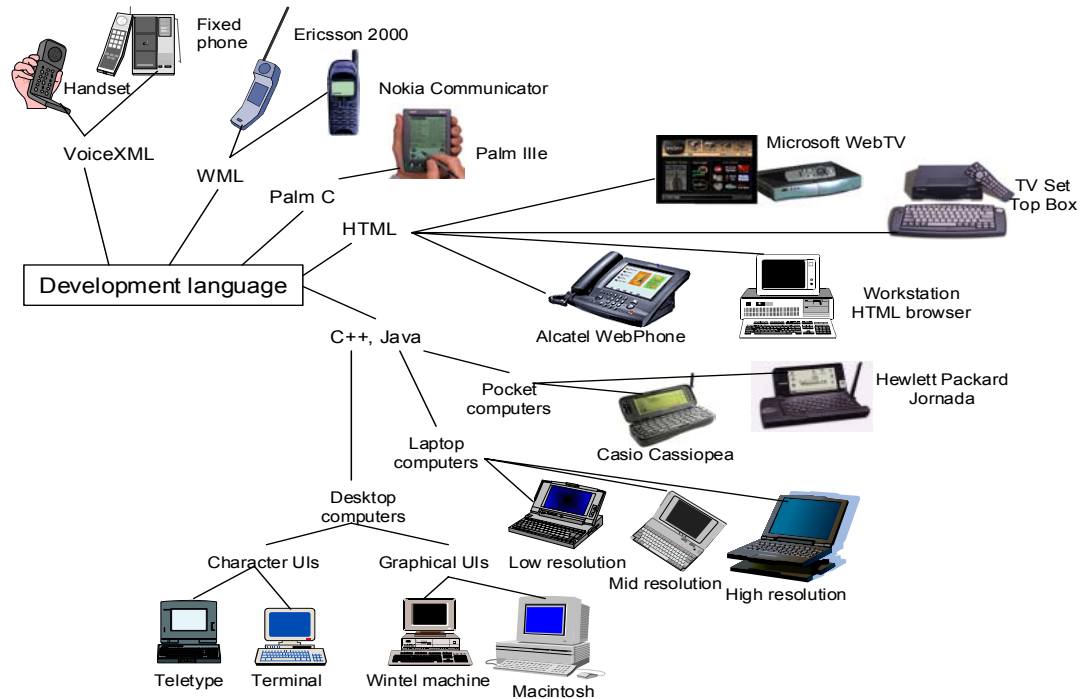


Figure 1-4 Different languages for UIs development

Sales Projections, in Millions of Units

	2002	2003	2004	2006	2008	2010
Worldwide						
Windows PCs	126	135-138	145-150	170-175	190-200	215-225
Windows CE Devices	9	14-17	30-35	105-115	200-220	300-340
USA						
Windows PCs	41	42-44	45-47	49-52	52-55	55-60
Windows CE Devices	3	4-5	6-8	19-22	35-40	55-60

Table 1-1 Windows PC/ Windows CE sales projection[EtFo03]

To address these demands, ad hoc development is no longer considered acceptable in terms of the cost and time required for software engineering, development and maintenance. There are several existing methods to ensure the cross-platform use of a UI. *Portability* of a UI generally refers to the capability of a UI to be ported from one computing platform to another with an effort that remains minimal. Galaxy (www.ambienca.com) enables developers to design a UI on one platform, say MS

Windows, and to export it without any change to other platforms, e.g., Mac OS X and Linux. The individual UIs then adhere to the “look and feel” that is proprietary to these respective platforms. The advantage is consistency but the drawback is that the UI does not take into account specific properties of each platform. In addition, each produced UI does not take any benefit from the interaction capabilities of each platform (or restrictions) since the UI basically remains the same. The development of a UI does not frequently envisage the future desire for portability [Boui04]. Consequently, when the need arises to port a UI from one platform to another, it is rather difficult to support this porting. Developers do not necessarily want to start from scratch again to design a UI for a new platform since a UI already exists that could be a potential source of inspiration, if not a starting point.

In this second case, *transcoding* tools automatically transform a UI code from the original platform to a new UI code for the target platform. This transformation can occur at design-time (i.e., the transformation is made only once and re-inserted in the new platform) or at run-time (i.e., the transformation is performed on demand when the UI is requested). Any HTML page can be transformed into a WML (Wireless Markup Language) deck of cards on the fly when the mobile phone user accesses a web page [Kaas00].

Portability and transcoding tools suffer from shortcomings: the former only produce the same UI layout for all platforms while the latter only apply code to code transformations that are peculiar to any couple (source platform, target platform). Both remain rather inflexible (no design alternatives), uncontrolled (no human intervention to fine tune the transformation), and very specific (hard to generalize to other couples). They do not necessarily consider constraints imposed by the target platform such as: operating system, programming language, screen resolution or interaction capabilities.

1.5 Objectives and working hypotheses

1.5.1 Aims and thesis statement

The aim of this research is to apply the reverse engineering process of existing UIs thanks to a model-based approach in a way that is as much automated as possible as to minimize resources required for this process (time, costs, persons). As the subject of this thesis is reverse engineering, this research is embedded in a larger process, and the results of the abstraction obtained by this approach will be reused in a forward

engineering phase to regenerate UI code, thus achieving a complete cycle of reengineering. Forward engineering is supported in several other works [Fole91, Luo93, Szek92, Wils93, Vand94, Vand95, Szek96, Puer96, Puer97, Olse00, Thev99, Mori03].

The thesis statement for this dissertation is that *the application of the reverse engineering at a higher level of abstraction than the code level supports UI reengineering with flexibility while preserving predictability, more generality and controllability in the process than with code-to-code (transcoding) approaches or current reverse engineering approaches.*

The particularity of this approach lies in its *flexibility*, the fact that the reverse engineering can be adapted according to the context of use. Thanks to a flexible approach, the process can be modified following targeted platforms while ensuring *predictable* results. Therefore, a set of reverse engineering techniques and heuristics is available and designers can *control* the selection of the most suitable techniques to be applied during the reverse engineering. The flexible techniques and heuristics on which the approach is based have been formalized and will be expanded, to enhance *generality* of the approach by considering alternative design options and cover the maximum number of transformations of the UI model.

1.5.2 Working hypothesis and methodological choices

The approach developed in this thesis is characterized by the following methodological choices:

Model Based Approach

To overcome most shortcomings of transcoding identified in section 1.4 and to address the needs of UI portability, we argue that a UI reverse engineering process can be combined with UI forward engineering process to produce not only more usable UIs in a logical way, but also to benefit from the reverse engineering to port a UI to any other target platform. In a model based approach [Szek96, Puer96, 97], it is possible to generate several UI for different platform (or contexts of use) thanks to a single specification. Instead of having a UI specified in m source languages to translate into n different target languages ($m \times n$ translation tools), the model-based approach reduces this number to m plus n translation tools ($m + n$) as each of the n languages need one translator to models, and conversely, each of the m languages requires one translator from model to the language.

Model-based approaches (MBA) can produce a user interface in a forward or reverse engineering manner by exploiting knowledge captured in various models, such as task [Kong99], domain [Deba95], presentation [Moor96] and dialog [Merl93] models. However, most existing UI have been developed without any model-based approaches. The goal of this work is to address this problem by examining how a user interface description can be reverse engineered to migrate it to another environment. Finally, we also want to work towards making the process compatible with the Model-Driven Engineering (MDE) as recommended by the Object Management Group (www.omg.com), as it represents a major trend in software engineering.

Output language

The UsiXML (User Interface eXtensible markup Language – <http://www.usixml.org>) language [Limb04a] has been chosen to express the abstract representation of the UI and this for three main reasons: firstly, it is able to represent most aspects of interest of this thesis (context description, large list of interaction objects, etc...). Secondly, it is license-free language and can thus be used without restriction, such as extension (this is not the case with other markup languages such as UIML or XIML [Puer01]). Thirdly, a set of tools is already available, thus allowing us to test the results of the reverse engineering by combining them with a forward engineering tool.

Context limited to the platform

The goal of this thesis is to develop a reverse engineering process to allow “mono-context” user interfaces to be used in different contexts of use. In the reminder of this work, the context of use will focus on the computing platform as this parameter has the greatest impact on UI when a change of context of use occurs. In the general definition of a context of use [Calv03], the context of use is defined as $C=(U,P,E)$ where U denotes the end user, P denotes the computing platform used by the end user, and E denotes the physical environment in which the end user is carrying out her interactive task.

Scope

The most deeply investigated type of UI is web pages of information-centric Web sites. The scope of this research is not only Web UI, but it focuses particularly on the reverse engineering of standard HTML 4.0 Web pages as this is the most widespread type of UI. The particularity of the HTML language is that it is a declarative language (the code is composed of a set of declaration, which are interpreted) and the source

Chapter 1 The need for reverse engineering

code is directly accessible, without decompilation. The reverse engineering of this type of languages is easier than imperative languages (such as C++ or Delphi) in which the code is composed of a set of instructions and is generally compiled.

But HTML is not the only UI language that has been analysed in this research. In chapter 7 the reverse engineering of UI expressed in another modality (VoiceXML) is treated. The seventh chapter contains also the description of a method for the reverse engineering of WML (mobile phones UI). The reverse engineering of Windows UI is the subject of chapter 8. This has been done by adding new reverse engineering rules or adapting existing techniques developed for HTML UI. Figure 1-5 shows the history of the most widespread markup languages. The mark-up languages analyzed in this thesis are surrounded with a dashed line.

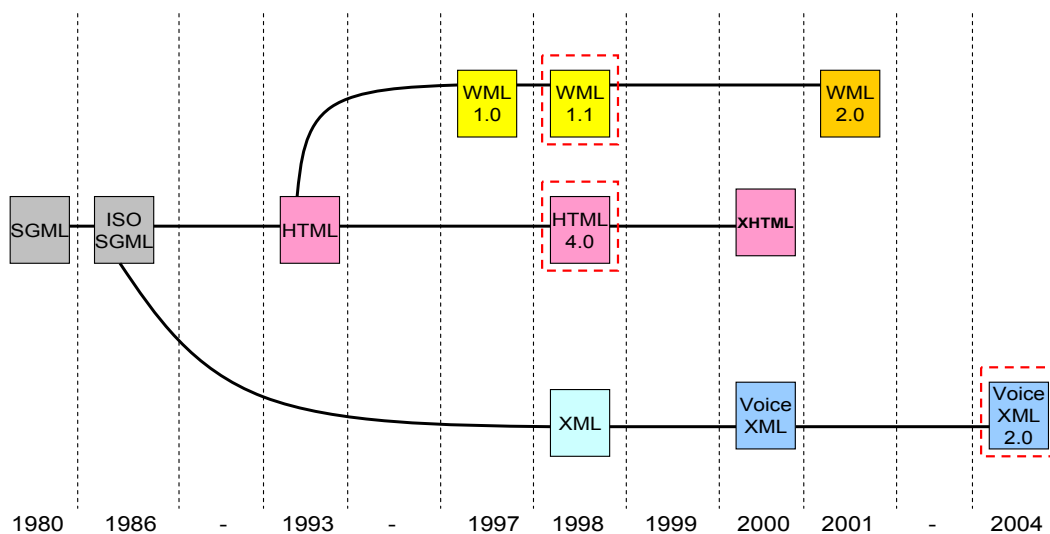


Figure 1-5 History of markup-languages

This research focuses on the reverse engineering of the declarative portion of the UI specification. This implies a limited recovery of the dialog properties of UI as an important part of the behaviour or dynamic effects are coded thanks to scripts and imperative languages (i.e. JavaScripts for HTML, WmlScripts for WML, EcmaScripts for VoiceXML or an imperative language such as C++ for windows UI). However, some dialog components can be specified declaratively and thus the dialog can be partially recovered.

Design time and run-time applicability

The approach proposed in this research is developed so that it can be used in two different manners: either manually by a designer who can control the reverse engineering process with some accuracy, or automatically without any human intervention, but with some loss of precision and quality.

Multiple level of abstraction

Another goal of this research is to allow designers to reverse engineer UI at multiple levels of abstraction on demand. Typical levels include device independence (i.e. obtaining a UI model that is independent from any interaction device), computing platform independence (i.e., obtaining a UI model that is independent from any computing platform), and modality independence (the system should also produce a UI model that remains independent of any modality of interaction). At this level, there should be no reference to the peculiar interaction techniques and modalities (e.g., character UI, graphical UI, vocal UI, and virtual UI) [Boui05].

Static analysis

The reverse engineering applied in this research consists in a static analysis of the source code, i.e. the derivation rules are applied without executing the code.

Validation

Finally, the validation of the thesis can be divided into two different aspects, the internal and the external validation. Internal validation is a theoretical analysis of the proposed approach, achieved in this document by evaluating the correctness, coherence, performance and coverage of the approach. The second part, the external validation, is an evaluation of the quality of the UI reengineered thanks to this approach. The produced models have to be regenerated to be able to evaluate the new UI. As it would imply the combination of another tool/approach with the current one and thus biases the result of the investigation, a qualitative analysis of the produced model and the tool has been preferred. This was done by conducting an exploratory study completed by case studies.

1.5.3 Reading Map

The remainder of this document is structured in ten chapters.

The next chapter is decomposed into three parts: the first is the description of the conceptual reference framework in which this research takes places. The second part is about the adaptation of reengineering definition to the UI domain and their

Chapter 1 The need for reverse engineering

location the framework. The third part of this chapter describes the retargeting concept, one of the contributions of this thesis.

The third chapter contains the related work on reverse engineering and reengineering of UIs. The first part of this chapter is dedicated to legacy system reverse engineering, and the second part contains approaches for Web reverse engineering.

The fourth chapter presents another contribution of this thesis, a semi-formal notation to express the reverse engineering rules. Firstly, a description of this notation is given and secondly some examples of reverse engineering rules for different languages are explained. The complete reverse engineering rules are given in appendix C.

The fifth and sixth chapter describes two tools developed to support retargeting of HTML Web pages. Each chapter contains a large description of the tool, an analysis of the strength and weaknesses of the tools, an illustration of some derivation rules, future possible enhancements and some examples of reverse engineering and reengineering.

The seventh chapter is dedicated to the reverse engineering of two other markup languages, WML and VoiceXML. This chapter is divided into two parts, the first describes the Wireless Markup Language, the structure of the language, reverse engineering methods and rules, and the presentation of a reverse engineering prototype for the reverse engineering of WML UI. It also contains an example of a reengineering thanks to models extracted with the tool. The second part contains also a description of the VoiceXML language, its structure, the illustration of several derivations rules for this language. Examples of reverse engineering can be found in appendix H.

The eighth chapter follows the same structure as for the seventh chapter, but for another type of UI, windows resource files. The chapter contains a description of the language and its structure, so as methods and rules to recover an abstract model from this language. A dialog box is also reverse engineered in appendix H to illustrate the method developed in the chapter.

The ninth chapter contains the internal and external validation of this thesis and the last chapter concludes this dissertation.

Chapter 2 A Conceptual Framework for UI Reverse Engineering

After the presentation of the reengineering problem and the incentives for reverse engineering, this second chapter defines some terms and concepts so as a conceptual framework used throughout this thesis. The first section presents the multi-context framework and the position of the reverse engineering process in this framework. The second section contains several software-reengineering terms adapted to the reengineering of UI. The third section introduces a new concept in reverse engineering, the retargeting of UI, and locates this process in the conceptual framework.

2.1 The Multi-context Framework

The Cameleon multi-context reference framework [Calv01,02,03] locates UI development steps for context-sensitive interactive applications. A context is defined as a triple of the form $C=(U, P, E)$ where E is an element of the environments set considered for the interactive system, P is an element of the platforms set considered for the interactive system and U is an element of the users set for the interactive system. As shown on figure 2-1, the framework structures development for two contexts of use, here for two platforms: the one on the left represents the source and the one on the right represents the target.

The development process can be decomposed into four steps:

1. *Task and concepts*: describe the various tasks to be carried out and the domain-oriented concepts as they are required by these tasks to be performed.
2. *Abstract UI*: a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent of the concrete interactors available on the targets. The elements used in the abstract UI are abstractions of existing widgets.
3. *Concrete UI*: concretizes an abstract UI into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. This interface is now composed of existing UI widgets.
4. *Final UI*: The UI produced at the very last step of the reification process is supported by a multi-target development environment. It is expressed as source code.

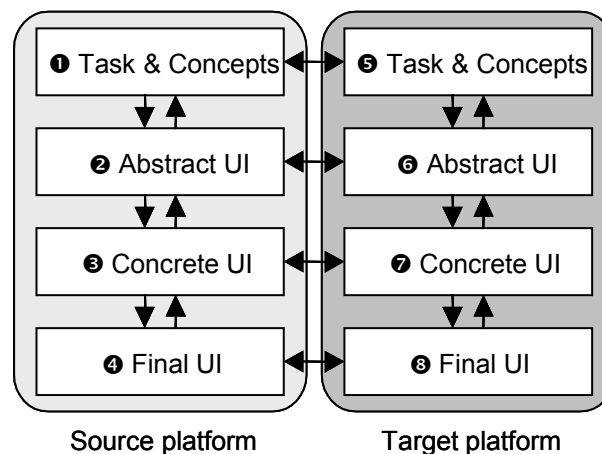


Figure 2-1 The Multi-Context Framework

The downward arrows on figure 2-1 represent reification steps (forward engineering), from the more abstract to the operational interface. Reification is the transformation of a description (or of a set of descriptions) into a description (or a set of descriptions) whose level of abstraction is lower than that of the source one(s). In the multi-target reference framework, it is the inference process that covers the inference process from high-level abstract descriptions to run-time code.

Upward arrows stand for abstraction steps. This process transforms a description into a description whose semantic content and scope are richer/higher than the content and scope of the initial description content. In the context of reverse engineering, abstraction is the elicitation of descriptions that are more abstract than

Chapter 2 A Conceptual Framework for UI Reverse Engineering

the descriptions that serve as input to this process. Finally, horizontal arrows correspond to the translation of the interface from one type of platform to another, or more generally, from one context to another. The complete definitions of the terms used in this framework can be found at <http://giove.cnuce.cnr.it/cameleon/glossary.html>.

Not all steps should be achieved in a sequential ordering dictated by the levels. Instead, locating **what** steps are performed, when, from which entry point and toward what subsequent step is important. In figure 2-1, transcoding tools start with a final UI for a source platform (4) and transforms it into another final UI for a target platform (8). Similarly, portability tools start with a concrete UI for a source platform (5) and transform it into another concrete UI for a target platform (7), that in turn leads to a new final UI for that platform (8). To overcome shortcomings identified for these tools, there is a need to raise the level of abstraction by working at the abstract level. *UI Reverse Engineering* abstracts any initial final UI (4) into concepts and relationships denoting an abstract UI (2), which can then be translated into a new abstract UI (6) by taking into account constraints and opportunities for the target platform. *UI Forward Engineering* then exploits this abstract UI (6) to regenerate a new UI adapted to this platform, by recomposing the concrete UI (7) which in turn is reified in an executable UI (8).

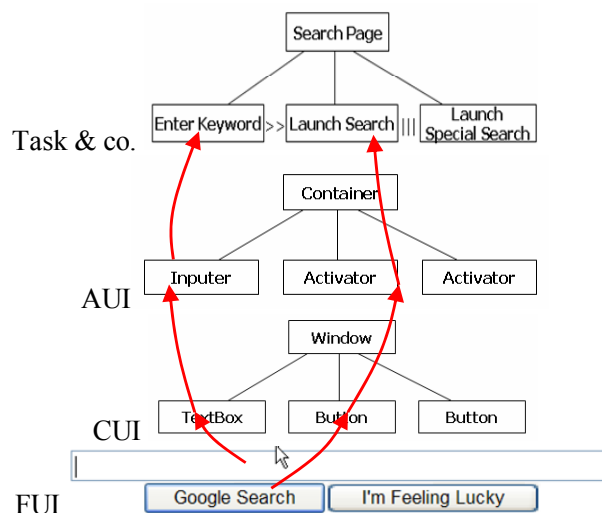


Figure 2-2 Four levels of abstraction

Example on figure 2-2 illustrates the four levels of abstraction of the framework on a part of the Google UI. The final UI corresponds to the sub-part of the Google UI as it is presented to the user. The same excerpt at the concrete UI level is represented

thanks to a window containing a textbox and two buttons. At the abstract UI level, the textbox is abstracted by an inputer and the buttons as activators embedded in a container. Finally, at the tasks and concepts level, the UI can be described by a principal task “Search Page”, which is decomposed into three subtasks, the “enter keyword” task followed by either the “launch search” task or “launch special search” task.

2.2 Reengineering Definitions

In this section, some definitions of software engineering [Chik90] [Geor01] are adapted to UI domain so that those definitions can fit into Cameleon reference framework (figure 2-3) [Calv03].

Reengineering is the examination and the alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. In the UI domain, it consists in a transformation of the UI at a higher level than the code to adapt it for another context of use. The reengineering of a UI thanks to a model-based approach (MBA) – which is the method used in this document – is composed of two phases : reverse engineering which recovers a specification at a higher abstraction level than the code and forward engineering which produces a new UI code (or less abstract specification) based on an abstract description of the UI. It can be applied at any level from the reference framework.

Recoding is defined as a change to implementation characteristics such as language translation and control-flow restructuring (source code level changes), conforming to coding standards, improving source code readability, renaming program items, etc.

Reformatting is defined as the functionally equivalent transformation of source code that changes only the structure to improve readability. A synonym for reformatting is *Refactoring*. Refactoring is about improving the design of existing code. It is the process of changing a software system in such a way that it does not alter the external behaviour of the code, yet improves its internal structure. Refactoring is a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behaviour. These three operations (recoding, reformatting and refactoring) occur at lowest level, the final UI level.

A term often used with different meanings in the UI reengineering literature is *transcoding*. In our approach, we define *transcoding* as the modification of the UI code

to adapt it for another platform. It is opposed to reengineering (reverse and forward) as no abstraction process is used during the transcoding. A representative example of a transcoding approach is IBM Websphere [Brit01]. Transcoding differs from recoding because the first is only used to modify the UI while the latter modifies the functional core of the application. As these last definitions are applied at the code level, they are situated at the final user interface level on the framework (Final UI, between ④ and ③ on figure 2-1).

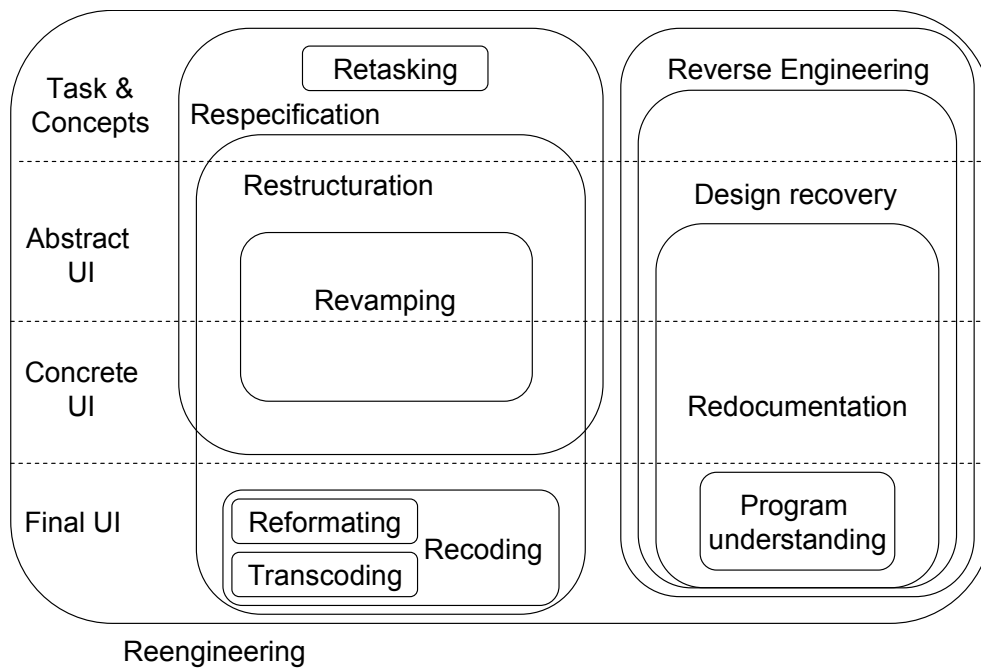


Figure 2-3 Reengineering definitions in the framework

Revamping consists in changing the UI without modifying the core application, which, in fact, ignores the changes operated at the terminal level. Revamping makes it possible to considerably modify the look and feel of the user interface. But not only the UI presentation can be changed. Also the phrasing can be redefined, multimedia features can be added, or on-line documentation can be created. Revamping is mostly used for improving the aesthetics of an existing UI to make it more usable. It is applied at a higher level than a transcoding approach, on UI specifications.

Re-specification is defined as a change to requirement characteristics. This type of change can refer to a change of the form of existing requirements or to a change in the system requirements (e.g., adding new requirements, changing or deleting existing

Chapter 2 A Conceptual Framework for UI Reverse Engineering

requirements). This can be applied at every stage of the development process until the final UI.

Restructuration is the transformation from one presentation form to another at the same level, while preserving the subject's external behaviour (functionality and semantics). Therefore, the task, the functions (application model) and the domain models should remain identical, but the process can be applied on the three other levels.

Retasking is the translation of the context-dependent task specification from one context to another, at design time. For instance, if an application is to be used on both a PDA and a computer, the task on the PDA will be less complicated than on the computer (the task tree will be different). This definition applies to the most abstract level from the reference framework.

Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction. It can be applied at any level to recover a more abstract representation from the UI.

Program understanding is a related term to reverse engineering. Program understanding always implies that understanding begins with the source code (final UI level) while reverse engineering can start at a binary and executable form of the system or at high level descriptions of the design.

Design recovery is a subset of reverse engineering in which domain knowledge, external information, and deduction or fuzzy reasoning are added to the observations of the subject system to identify meaningful higher level abstractions. As reverse engineering, it applies to all the levels of the reference framework.

Redocumentation is a form of restructuring where the resulting semantically equivalent representation is an alternate view intended for a human audience. As restructuring, it applies to the three first levels of the framework.

2.3 Retargeting

2.3.1 The retargeting concept

In this section, a new approach is proposed to support the reengineering of UI, and more particularly HTML pages, into other formats thanks to a model based approach: retargeting. Retargeting [Boui2b] can be used to limit the number of different UI models produced by traditional reverse engineering combined with a translation, and helps the designer in selecting more appropriate models for a specific platform or category of platforms.

Several possible models resulting from abstraction and translation. An interface created for a particular platform can be abstracted at the concrete UI level as a collection of interrelated widgets called concrete interaction objects. They are said to be concrete because they are independent of the computing platform. If we consider the relation between a concrete interaction object (CIO) -platform independent- and an interaction object belonging to a final UI as of the type one-to-many (see figure 2-4), a reverse engineering process will produce one concrete model. When the need arises to translate the model to another context of use, several possibilities exist, as a widget can be translated by every concrete interaction object belonging to the same class of interaction object having the same behavior. For example, a group of radio buttons can be translated by a group of checkboxes where only one choice is possible, a drop down list box, a simple list box, a text-box, a group of buttons or can still be represented by a group of concrete radioButtons. Translations are thus of the type 1-m.

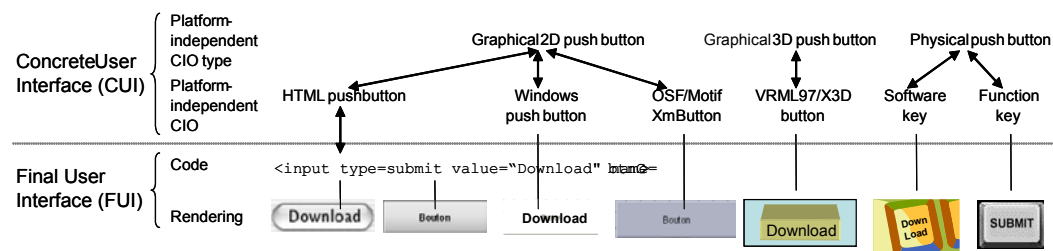


Figure 2-4 Different representation for a button

Two major constraints reduce this number of possible results: the domain and the platform. The domain contains the data that has to be displayed on CIOs and therefore it restricts the number of possibilities, by the fact that a CIO cannot represent every type of information. The “nature of a domain object is the determining factor in how to present and make the object available to the user” [Puer99]. The second constraint is the platform, on which a predefined set of CIO is

available (e.g. there are no checkboxes or radio buttons on WML-enabled phones, these objects have to be represented by list boxes with one or more selectable items). Further (weaker) constraints can later be used in order to select the most appropriate interaction object, such as the space consumption, the ease of use, the “look and feel” of the application. In our approach, as we reverse engineer a concrete or abstract UI specification, we do not have information about the domain model, but it is possible to reverse engineer an interface for a pre-selected computing platform and so limit the number of possibilities. This new type of operation, retargeting, was introduced in the reference framework (figure 2-6).

Reengineering by composing an abstraction followed by a translation. This classical approach consists in going up to the concrete or abstract interface point (from level ④ to ② or from ④ to ③ on figure 2-1), and translate it in another abstract interface specialized for a particular platform (from ② to ⑥ or from ③ to ⑦ on figure 2-1). For the translation step, there are two possibilities: either the target platform is more constraining than the source platform, and in this case the translation will reduce the number of possibilities (e.g., the translation from a PC UI into a mobile phone UI), or the target platform is less constraining, and the translation will increase the number of possible UI description (e.g. the translation from a Palm UI into a Macintosh UI).

These adapted abstract interfaces are then reified (from ⑦ to ⑧ or ⑥ to ⑧ on figure 2-1) in order to obtain UI code for another platform in the forward engineering phase. This method may generate several concrete UI (or abstract UIs) in the translation step, as every interaction object can be translated by a series of other objects (figure 2-5).

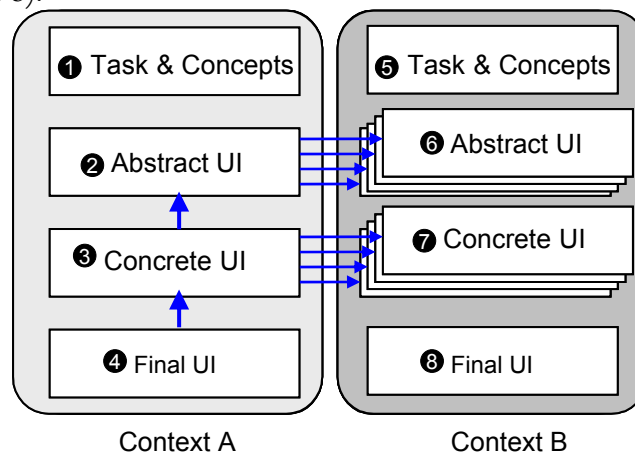


Figure 2-5 Proliferation of UI descriptions

The abstraction process is a relation of the type m-1, while translation is of the type 1-m, thus the combination produces a relation of the type m-m.

Reengineering by retargeting. Its rationale is that, if the target (class of) platform(s) is known before the reverse engineering, the resulting user interface description for the new context of use will be of better quality - in the sense that more accurate reverse engineering rules will be applied, producing models suiting a particular context of use - and that this process will help the designer in the translation step by reducing the number of possible results.

The retargeting process is represented by the dotted line on figure 2-6. As shown, the reverse engineering is directly applied for the target platform. Once an abstract or concrete UI model has been extracted for a particular context use, it can be used in the forward engineering phase to produce a new UI.

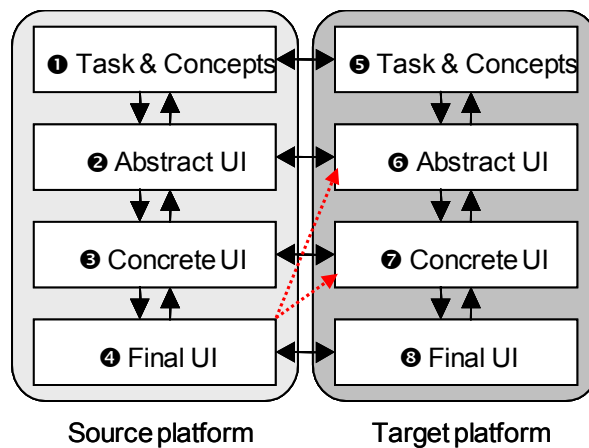


Figure 2-6 Retargeting in the Cameleon framework

Retargeting is more relevant at the CUI level, as the possible transformations are abundant at this level. The concrete interaction objects are platform-independent widgets, but depending on a modality. This level of abstraction gives several possibilities during the translation of the interaction objects, as several corresponding CIO exist for every object belonging to the UI (the CUI level is composed of 34 elements in the version 1.4.6 of UsiXML). Moreover, attributes are also numerous at the CUI level and this fact allows also several possible transformations.

At the abstract level, elements are both amodal and platform-independent. This property reduces dramatically the number of abstract interaction widgets (there are only five different types – called facets - of interaction objects in the last version of UsiXML). However, retargeting can also be applied at the AUI level, but these

transformations are almost constituted of transfer of elements from one container to another. Some other possibilities exist, for example the conversion of one component into an AUI element possessing two facets (for example the transformation of an image-link into a navigation facet with/without an output facet). As possibilities at this level are limited, the retargeting is only applied at the CUI level in this document.

Comparison of both approaches. There are several advantages for retargeting:

1. First of all, this technique *decreases the number of possible results*, and thus the cognitive load for the designer during the *translation* step. Indeed, by removing useless elements for a particular context of use and by transforming some of them in elements suited for a particular class of platforms, the number of transformations and modifications to be achieved afterwards by the designer is reduced. She has also fewer components to take into account during the translation (if some operations are still needed) as the model is contracted during the retargeting process.
2. Secondly, another implication of the previous advantage is that retargeting enhances the *adequacy for a particular context of use*, as it reaches a better level than traditional reverse engineering and translations, thanks to reverse engineering rules tailored for a given context.
3. Thirdly, retargeting *benefits from accessing the FUI code to refine the reverse engineering rules*, as the UsiXML language can not take into account all the information contained in the final UI code (as it is an abstract language). Therefore, by knowing the future context of use, it is possible to take advantage of this information to apply more appropriate reverse engineering rules for particular code statements. A simple example of this is the **address** tag in HTML, which is rendered as a simple label in browsers. This tag is used to describe information about the author, date etc. of the current Web page. By applying “standard” rules, it would be reverse engineered as a **textComponent** but thanks to retargeting rules, it is possible to determine that this component can be discarded if the target context of use is for platforms with limited display capabilities. The same objective could be achieved by using annotations, but in this case, there should be modifications of the current forward engineering tools to be able to process these annotations, or a new tool should be used to complete the translation step to take into account annotations and transform elements accordingly. Thus, by using this

approach, the process remains compatible with the other forward engineering approaches currently developed at UCL.

4. Fourthly, it avoids unnecessary transformations due to the target context of use (possible to *exploit code* to eliminate portions of it known as platform specific). Some code structures are easy ways specific to the final UI code languages, which can also be discarded if we know that the model will be used in another context of use. An example of this is the extensive use of table containing blank images to stabilize layout in HTML 4.0. If we consider a full (or normal) reverse engineering, these elements have to be recovered in the CUI or AUI model, so that the same layout can be regenerated in the case of a looping reengineering to HTML. However, in the case of reengineering to WML or Java, these elements can be removed from the specification during the reverse engineering as they have no utility for these types of UI.
5. Fifthly, retargeting allows to save resources, and thus it provides a gain in *efficiency*, as the models are shortened (so as the processing time), and it should also reduce the time required to achieve the transformation step. This *efficiency* property also encourages potential automated reengineering processes, by producing a model consuming less resources and simpler to process afterwards than one model generated by a “standard” reverse engineering, as retargeting combines two separate processes of traditional approaches.

But these advantages have a price, particularly a loss of generality of the model-based approach and a loss of maintainability. It also implies that the principle of separation of translation-abstraction and the platform-independence principle are not preserved in the retargeting approach.

1. Loss of *generality/reusability*, because contrary to a classical reverse engineering that produces a model which is the source for the generation of many UI, the models produced by retargeting can only be used for a group of platforms targeted by the process. The model does not contain the entirety of information produced by traditional reverse engineering as choices are made during the retargeting process (transformations, deletion, transfer of elements...) and it would be necessary to perform again the reverse engineering if another group of platforms would be targeted. However, the possibility to achieve a reverse engineering without retargeting operations is still possible in our approach, and it is also possible to select retargeting operations suited for a large group of platforms (e.g. monochrome platforms,

Chapter 2 A Conceptual Framework for UI Reverse Engineering

small platforms, sound-disabled platforms etc.), i.e. moderate the retargeting process and thus the loss of generality.

2. There is also a loss of *maintainability*, a loss linked to the loss of generality, as the modifications of an abstract or concrete UI specification has to be reflected on each retargeted model, which is not the case in standard reverse engineering. In this latter case, the general model is modified, and thus maintenance operations are kept during the translation/forward engineering steps.
3. The separation of concerns is intrinsically preserved in traditional reverse engineering, i.e. the *separation of the processes* of abstraction and translation. This principle is lost in a retargeting process, as some parts of the translation are performed concurrently with abstractions. Note that this does not affect the conducting of the process itself, as the retargeting rules are focusing on one sub-problem at a time.
4. Finally, another property of classical reverse engineering is the *platform independence* of the abstracted UI model. This property is lost to some extent (it depends on the retargeting rules that are applied) as the objective of retargeting is to obtain a model tailored for a particular (class of) platform(s). Therefore, this could be interpreted as a loss for the independence principle.

Table 2-1 summarizes and compares properties of both approaches.

Property	Classical reverse engineering and translation	Reverse engineering by retargeting
Possible translations	High	Low
Model Size	High	Medium
Adequacy to context	Medium	High
Model refinement by code	None	Achieved
Efficiency	Medium	High
Generality/reusability	High	Medium
Maintainability	High	Medium
Separation of processes	Intrinsically preserved	Not preserved
Independence	High	Low

Table 2-1 Comparison between reverse engineering and retargeting

An important property of MBA is kept in the retargeting, the reduction of the amount of tools needed to transform a UI from one platform to another. This

reduction is from $(m \times n)$ to $(m + n)$, where m is the number of source language and n is the number of target languages. Without an abstract layer, we would have to write $m \times n$ translators, whereas with the abstract layer, only $m + n$ translators are needed. This remains valid with a UI retargeted thanks to our approach.

Deciding between retargeting and classical reverse engineering depends on the purpose of the UI reengineering: if a UI should be generated for a small group of platforms (e.g. mobile phone and pocket pc), retargeting would be more adequate, but if a UI should be generated for a large group of platforms with various capabilities then classical reverse engineering would be preferred instead.

Retargeting is only applied on HTML UI in this thesis, as one of the major motivations for HTML reengineering (see chapter 1) will be to transform these UI so that they can be displayed on portable devices.

2.3.2 Alternatives during the abstraction process

Retargeting is achieved at the CUI level by translating the interaction objects in the source UI into the chosen (by the user) concrete interaction object. There are four main basic types of retargeting operations:

- *Translate* an interaction object of the source page into an object type specific to the target platform. For example, it would be possible to translate an image-map in a group of text-links or buttons when the target platform has no mouse device.
- *Modify* the value of some attributes of the original interaction objects. For example, for a monochrome platform, every colour attribute in each object has to be modified in a black or white colour.
- *Delete* elements that cannot be displayed on the target computing platform. For example, for a platform without sound capabilities, the background sounds of an HTML page should be discarded.
- *Transfer* (Move) of interaction objects: platforms with smaller display capabilities than the source imply that the layout and page composition have to be modified. Some interaction objects can be moved from one logical window (or abstract container used to structure the UI) to another one, or new logical windows can be created and filled with transferred elements in order to cope with the available screen surface.

The combination of these operations allows the designer to choose between several alternatives during the reverse engineering process. For example, the retargeting of a

Chapter 2 A Conceptual Framework for UI Reverse Engineering

HTML UI to a mobile phone context offers multiple transformations that have to be chosen by the designer. For each target context, a list of possible transformations is available to the designer and the most probable one is already pre-selected. An example for the translation of an image in the context of the retargeting for a mobile phone (or any small platform) is given in table 2-2.

There are six available transformations for images for the retargeting towards mobile phone context. The designer can choose to keep the image - in the .WBMP format - with a direct access or not (in this latter case, the end-user will choose if it is worth losing connection time to see the image). The designer can also put constraints on the transformation: he can choose a threshold of size (i.e. the image would become unclear on a small screen) or to skip small-sized images (usually, these images are only used to have a stable layout in HTML).

Images	Transform into a label composed of the alternative text.
	Transform into a text-link that redirects to the image transferred in another window.
	Convert it to the .WBMP format.
	Suppress all images.
	Suppress images bigger than x pixels.
	Suppress images with a size lower than 50 bytes.

Table 2-2 Alternatives in the transformation of images.

Table 2-3 shows another example of transformation alternatives for text-links.

Text Links	Keep only intra-site links.
	Transform intra-page links into extra-page links.
	Suppress download-links.
	Suppress redundant links.
	Suppress style attributes.
	Suppress colour attributes.

Table 2-3 Different alternatives for text links.

Text-links are divided into three categories: internal or external to the Web site, internal or external to the page and navigation or download links.

The designer can choose to suppress links external to the Web site, in order to keep only the really important links for the navigation of the Web site. She can also choose to transform intra-page links into extra-page links: intra-page links are generally used to jump to precise parts of long Web pages that are not suitable for

small-display platforms (huge scrolling manipulations). This problem can be reduced by breaking the page into smaller ones, and this subdivision does not disturb the user's perception of the site structure. When this option is chosen, the rest of the page is automatically transformed into a set of smaller pages (see figure 2-7).

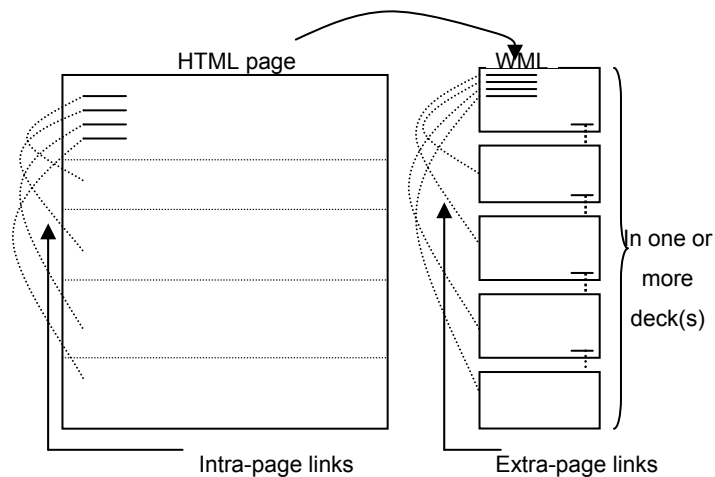


Figure 2-7 The transformation of intra-page links into extra-page links

Since mobile phones do not have a secondary memory, download links are useless on this platform and can be removed. Sometimes, we can find the same target for different links on the same page. These redundant links can be automatically removed to save screen space. And finally, the designer can choose to suppress the colour and the style (bold, italic ...) attributes for each link. These two last options are available for each textual object (links, labels, marquees ...), but can also be selected at the model level in order to avoid repetitive manipulations. Other examples of retargeting rules can be found in appendix G.

2.4 Conclusion

This chapter described a conceptual framework for UI reengineering and some definitions particularized to the UI domain. This conceptual framework and its associated definitions will be used in the next chapter to locate and characterize the approaches of the state of the art in reverse engineering so as to position the approach presented in this thesis.

A new approach in UI reverse engineering was also illustrated in this chapter, the retargeting of UI, allowing extracting a UI model suited for particular context of use. Retargeting will be applied in the analysis of HTML (chapter 5 and 6), permitting us

Chapter 2 A Conceptual Framework for UI Reverse Engineering

to achieve a flexible reverse engineering of UI. A case study using the retargeting concept is given in the section 9.2.3, and performance and correction analysis of this process –compared to traditional reverse engineering- can be found in section 9.1. This new approach can be characterized by a trade-off between a loss of generality and maintainability, compared to traditional reverse engineering in model-based approach, and a gain in the precision and adequacy of the produced model so as resources needed to perform the translation step. It is true on the one hand that retargeting is no longer compliant with the independence between abstraction and translation. On the other hand, it is likely that the designer will devote more efforts on fine tuning the final results than for the abstraction followed by the translation. Indeed, results of abstraction are transient and are not reused for another purpose in this thesis.

Chapter 3 State of the art in reverse engineering

After defining in the previous chapter some reverse engineering terms that will be used afterwards, some related work of this field of research is now presented. This chapter contains several approaches dedicated to the reverse engineering and reengineering of user interfaces. Each time a related work is presented, we will only focus on those aspects which are directly relevant to this thesis. There are two main sections in this chapter, the first is about legacy systems user interfaces reverse engineering and the second section contains approaches for Web sites re-(verse) engineering (which only uses static analysis techniques). An Internet version of this state of the art can also be found at <http://www.isys.ucl.ac.be/bchi/research/soare.htm>. Its last update was 14th July 2002. This chapter presents the related work we are aware of at the date of this manuscript.

Several techniques exist for the reverse engineering of applications, and most of the approaches presented in this chapter use one or a combination of them:

Static analysis of Web pages examines the code of a Web page without interpreting or executing it in order to understand aspects of the Web site [Boui02a]. Since static analysis has been successfully used in software testing and compiler optimization, it

has been extensively used for analyzing the HTML code of Web pages. However, this technique leaves all non-HTML parts of the Web page untreated. Therefore other techniques need to be investigated such as the following methods.

Slicing techniques [Gall91] examine selected, related portions (slices) of code to analyze behavior and to recover design decisions. The code statements in a program slice are not necessarily contiguous; this facilitates design recovery even if the code is poorly structured. This seems promising for SVG code.

Pattern matching [Merl95b] parses the code of a UI to build a manipulable representation of it. Then slicing techniques are used to extract interface fragments from this representation, and a pattern matcher identifies syntactic patterns in the fragments. Using the code fragments as a basis, details about modes of interaction and conditions of activation are identified with control flow analysis.

Syntactic Analysis and Grouping [Vans93b] relies on a recognition algorithm that identifies input/output statements and attempts to incorporate them into groups. The grouping information is then used to define screens from the original user interface. This is particularly appropriate for scripting languages.

Cliché and Plan recognition [Will90] automatically identify occurrences of clichés, stereotyped code fragments for algorithms and data structures. The cliché recognition system translates the original code into a plan calculus, which is then encoded into a flow graph, producing a language-independent representation of the interpretation's flow that neutralizes syntactic variations in the code.

Abstract interpretation [Lech96] parses the entire code to build an abstract representation of it that can be virtually executed based on first-order predicated expressing conditions before and after each instruction. An abstract interpretation is an approximation of the semantics of computer programs, based on monotonic functions over ordered sets. This technique seems particularly appropriate for examining Java and JavaScript.

Trace analysis [Stro99] analyses user inputs and screen composition to infer an abstract representation of the system. This analysis is followed by a clustering of the results to merge similar UI screens and actions and avoid “doubles”. The code is not necessarily needed for this technique.

Transcoding [Brit01] transforms the UI code for a particular platform into another form or into another language without constructing a model of the UI. The particularity is that there is no abstraction process in this technique and it is usually achieved on-the-fly with rules peculiar to a couple of platforms (see section 1.4). This technique is not a reverse engineering approach but is often applied to transform a UI for another context of use, especially for Web sites.

Annotation techniques [Lope01] (consisting in adding information in the original code before the application of the transcoding rules) are also used in transcoding approaches, allowing to enhance the quality of this dynamic process.

3.1 Reverse engineering of legacy systems

This section will report on a state of the art of selected work related to reverse engineering of legacy systems, which will be useful for this research. Each work will be systematically analyzed according to the following structure: a general description, the software architecture of the system if it is relevant, the underlying principles and the type of algorithm/rules used in the system. The level of detail of this analysis will depend on the interest of this work for our analysis and the existing literature. A common diagrammatic representation is given for each approach in the general description section. The figure represents the location of the approach in the conceptual reference framework presented in chapter 2. The name of the source language is displayed at the final UI level on the source-context side, and the output language (if any) on the target-context side. Some additional information is given such as the reverse engineering technique and the abstract language used in MBAs (at the level of abstraction reached by the approach).

3.1.1 Morph

General description

Information systems are critical to the operations of most businesses, and many of these systems have been maintained over an extended period of time, sometimes twenty years or more. Nowadays, many organizations are choosing to reengineer their critical applications to better fit their needs and to take advantage of the new technologies. MORPH [Moor93, 94, 96, 97] is a process for reengineering the user interfaces of text-based legacy systems to graphical user interfaces (GUI), and a toolset to support the system. MORPH identifies basic user interaction tasks and associated attributes in legacy code. The resulting model is then used to transform the detected abstractions in the model to a specific graphical widget toolkit.

Morph allows reengineering a UI by abstracting the legacy code to the abstract UI level in the reference framework (see figure 3-1), as the categories of elements recovered are very generic. This AUI model is then translated to a CUI model for another context of use, by using a human input to control this process. Finally, a UI is generated in Motif, Java or Windows resource files based on this CUI model.

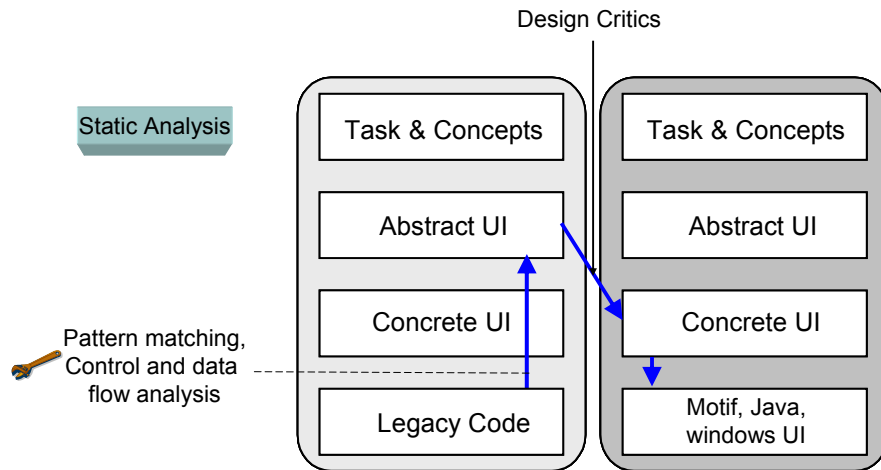


Figure 3-1 The reengineering process with Morph

Principles

The reverse engineering process is composed of three steps (see figure 3-2).

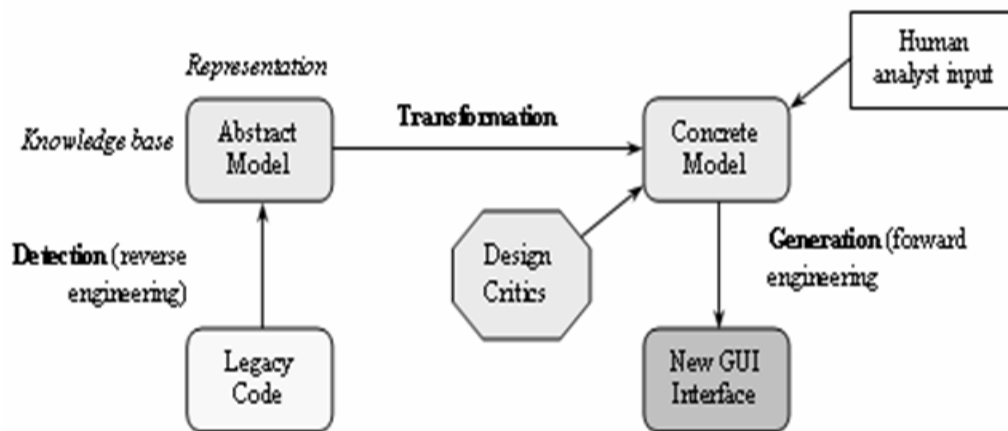


Figure 3-2 Morph's reengineering process

1. The *detection* which is also called program understanding. It analyzes source code to identify UI components in the legacy system, thanks to a detection engine.
2. The *representation* step in order to build a model of the existing user interface as derived from the detection step. This model is stored in the knowledge base.
3. The *transformation*: thanks to the transformation engine, it is possible to manipulate, augment and restructure the resulting model to a graphical environment. The human analyst can refine the model in the transformation stage. The transformation stage suggests specific graphical implementations and integrates them for user interface abstractions into the legacy code.

The detection step identifies interaction tasks (entries of a unit of information by the user) from code that enables the construction of an abstract model of the UI. Four basic interaction tasks can be recognized:

1. Selection: the user makes a choice from a set or list of choices.
2. Quantification: the user enters numeric data.
3. Position: the user indicates a screen or world coordinate.
4. Text Entry: the user enters text from an input device.

The technique for determining UI component in the code consists of three steps:

- First of all, it is necessary to isolate the User Interface Subsets (UIS), which include all routines and data structures that are affected by user I/O.
- The next step consists in identifying data structures that can be read or displayed from the UI. These data structures can give clues about user interface objects in the program, which will define the application interface model for the system. These data objects will be used to communicate between the user interface and the computational code of the application.
- The final step consists in finding the dialog control model (action, states and transitions). A set of coding patterns is used to detect dialog elements.

Representation is done thanks to patterns. Coding patterns that describe the implementation of each of the basic interaction tasks are used to detect these tasks in text-based user interfaces. In a GUI, however, there are many different possibilities for the implementation of a particular interaction task. Therefore, once the interaction tasks are identified, attributes are collected to pinpoint the appropriate abstract interaction objects. Morph maintains a hierarchy of concepts (see Figure 3-3), composed of the set of Abstract Interaction Objects (AIOs) [Vand93] categorized under the basic interaction tasks, allowing to select the appropriate objects for each interaction tasks. These AIOs present indeed some abstraction with respect to the physical widgets since they are oriented towards the input/output of information in a generic way. However, these AIOs are more defined opportunistically and do not involve any task or user aspect. The same observation holds for the context of use.

Transformation thanks to the knowledge base (built in CLASSIC) strengthens this approach. This knowledge base can be used to aid in the detection of user interface components from legacy code and then used to identify the appropriate replacement interaction objects within a specific toolkit such as MS-Windows.

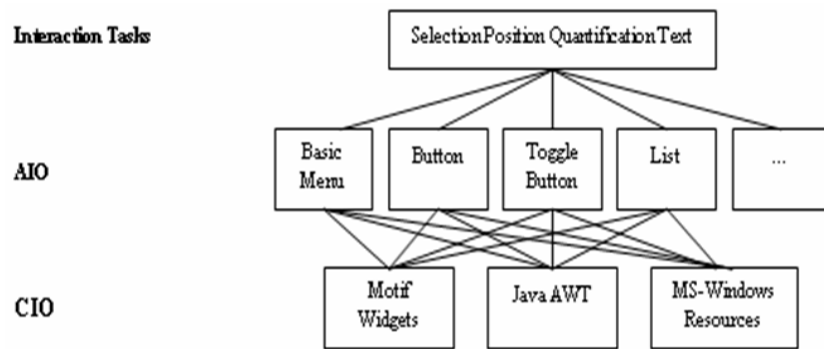


Figure 3-3 Morph's categorization of concepts

Type of algorithm and rules

The reverse engineering is achieved by applying static program analysis techniques, including control flow analysis, data flow analysis, and pattern matching. Pattern matching is the technique mostly used in MORPH, as it is needed to detect UI and dialog components, technique that is supported by a knowledge base.

3.1.2 The AUIDL Environment

General description

In this approach [Merl93, 95], the UI is represented in an object oriented manner while the interface behavior is described using Milner's process algebra. The original UI is first translated in AUIDL (Abstract UI Description Language) which is able to represent UI objects in terms of both structure and behavior.

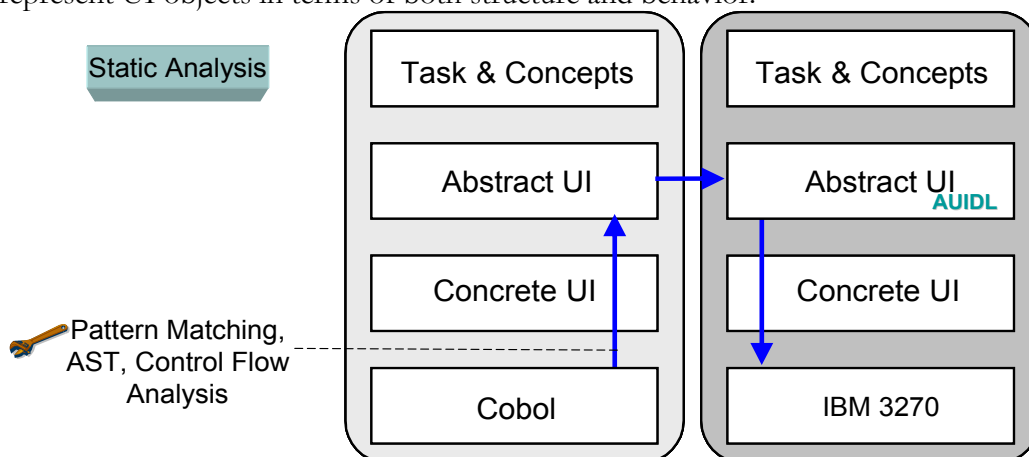


Figure 3-4 The reengineering process with AUIDL

Chapter 3 State of the art in reverse engineering

The entire reengineering process is represented on figure 3-4. The source FUI is the Cobol language, which is first abstracted at the AUI level. Based on an equivalent AUI for another context of use, the forward engineering allows producing screens for the IBM 3270 environment.

Principles

The approach starts with the generation of an AST by parsing the source code. A module extracts AST UI fragments (I/O system calls or user interaction). Components are identified and used to construct the abstract specification of the UI. This step is semi-automated, a part of the code is parsed and automatically abstracted, and the remaining of the sliced code is left to the programmer. For the layout, the spatial organization of display objects is explicitly described with two mechanisms: containment and importation (allows an object to import an attribute from another object: for example import the x-coordinate of an object to have the same alignment for these objects).

The Milner's process algebra (MPA) is used to map out the behavior of the system. In MPA, the behavior of a system is defined as either its entire communication capabilities, or by what is observable in such a system. During the forward engineering, when the new UI is integrated in the information system, constraints on data and control flow graphs have to be respected at the cut points in the code to obtain an equivalent interface. In the last step, the AUIDL specifications are translated in the EASEL language. The EASEL language allows the automated generation of screens for the IBM 3270 environment. Finally, the generated code is linked with the functional core.

Type of algorithm and rules

UI composition is obtained from an abstract syntax tree (AST) and from the related syntactic information of a particular implementation. Using Refine/Cobol and the BMS parser, an AST is obtained from the source code of the system. To translate the components and their behavior in AUIDL, 'user actions' and 'system responses' have to be identified using pattern matching techniques and control flow analysis completed by a human intervention.

3.1.3 Cellest

General description

Cellest [Stro00a, Kong99, Stro02] is the acronym for CEL Legacy Enhancement Software Technologies. The Cellest purpose is to migrate and optimize uses of a

legacy system on a new platform. Cellest adopts a transversal approach in order to extract the necessary information to complete the possible tasks on the old system. Transversal means that the task is the subject of interest and information related to tasks are gathered even if different programs are used to achieve a particular task.

The reengineering process is depicted on figure 3-5. The reverse engineering recovers both tasks and abstract UI models from screens coming from various legacy systems. Then, thanks to these models, an abstract UI for another context of use is translated/reified. The forward engineering phase allows to generated XHTML and WML UI based on this new AUI model.

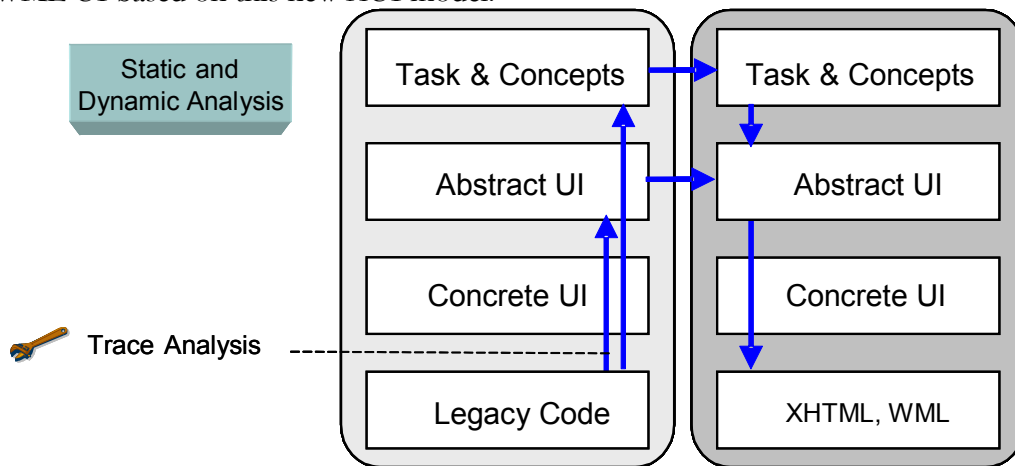


Figure 3-5 The reengineering process with Cellest

Architecture

Cellest is composed of two middle-ware tools: the *recorder* and the *pilot* allowing the capture user interaction during the use of the legacy system. Therefore, a *recorder* localizes aspects of the legacy system relevant to a specific task, instead of having to deal with the whole body of the system code.

In this approach, an interface is viewed as a collection of uniquely identifiable screens(traces), each of which allows a set of possible actions to transition from this screen to other screens. The screens are captured by the recorder that records each session (visited screens, actions in order to achieve specific tasks).

The *pilot* translates these actions in functionally equivalent actions in a new GUI. Its role is the same as an application model [Gurm89]: to bind the new UI with the old legacy system.

The complete architecture of Cellest is shown on figure 3-6. Traces captured by the recorder are used as input of Lendi to generate interface graph and also used by

Mathaino to construct the tasks, domain and abstract UI model and generate a UI by using these models and interface graph. QandA allows editing the produced models while Babel is used to bind the different legacy systems involved in the reengineering.

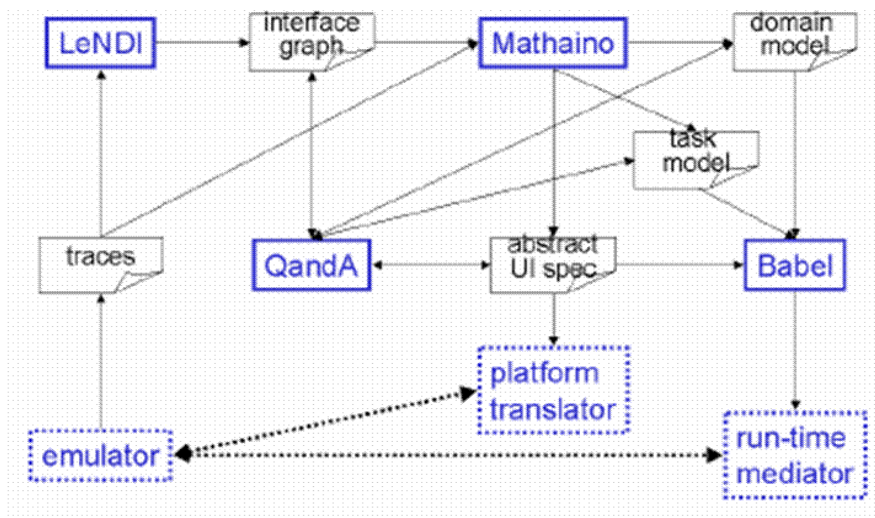


Figure 3-6 The architecture of Cellest

Principles

A more accurate definition of each component of the Cellest toolset is given below.

A. Lendi (LEgacy Navigation Domain Identifier)

The interface mapping task is accomplished by the Lendi system. In the first step of this process, the *recorder* collects the snapshots of the system screens and all the user-system interactions. Then, the interface mapping process proceeds to recognize identifying features in the screen snapshots of the trace, in order to cluster several screen snapshots together as instances of a single unique system screen. The user configures the clustering process by deciding which features to use and their relative weights in the similarity detection function between snapshots. Finally, the user actions that enable the transition (and their preconditions) from one screen to another have to be identified and specified. The output of this process is the interface graph whose nodes correspond to the individual screens of the system and its edges correspond to the user action sequences enabling transitions from one screen to another.

B. Mathaino

Mathaino is a system for the migration of legacy systems to multiple platforms. The objective of Mathaino is to reverse engineer the information-exchange plan between

the legacy system and the user and to specify an optimized abstract UI to accomplish this task. Traces stored by the recorder are the inputs of Mathaino. The process starts with the output field recognition. The designer has to associate each output with a name. Then, a navigation plan is generated with variables that must appear on these screens. The last step consists in the association of fields with domain objects to generate a domain model. An abstract UI can finally be created for each task, either automatically or manually. The tool can then produce a new UI based on these abstract models thanks to two translators: a XHTML and a WML translator.

C. QandA (Questions AND Answers)

The QandA system guides a user to inspect and review the process and its products: a user can inspect the clustering of the traces in order to discover within a cluster some traces that may lead to a reclustering. The designer can also review the task analysis in order to revise Lendi's decisions regarding variable scope and to provide meaningful names for the identified variables. The generated GUI can also be revised by QandA, by requesting from the user only the necessary inputs and by driving the underlying interaction with the legacy system through the pilot middleware.

D. Babel

Babel's aim is to bind several legacy applications together. Babel is a query planning and application monitor tool that integrates divergent processing models. Babel provides an environment for specifying related applications in terms of the functionalities they deliver and the data these functionalities expect as input and produce as output. These functionalities are integrated following 'business rules' specifications. Based on these specifications, Babel produces a run-time mediator that monitors the behavior of the underlying applications, evaluates the defined rules on the global state of the integrated system, and generates triggers for new functionalities (new tasks) to be accomplished according to these rules.

Type of algorithm and rules

The reverse engineering in Cellest is based on trace analysis to recover abstract models. A human analyst is needed to define the clustering, and to determine variables and domain elements required on those screens so as the transition between them. Then the task and navigation plan can be generated automatically, and based on the information defined previously, an abstract UI is produced.

3.1.4 Cobol UI reengineering for Windows

General description

In this approach [Csab97], the original program is run and remote controlled through a communication module by the new application. The new application is recompiled for Windows with the I/O (input/output) calls replaced. The source program was written in COBOL and was running in a mainframe environment. It had its own data management system built in and uses the ADIS (ACCEPT/DISPLAY System) run-time module of Micro Focus COBOL.

This process is located at the FUI level (figure 3-7). The legacy system is wrapped, and the old UI components are replaced by a Windows UI, which communicates with the functional core of the legacy system.

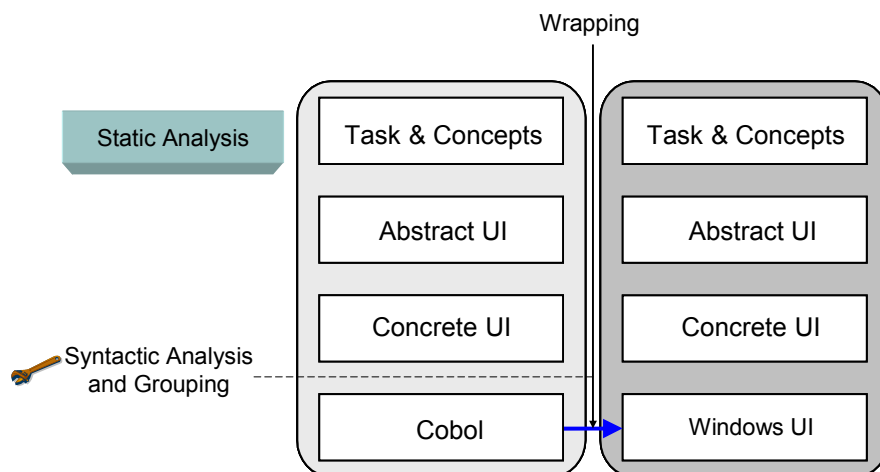


Figure 3-7 The reengineering with Csaba's approach

Principles

The problem with old COBOL program reengineering is that it is poorly structured ('spaghetti code') and has to be converted into an event oriented code. This task is very hard, especially for huge monolithic mass of code which was developed gradually over many years. The solution adopted in this approach is the 'remote controlling'. First the COBOL program had to be modified in two ways: replacing the I/O calls by functions communicating with a shared memory area, and making it loadable by Windows. The replacement functions copy the information into a buffer and another program - the VUP (for Virtual User Program: program that manages the communication to the shared memory from the GUI point of view) - reads this information out of the buffer and use it to build its own dialog. When a Windows-GUI is used, the VUP places the new data into the memory and pushes the virtual

key by putting the appropriate keycode into the shared return area. The buffer is used in order to avoid a continuous refresh of the GUI because the DOS program redrew the screen for each keystroke. Finally, the presentation of the GUI is done manually.

Type of algorithm and rules

As this approach is an experience report, no specific rules were designed to achieve this reengineering.

3.1.5 PUC

General description

This approach [Nich02] introduces an intermediary graphical or speech interface on several platforms (such as a Pocket PC or a PDA) for complex appliances (VCR, camcorders, thermostats...).

The reengineering process is shown on figure 3-8. The abstract UI of the complex appliance is abstracted and this model is used to generate a final UI for a Pocket PC or a PDA.

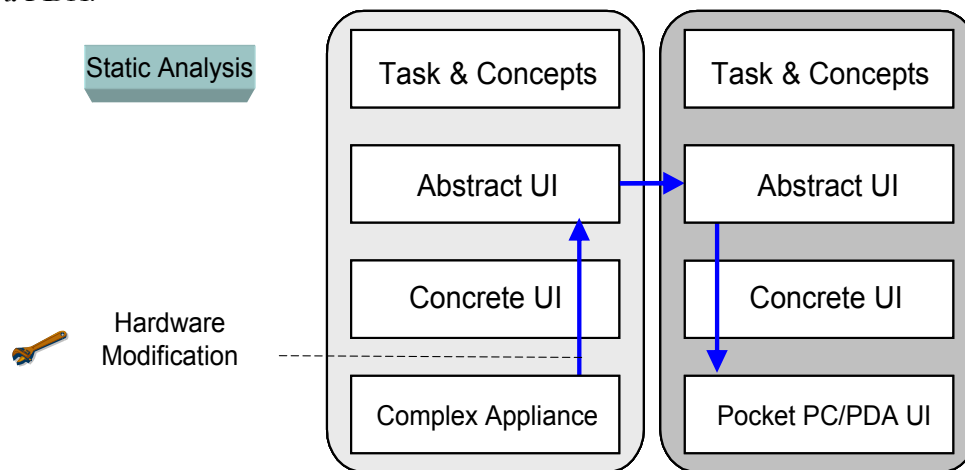


Figure 3-8 Reengineering process with PUC

Principles

The PUC (personal universal controller) can communicate both-ways with an appliance thanks to modification of the hardware of this appliance. The specifications of the appliance's functionalities are captured in a model, which is used by the mobile platform to construct the UI for controlling that appliance (a graphical UI, speech or both). The specification of the functions is grouped hierarchically with their state-dependence information in a specific XML language. The automatic

generation of UIs comes with a “Smart Template” mechanism [Nich04]. A Smart Template is a kind of domain specific design pattern, which defines the set of state variables (e.g. “time”, “volume”) and functions (e.g. “play”, “record”...) typical for a given kind of appliance. Multiple combinations of states and commands are allowed in the template definition, which allows a single template to be applied across slightly different appliances. At the rendering stage, the elements specified in the Smart Template are then rendered using platform-specific controls and intelligent layout and resizing mechanisms.

Type of algorithm and rules

The reverse engineering step of this approach is based on observation and hardware modifications of the devices, and therefore no reverse engineering rules were developed in PUC.

3.1.6 Visual TAP

General description

To overcome the problems and limitations of traditional screen readers, the Archimedes project (<http://archimedes.hawaii.edu/>) developed VisualTAP [Scot01], a software that recovers GUI screen information directly from the video signals and a GUI-accessor that processes this information to separate textual and graphical components and presents them to the user through haptic display and speech synthesis. Figure 3-9 represents the reengineering process. No abstraction process is done in this approach, as the video signal is directly used to generate a new UI.

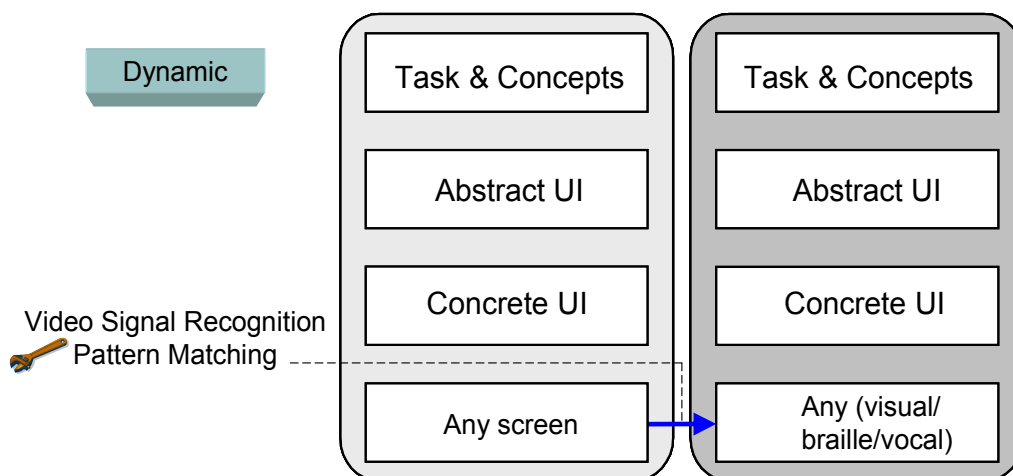


Figure 3-9 The reengineering with TAP

Architecture

The architecture of Visual TAP is illustrated on figure 3-10.

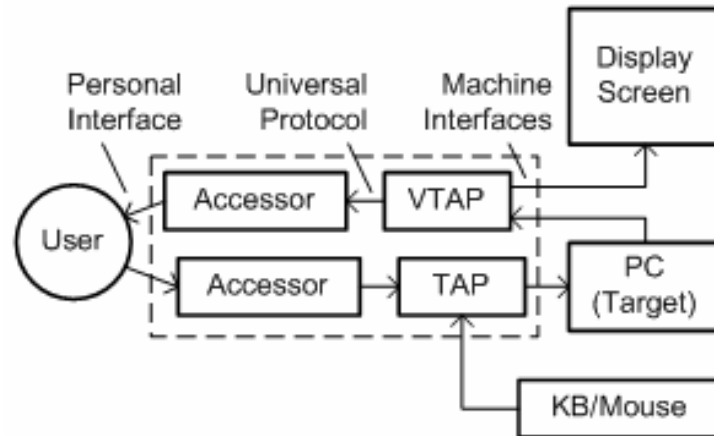


Figure 3-10 Visual TAP

We can distinguish three components [Scot01]:

- Accessor, a personalized device that provides its user with his or her preferred interaction modalities.
- Total Access Port (TAP), a small keyboard and mouse emulator that provides direct access to any user accessible functions on the target computer.
- Universal Protocol, a communication protocol that allows any accessor to connect to any TAP. The connection can be wired or wireless (Infrared or Radio).

Principles

The different steps are the following:

- Analog RGB signals for the GUI computer screen are captured and stored as digitized data in the computer memory via the PCI bus so as to obtain a bitmap image of the screen.
- The bitmap image is turned into a grayscale image which is passed through an edge detector using a modified Sobel operator.
- The image is turned into a black and white image and fed into two different processes: recovering navigational objects such as window outlines, buttons and scrolls bars (corners and images are identified by pattern matching) and dynamic cluster detection (to identify icons, text, and pictures).
- The GUI components are then rendered through a force feedback (haptic) display or as three-dimensional sound and text is produced as synthesized speech or Braille.

Chapter 3 State of the art in reverse engineering

Type of algorithm and rules

The reverse engineering is achieved by processing video signals, by applying functions on images representing the UI.

Another similar approach is currently developed by M. Felsberg (<http://www.isy.liu.se/~mfe/>) allowing retrieving components on screenshots thanks to a camera. Rather than processing a video signal, the system captures screens and analyzes the produced images.

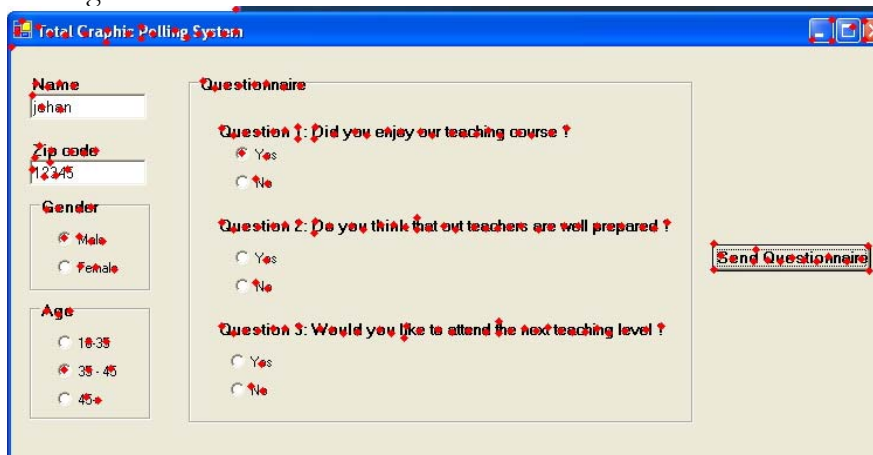


Figure 3-11 Widgets recognition in Felsberg's approach

This approach goes one step further than the visual tap approach as the analyzed screen could even not exist anymore (e.g. by analyzing images in documentation of a legacy system). However, this technique has been developed with another objective, border recognition in images, but could also be used as a new UI reverse engineering technique. An example is given on fig. 3-11 illustrating the analysis of a screenshot of a window by the system. Dots represent the borders of the components of the UI.

3.2 Reverse engineering and reengineering of the Web

3.2.1 Digestor

General description

The main purpose of this approach [Bick97] is to address the problem of displaying desktop computer screens on small-sized screens of portable devices. Web sites could be accessible to any mobile device by re-authoring existing HTML pages.

The figure 3-12 represents the transcoding approach achieved by Digestor, between two FUI expressed in HTML.

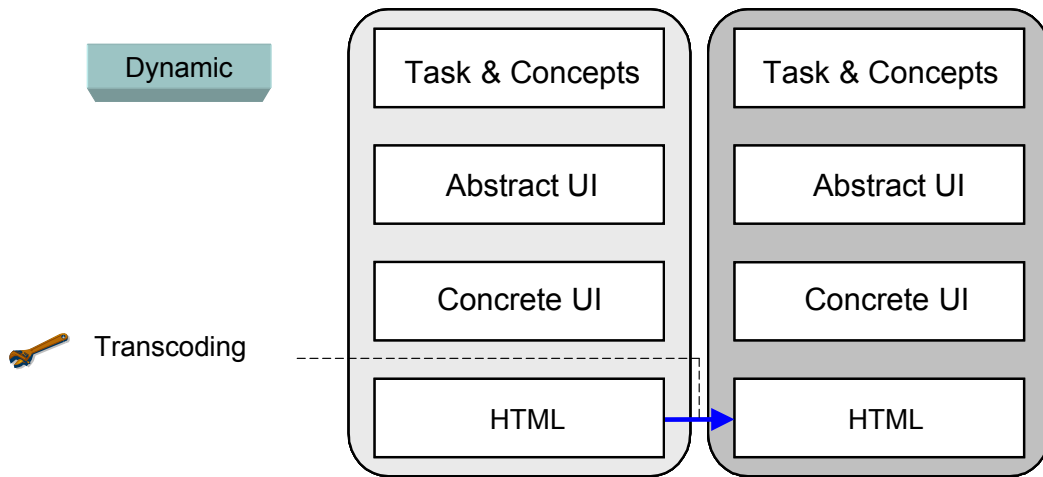


Figure 3-12 The transcoding approach with Digestor

Principles

There are two main categories of techniques for the re-authoring of Web pages: it can be semantic or syntactic techniques and the re-authoring can be realized by using transformation or elision techniques. Semantic techniques rely on some understanding of the content of the Web page while the syntactic techniques operate on the structure of the page. Elision is the operation of removing content and transformation involves the modification of some aspect of the page.

The techniques belonging to both categories are ranked by priority and can be used conditionally. The techniques are applied following a strategy optimized for a given document/display-size pair. The strategy is realized according to a re-authoring method which searches a document transformation space in a best-first manner. Each state in this space represents a version of the document. At every step in the search process the most promising state (that with the smallest display area requirements) is selected and a transformation is applied to it. Once a sufficient screen space has been saved, the document is returned to the client for rendering.

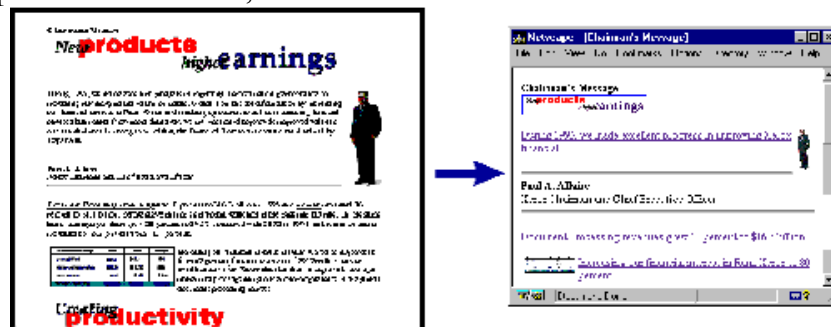


Figure 3-13 Digested Web Page

An example of a page transcoded by Digester is shown on figure 3-13. The paragraph elision technique is illustrated on this figure.

Digester is implemented as an HTTP proxy server. The system is implemented in Java, and the HTTP proxy server software was based on the MBServlet system. This system can also respond to requests for certain URLs with documents generated by the proxy itself, which is used to provide the user with form-based control over the proxy and the re-authoring process.

The approach has been enhanced during years, and is now named “M-links”. More information about M-links can be found in [Hilb03].

Type of algorithm and rules

Digester uses a set of re-authoring techniques to save screen space: syntactic elision techniques (section outlining, first sentence elision, image elision ...) and syntactic transformation techniques (image size reduction, font size reduction ...) that are applied by pattern matching.

3.2.2 Mobile Transparent access

General description

According to the authors, there are two main approaches to bring the internet on mobile phones. Web content can be adapted for different clients by specialized proxy servers that perform different tasks of conversion, caching and content adaptation (mobile-transparent approach). The second type, the mobile-aware approach, is to design and implement totally new services that are specially designed for mobile users.

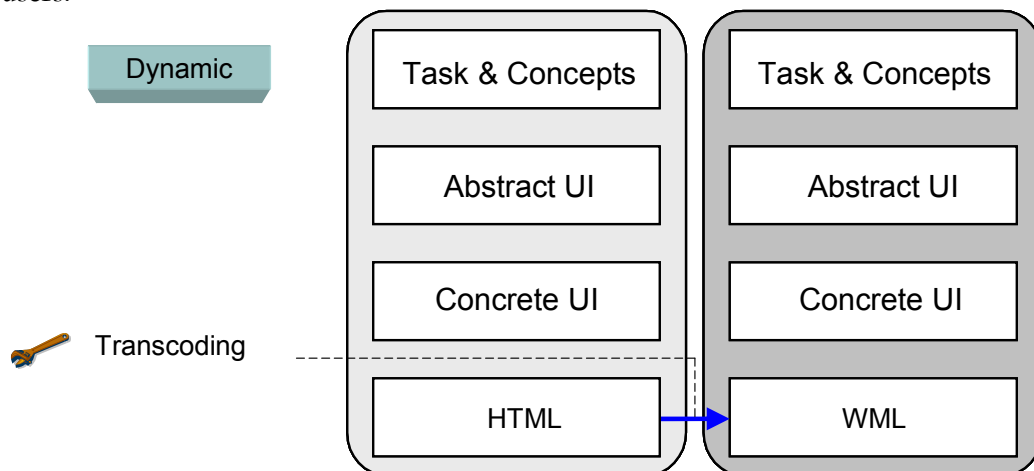


Figure 3-14 The mobile transcoding approach

Chapter 3 State of the art in reverse engineering

This study [Kaas00, 01] proposes a HTML/WML conversion proxy server through which the users of different WAP phone can access the same Web services as they use on their PC. This approach is located at the FUI level on the reference framework, as the transcoding occurs at the code level between a UI specified in HTML and a UI expressed in WML.

Architecture

The proxy server aims to format the service according to the capabilities and preferences of the mobile client, as defined by the User Agent Profile (UAProf) of the device. The architecture of the tool is depicted on the top part figure 3-14. Requests coming from WML-enabled phones are forwarded from the WML proxy server to the HTML server. Then the HTML code is transcoded on the HTML/WML proxy. Finally, the code is sent back to the WML-enabled phone. The ad-hoc approach is shown on the bottom part of figure 3-15, where no transformations are applied as the WAP server gives access to Web content designed in WML.

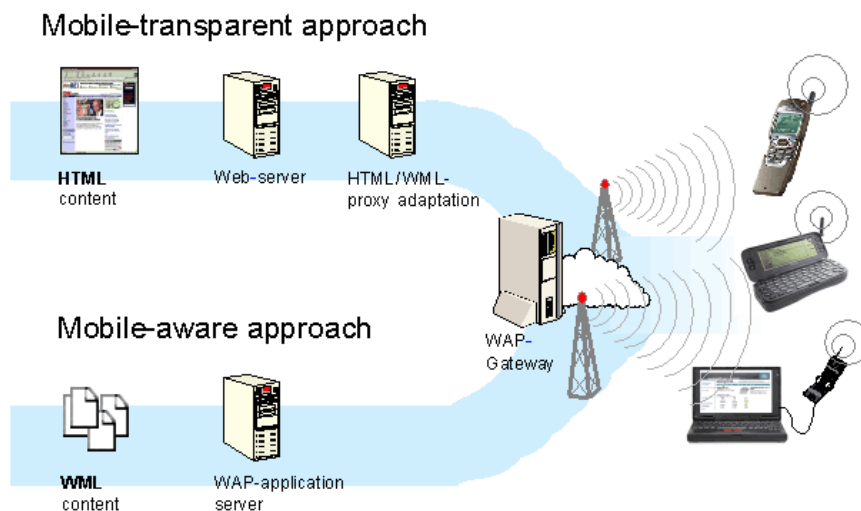


Figure 3-15 The mobile-transparent approach

Principles

The original HTML pages are converted dynamically in several decks (depending on the memory limits), and their content is adapted to WML. For example, images are presented as short text entries using either the included meta-data or the image source file name.

This type of transcoding had globally good results – principally by preserving the familiarity of the service – but brought some usability problems: users often felt lost if

Chapter 3 State of the art in reverse engineering

they proceeded too far from the top of the page. There was also a lack of navigation aids to ease browsing. Sometimes, the proxy failed in the transcoding because of a bad structure or syntax errors of the original code, and the page was unavailable. This study indicates that if HTML-based Web services follow certain guidelines, they can be converted automatically to WML and adapted to the client device. In principle these guidelines already exist as W3C Web Content Accessibility Guidelines and W3C Note for HTML 4.0 Guidelines for Mobile Access.

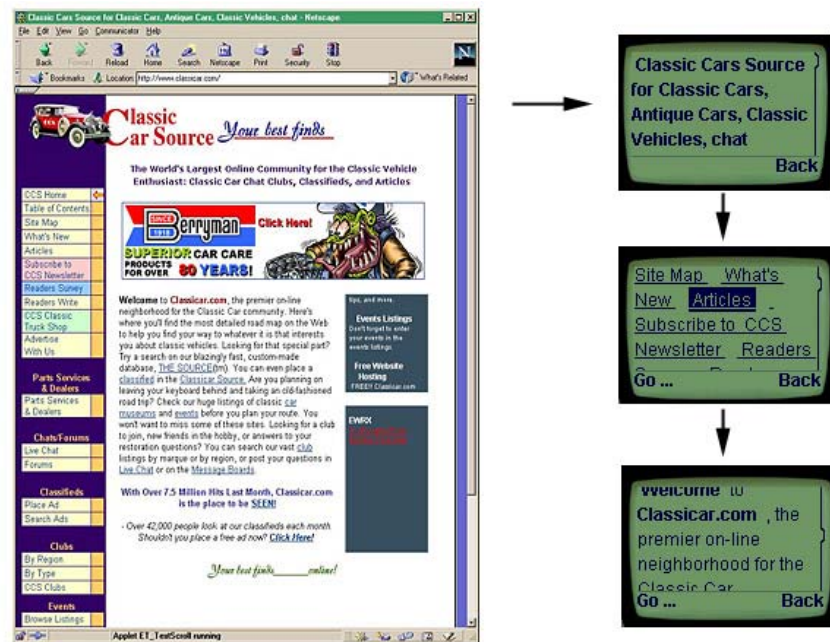


Figure 3-16 Transcoded Web page

An example of an HTML page transcoded into WML thanks to this approach is shown on figure 3-16.

Type of algorithm and rules

This approach uses a set of 1 to 1 transcoding rules applied by pattern matching in a static analysis. A selection of some of these rules is achieved thanks to the UA profile, allowing adapting the transcoding to some properties of the target platform.

3.2.3 PIMA

General description

The PIMA [Bana00, Susm01, Gaer03, Berg04] project aims at producing applications that are device independent. Functionalities and their corresponding UIs should adapt to the device and the physical environment, or in other words, devices should be seen as portals and the application should adapt to the current available resources. In this context, the authors introduce the concept of generalization. Generalization is the process of capturing the semantic information inherent in a UI and represented in a platform independent model (PIM), by inferring such information from a UI designed for a particular device. This information can then be used to create UI for other devices. The complete process envisioned by the PIMA project is positioned in the reference framework on figure 3-17. The reverse engineering is applied up to in platform independent representation at the CUI level, then the model is translated for another context of use and reified to a final UI. The final UI here is represented by “any” language, as the approach aims to cover a maximum of languages and the scope is not well defined for the moment.

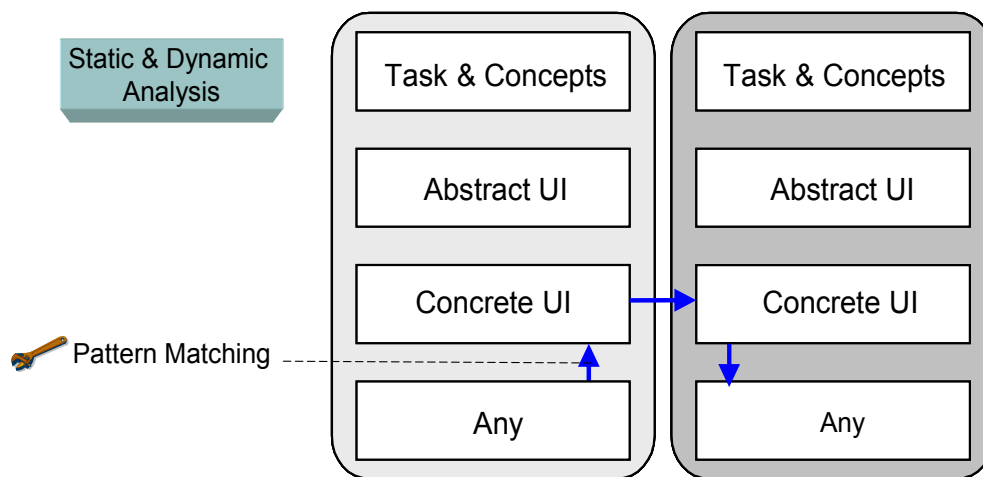


Figure 3-17 The reengineering process in the PIMA project

Architecture

The architecture of the PIMA project is shown on figure 3-18. A Platform Independent Application can be created either by a design tool or by abstracting a concrete UI thanks to the generalization process. A specialized engine with a device profile then creates another application specialized for a particular device. This UI

can be modified by hand by a dedicated tool. Finally, the specialized application can be translated into a language compatible with the target device.

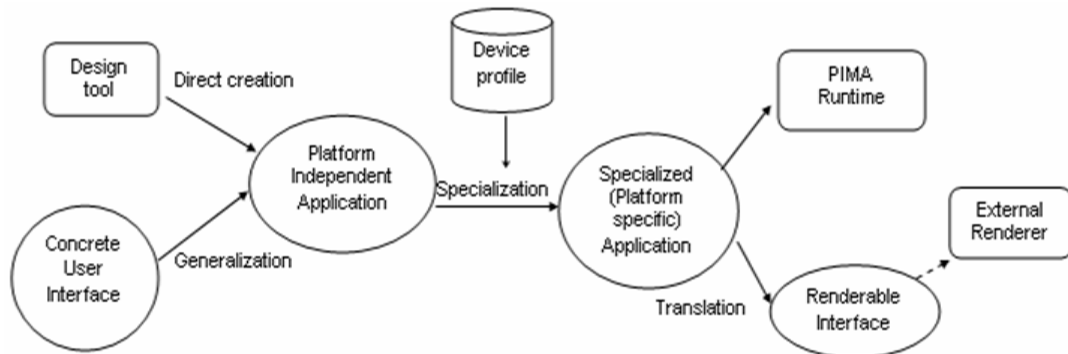


Figure 3-18 The complete Pima process

Principles

The PIM is composed of four main concepts: a set of task and the navigation among them, a set of interaction elements, a set of event handlers (responses to user actions) and a set of directives that are used by the specialization engine to specialize the application properly (for example: specifications of strongly linked elements).

Generalization is done by reverse engineering the code of the UI. This process starts with the detection of interaction elements. Secondly, the properties and semantic information of these elements can be inferred. But the inference engine cannot resolve every case of this part of the reverse engineering. There must be a feedback mechanism that allows the designer to help the system to resolve ambiguity about the information that is inferred from the concrete design. An example of these ambiguities is the inference of the descriptive label for a textbox surrounded by three different labels. An automatic deduction system of this kind of relations has been implemented [Gaer03]. In this approach, all the possible semantic links between labels and their surrounding elements are evaluated thanks to a list of heuristics. For example, the descriptive label of a radio button is often displayed on its left-hand side or its right-hand side. This fact gives a higher probability for the labels positioned in this manner to be linked semantically with the radio button. Another example is the caption of images, which is most of time displayed on the bottom of an image. The complete page is thus first analyzed and points are given to each possible pair of elements. Then, the highest evaluation is selected to establish relations in the model.

Type of algorithm and rules

The generalization process is not applied directly to the UI (such as HTML) code, but to a more orderly internal representation to standardize the input of the process. Two kinds of rules are applied during this reverse engineering process. First, the elements are detected by pattern matching, and secondly an evaluation thanks to heuristics allows defining the semantic properties of elements.

3.2.4 WebRevenge

General description

In this approach [Paga02], a Web site is reverse engineered in order to reconstruct the corresponding task model. The task model uses the ConcurTaskTrees representation [Pate00]. The approach is represented in the reference framework on figure 3-19, on which an HTML page is abstracted up to the task & concepts level.

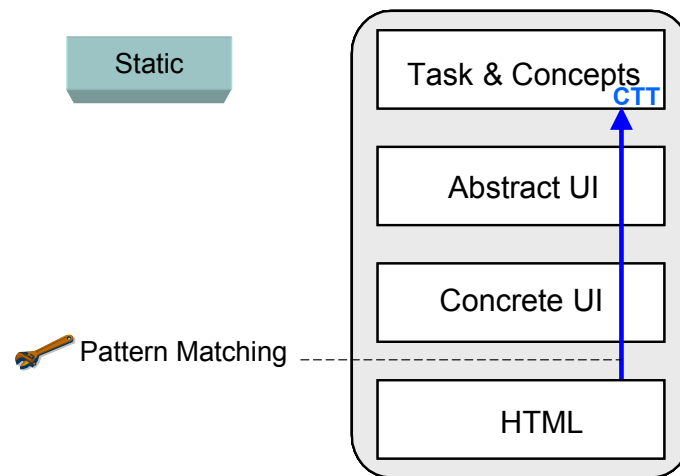


Figure 3-19 The reverse engineering process of Webrevenge

Principles

The HTML page is first syntactically validated by Tidy - a tool allowing correcting HTML files to obtain a valid XML syntax - and the corresponding DOM structure of the page is created. DOM stands for Document Object Model and is the representation of an HTML document in an object oriented fashion. Then, it applies several rules to create the task model. The output can be stored in XML or in a format that can still be subject to modifications by using CITE [Pate00].

The construction of the global task model for a Website is illustrated on figure 3-20: the site is reverse engineered page by page and a task tree is built based on these subtrees. Additionally, it is possible to use the output of WebRevenge as input for

WebRemUSINE [Lece98]. WebRemUSINE allows recording the interactions with a Web site, which can then be compared to its corresponding task model to detect usability problems.

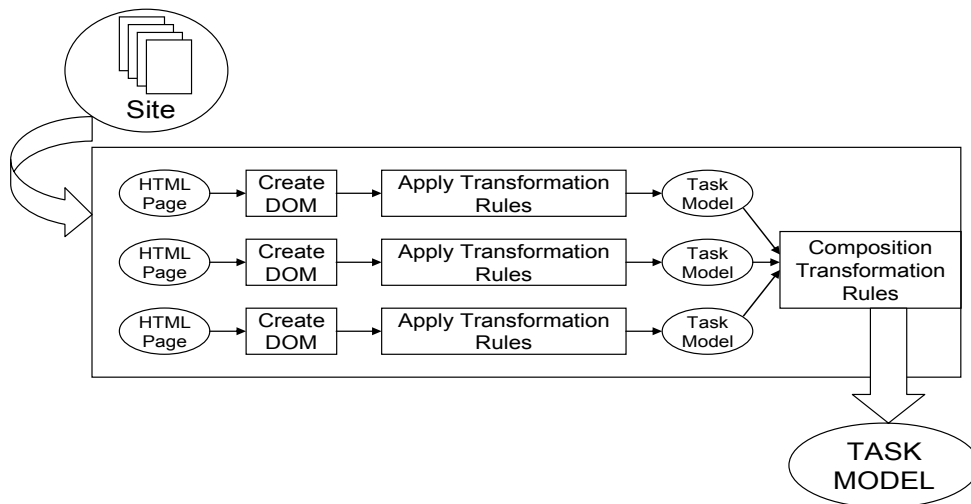


Figure 3-20 Reverse engineering of the task model

Type of algorithm and rules

Rules to recover the task model are applied thanks to pattern matching. Example of rules for creating tasks are : the creation of a root task for the page, the creation of task structure associated with a link, the creation of a new task level associated with **fieldsets** or the creation of task structures associated with **textareas**. Once the process of creating the task model associated with each page is completed, the overall navigation is reverse engineered in order to bind the different task models.

3.2.5 ReWeb

General description

ReWeb [Ricc01, Ricc00] is aimed at avoiding the inevitable degradation of Web sites by restructuring and rewriting the HTML code.

The process is represented on figure 3-21. The HTML code is reverse engineered, and maintenance operations are applied at the concrete level. This model is then used to generate an updated HTML UI.

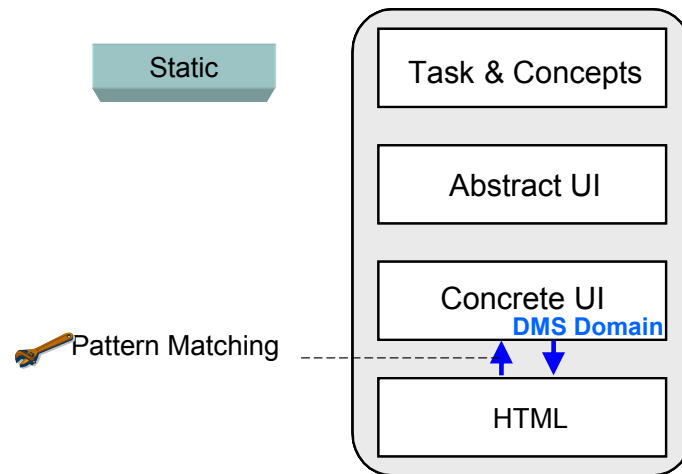


Figure 3-21 The Reweb reengineering process

Architecture

The architecture of Reweb is decomposed into three major components (figure 3-22). The Web Spider downloads all pages of a target Web site starting from a given URL. The Analyzer uses different versions of the downloaded Web site to calculate the difference between two successive versions of the site. The Viewer provides a graphical user interface to display the output of history analysis.

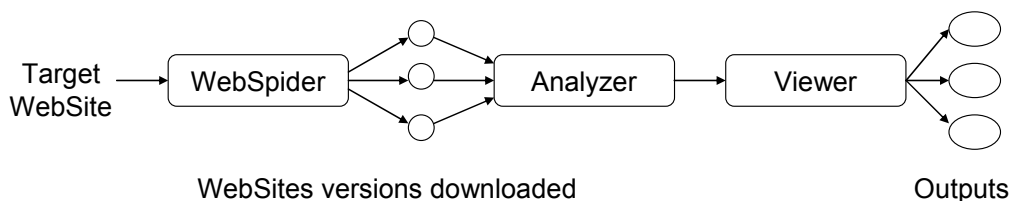


Figure 3-22 The Reweb's Approach

Principles

Reweb first reconstructs an abstract representation of the structure of the Web site in a directed graph. From this graph, information for the maintenance can be extracted such as the shortest path or the 'strongly connected components'. Reweb also allows monitoring the evolution of the site by periodically downloading the entire set of pages, and by comparing file names and HTML code.

Reweb applies several transformations to the pages to achieve the maintenance operations. There are two kinds of transformations: inter and intra pages transformations. The first type transforms several pages and updates the abstract representation of the structure. The latter type only modifies one page at a time.

Reweb aims at improving maintainability (M), usability (U) and portability (P). This is done thanks to 6 different types of operations:

1. *Syntactic clean up* (M): like tidy [Tidy03], corrects syntax errors in the HTML code.
2. *Design restructuring* (M): by automatic reorganization into frames, page extraction/insertion.
3. *Page restructuring* (M): this type of operation includes adding description for non-loaded objects, modifies absolute URLs into relative ones and adds the part "noframes" to framed pages...
4. *Style renovation and grouping* (U): transforms HTML pages with presentation markups into pages with style sheets.
5. *Improving accessibility* (U): provides keyboard shortcuts for important links, provides text links for image-maps, etc...
6. *Update to new standards* (P): depreciated tags are eliminated and replaced.

The maintenance applied in Reweb is a semi-automatic process, as intra pages transformations can be achieved automatically while inter pages transformations are done manually.

Type of algorithm and rules

Reweb use the DMS Reengineering Toolkit (Design Maintenance System) to rewrite HTML pages. DMS enables the transformation of arbitrary languages ('domains') by accepting a domain definition for the language consisting of: domain grammar, pretty printer rules, transformation rules (intra-domain rewrites), refinement rules (inter-domain rewrites), procedural analyzers. There is thus a need to write a language grammar, pretty printer and transformation rules as the two other components are optional. The transformation rules can be conditional in DMS, and can also be grouped in rules sets. More sophisticated control strategies can also be implemented procedurally.

3.2.6 WARE

General description

The main purpose of Ware [Dilu01, Dilu02] (Web Application Reverse Engineering) is to provide a support to the recovery from existing Web Applications (WA) of UML diagrams dealing not only with static content, but also with the dynamic content. This approach is a support tool for maintenance and site evolution. The aim here is to produce documentation (redocumentation process).

The reverse engineering process is depicted on figure 3-23, where the final UI specified in HTML is abstracted up to the concrete UI level.

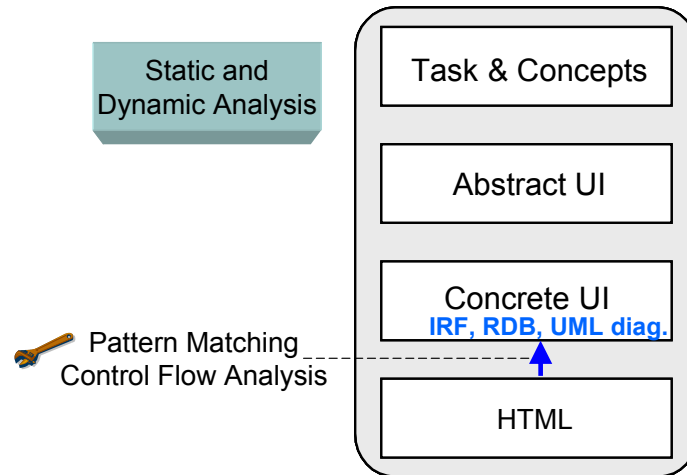


Figure 3-23 The reverse engineering process with WARE

Architecture

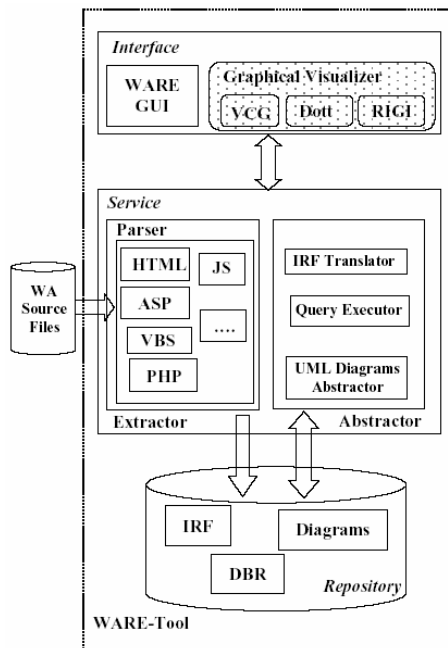


Figure 3-24 – The Ware architecture

The tool is composed of three components (see figure 3-24) :

1-Interface layer: GUI or Textual UI to visualize the results

2-Service layer: The *extractor* analyzes and classifies the HTML objects and scripts and stocks the results in IRF files (Intermediate Representation Form which is an internal markup language). The *abstractor* is composed of a translator that translates IRF files in a relational database (RDB), a Query Executor that allows predefined search in the RDB and produces information based on queries, a UML Diagram Abstractor that produces class diagrams and provides necessary data for the construction of sequence and use case diagrams (with the help of user-customizable queries).

3-Repository: in MS-Access. Data are stored under the three formats of the service layer (IRF, DBR, Diagrams). The taxonomy is represented by class diagrams.

Principles

There are four main steps in the process:

1. A static analysis and class diagram recovery
2. Identification of notable sub-graphs (i.e. sets of classes) in the class diagram, where each sub-graph (set) is responsible for a WA functionality
3. Use cases recovery by associating each set of classes to a single use case.
4. Sequence diagrams recovery, for obtaining several scenarios of using the WA, by analyzing the dynamic interactions among the WA components.

A conceptual representation of WA components has been developed in WARE to realize the static analysis, thanks to the definition of a classification for Web pages and their components.

For example, links (static or dynamic) are mapped as relations. They have identified three types of relations: relations related to the submit button (between a form and a server); redirection relations (between a script and a page or between a link and built page); Inclusion relations (between a client page and a client module). Interactions between components are triggered by events deriving either from the code control flow, or from user actions.

The reverse engineering process is automated up to the dynamic analysis (third of the four step). At this point, the user has to create a use case scenario for each class graph (found in the static analysis). The dynamic aspect and parameters involved are found back by a RDB search (query). The sequence diagrams are constructed by tracing the events in the objects of the WA.

Type of algorithm and rules

The reverse engineering - aiming at the generation of documentation - is achieved by using pattern matching techniques to recover static aspects, and control flow analysis combined with a human intervention to recover the dynamic aspects.

3.2.7 TAMEX

General description

The approach [Stro00b, Situ00] followed by Tamex is based on the concept of task-specific mediation: information sources within an application domain are encapsulated within wrapper agents interacting with an intelligent intermediary agent, the mediator. Mediation is the integration of diverse and heterogeneous data by abstracting away the representations differences among them, and by integrating their individual views of the application domain into a common model.

Chapter 3 State of the art in reverse engineering

The reengineering process of Tamex is illustrated on figure 3-25. The reengineering allows recovering the task and domain models (at the task and concepts level) from one or several HTML pages, to translate to another context of use and to generate a new user UI specified in HTML.

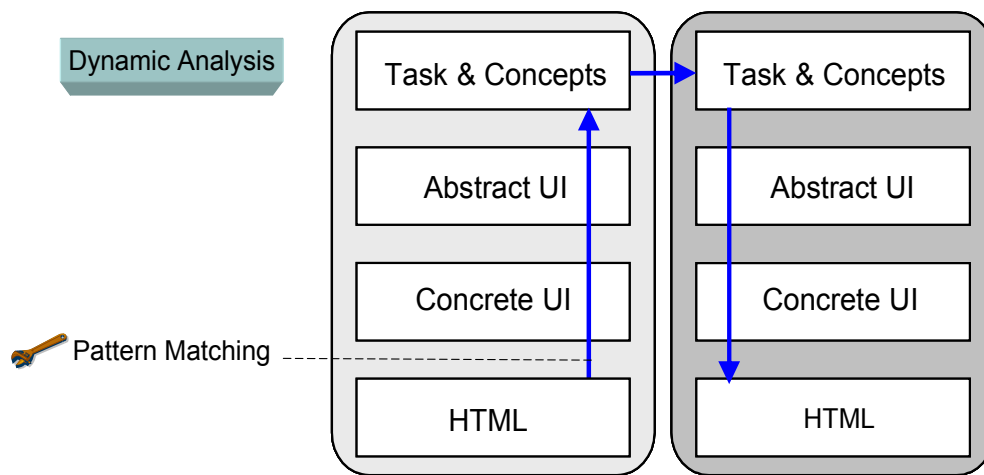


Figure 3-25 The reengineering process with Tamex

Architecture

The architecture (figure 3-26) is composed of three layers (UI left, mediator middle, wrappers right).

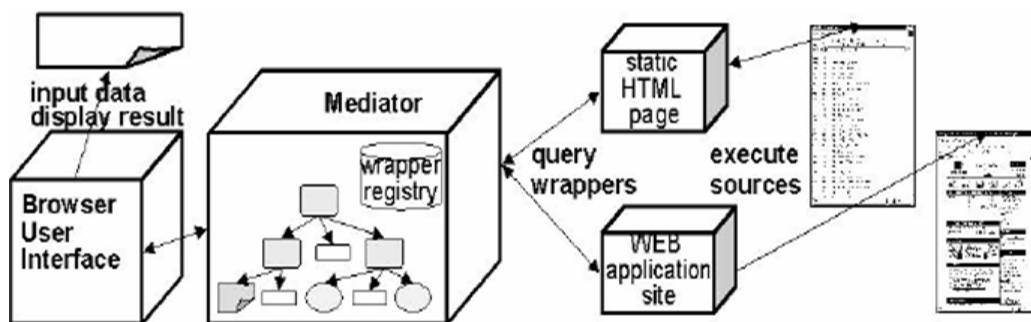


Figure 3-26 The Tamex mediation process

Tamex consists of three major components:

- Wrapper agents: they drive and extract information from a set of corresponding Web Applications (within a domain).

Chapter 3 State of the art in reverse engineering

- A mediator agent: whose task structure drives the interaction of the aggregate application with the users and controls the flow of requests and information to and from the wrappers.
- A UI representing the task and domain models generated thanks to the two previous components.

Principles

Tamex's approach is based on two models, the *domain* and *task* models. The *domain* model consists in a hierarchy of entities. It contains three types of information: names and attributes of the entities, their relationships and the invariants of these entities. The *task* structure models the agent's behavior. There are three types of tasks for the mediator:

- *User interaction tasks*: represents the mediator's interaction with the user (for example: ask the user the destination of a flight).
- *Information-collection tasks*: the mediator identifies the application that supplies the necessary information and sends a request to the corresponding wrapper (for example: search in an HTML page all the flights with a specified destination).
- *Internal tasks*: internal function to process information.

The high level tasks are implemented by a menu-driven interface and are activated and decomposed by the users. The low level tasks correspond to the mediator's interaction with either the user or the wrappers to invoke the underlying information sources. The task model represents the control of information exchange and the interaction between users and applications.

The mediator provides a UI generated by applying XSLT to the domain model and the task model. The mediator contains a JAVA servlet which connects the browser with the wrappers and processes the collected information. There is a wrapper for each HTML source and they map the requests in the appropriate protocol.

The wrapper construction is done in two phases, firstly the demonstration phase during which traces of the interaction between the user's browser and the resource servers are recorded by proxy servers. Secondly, the learning phase during which traces are compared against equivalent XML example request (composed according to the domain model) to learn the application request protocol.

Type of algorithm and rules

The information extraction -to recover the domain model- is done with an XPath-based algorithm that generates extraction rules from HTML. These extraction rules to retrieve information on Web pages are generated by following a hierarchical

approach based on the tree structure of HTML documents (by pattern matching). Same concepts are often in the same sequence of HTML tags. The learner observes the first XPath to the contents in the HTML page and reproduces it (for example: each cell `td [i*5+3]` in a table contains the price of the flight).

3.2.8 ADAPT

General description

The goal of this approach [Lope01, 04] is to adapt Web pages so that they can be displayed on any device, especially portable devices. The authors have identified three sources of variation in the access of Web sites: the variety of information (page content), the variety of devices (different capabilities) and the variety of users (skills, roles...). There are two main steps in the process of adaptation:

1. *Original Web Page Classification*: Classifies the different elements of the original page, divides the page into sections, classifies nodes, etc. Classification may be done manually or by an automatic classifier
2. *Adapted Web Page Generation*: Generates the Web pages adapted to the device and user preferences, using the classification and the device and user description. The adapted pages are generated by a set of transformations.

The transcoding process is depicted on figure 3-27, between two FUI expressed in the HTML language.

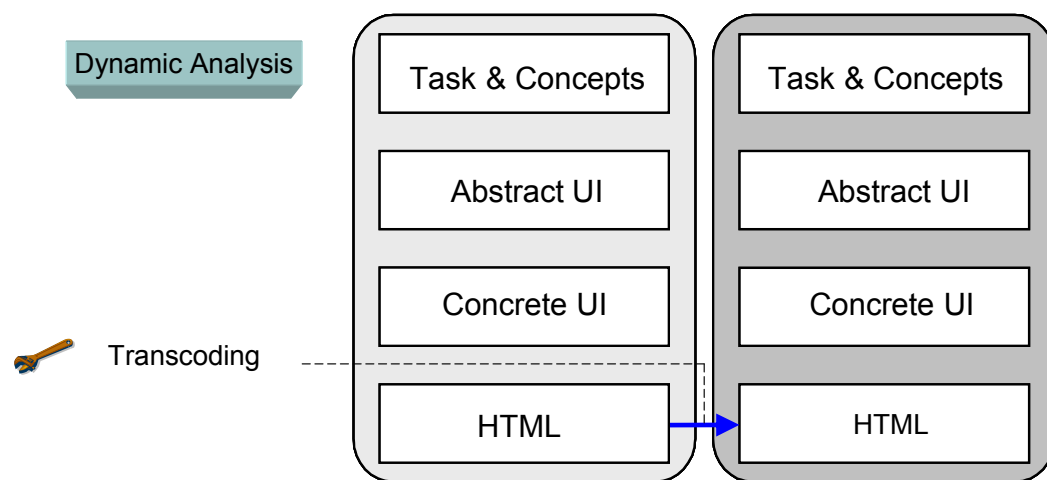


Figure 3-27 The Lopez's adaptation-transcoding approach

Architecture

The architecture of the tool (figure 3-28) is based on a proxy server – similarly to the WML transcoding approach in section 3.2.2 - that forwards the requests to Web

servers and on which the adaptation of the Web page is achieved. The modified page is then sent back to the user.

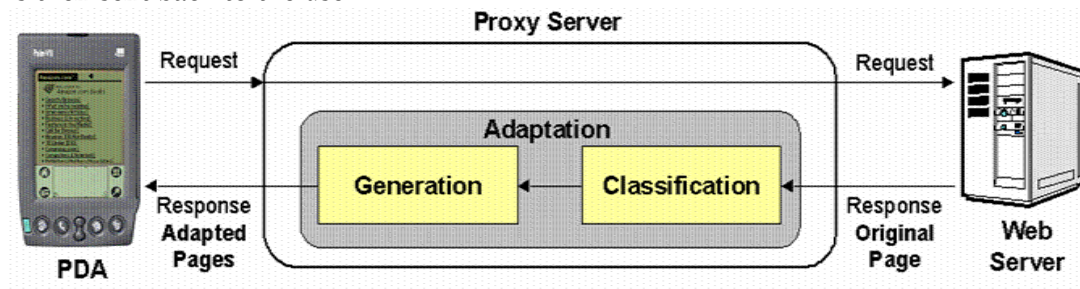


Figure 3-28 Adaptation of Web pages

Principles

The approach is composed of two steps:

The classification. Several types of relations can be used to segment the document, such as the spatial, structural, semantic or typographical relations. Two operations are made during the classification: the conversion of HTML into XHTML and the parsing and classification of the document to generate a tree representation of the different sections of the page.

The generation. The page is generated according to the device and the user preferences. Following these criteria, a navigation strategy (tree, summary, links ... navigation) is generated, in order to optimize how the information is transmitted and displayed to the device-user. According to this strategy, one or several pages are generated with their navigation elements thanks to a set of transformations.

Classification and generation are independent and cleanly separated processes. The system can use different classifiers or generators depending on the needs.

Type of algorithm and rules

To create the new page, the system generates a set of XSLT transformations applied to the original document (by pattern matching). These transformations include:

- HTML code adaptation to meet the device constraints.
- Layout adaptation to fit the device screen. Each section can be displayed with different levels of details.
- Images and Multimedia adaptation: graphics are scaled, transformed in monochrome images or replaced by their alternative text.
- Navigation mechanisms: code for navigation is added to each adapted page, or index pages with links are generated.

3.2.9 Transcoder HTML to VoiceXML

General description

This approach [Pere03, Shao03] is based on a taxonomy of tasks (based on information structure) to transform Web pages in usable voice UI. This is done by aiming at the development of voice navigation of Web spaces not as a replacement for visual Web browsing, but to support limited directed information seeking tasks. This approach follows two goals: to provide voice access to the existing Web pages and to present Web information in an effective and useable voice UI. The transcoding process is represented in the reference framework on figure 3-29, starting from a HTML UI and translating it to a VoiceXML UI at the FUI level.

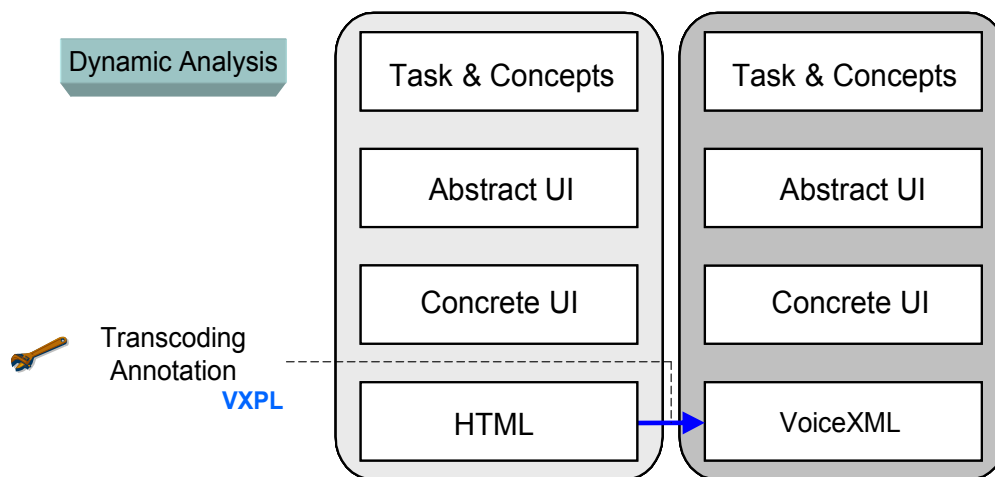


Figure 3-29 The voiceXML transcoding approach

Architecture

The system is based on the IBM Websphere [Brit01] software (see figure 3-30).

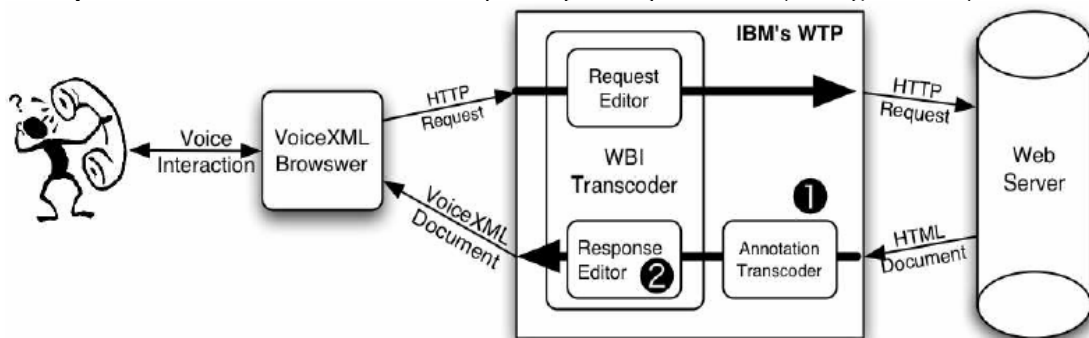


Figure 3-30 The VoiceXML Transcoder Architecture

Chapter 3 State of the art in reverse engineering

First, a request is sent for an HTML file. The Web page is then downloaded on the proxy server on which Websphere runs. This file is annotated with the VXPL language and serves as input in the transcoding process to produce a VoiceXML file. This last file is then sent back to the user.

Principles

As the voice interface has a serial and temporal nature, the UI should provide users (and user level tasks:) an understanding of its structure (*situate*), how to navigate that structure (*navigate*), how to filter information (*query*) and how to focus in on detailed information (*details on demand*) to mimic the principles of visual design of Schneiderman (overview, zoom & filter, details on demand). They have outlined four information structures and linked them with these user level tasks.

1. *Top-level visual regions*: headers, titles, ... are mapped as top-items of menus in the vocal UI.
2. *Menu/list structures*: Web pages present lists of information in a (pseudo-) hierarchical structure which is mapped as a hierarchical menu structure with top-level visual regions (*Navigate, details on demand, situate*)
3. *Text areas*: Large parts of texts are spoken thanks to a text-to-speech synthesizer. The UI provide the user with a speed of speech control. They also provide quick query for links in text, a summarization command (*details on demand*) and data extraction such as phone numbers, email address, ... (*details on demand and situate*).
4. *Related/Structured information*: set of information fields that are repeated for different instances of the information (database listings). On the voice UI, each record can be read in sequential order. *Navigation* commands allow the user to move quickly in the list and he can also request specific items (*query, details-on-demand*) or the list can be represented as a menu.

Another novelty of this approach is the introduction of the promotion concept which permits to put lower-level items that should appear as menu items at a higher level (as the hierarchy of HTML tags does not always reflect the importance of the subject).

Type of algorithm and rules

This approach uses an annotation language, VXPL, which has been developed to support the transcoding of Web pages. This language allows adding information about the structure of a page and how it should be translated in term of user's tasks. Then the transformation is applied following a pattern matching technique.

3.2.10 Reverse engineering end-user Web applications

General description

The approach [Bhar05] mainly consists of a toolset able to generate abstract representations from end-user developed Web applications implemented with Click [Rode05]. The goal is to provide documentation to expert developers who would enhance these existing Web applications.

This redocumentation process is located at two different levels of the reference framework (figure 3-31). This process produces a documented specification of the HTML UI at the concrete level and at the task and concepts levels.

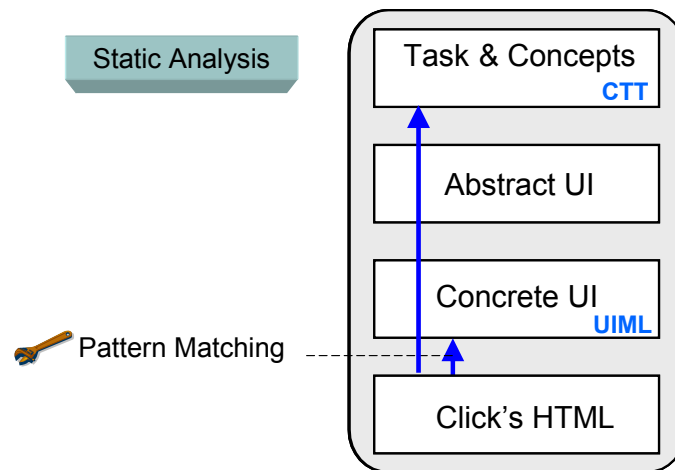


Figure 3-31 Click's end-application reverse engineering approach

Principles

There are four outputs for this reverse engineering process: a sitemap, a task model, a UIML representation and text documentation. The sitemap consists in a directed graph representing navigation and information flows. Documentation is produced to be used with the sitemap: for each page (node in the sitemap), some information is given such as the type (form, contains databases or not, static information page etc.) and purpose of the page. User's inputs that are used throughout the site are traced and represented in the documentation. When a page uses a database, its schema is also shown in the documentation. The task models are expressed in CTT and are very similar to the task models produced by WebRevenge (see section 3.2.4), but here some dynamic aspects of the Website are also recovered. The UIML diagrams are recovered by reverse engineering the Click internal language (enhanced HTML). Parts of the click's code that cannot be represented in UIML are also recovered by

making a link in the UIML specification to an external Java module (e.g. databases links and searches).

Type of algorithm and rules

This reverse engineering is achieved by applying pattern matching techniques. As several dynamic behaviors, e.g. database requests, are specified in the Click's internal language (and thus not on the server side or in a script language), some of them can be recovered in the UIML specification by pattern matching.

3.2.11 Revangie

General description

Revangie [Drah05] applies a reverse engineering process of dynamic form-based Websites by simulating actions and inputs. The tool (available at <http://www.revangie.formcharts.org>) can be used with two objectives: the creation of a model of the application for the purpose of product benchmarking or re-engineering and the creation of a model of its users for the testing application (e.g. load testing). The approach of Revangie is depicted on figure 3-32, by applying an abstraction from final UI specified in HTML up to the concrete UI level.

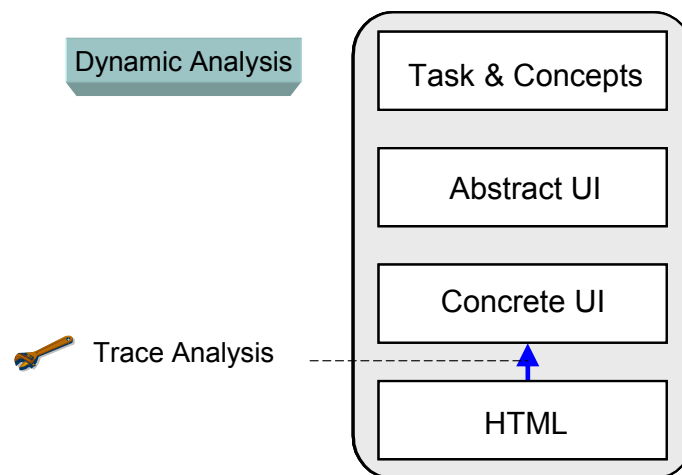


Figure 3-32 Revangie reverse engineering process

Principles

The tool can be used in various modes (i.e. manually, automatically and in a guided mode). When used automatically, Web forms are filled with random input values and actions (such as following link, pushing a button, etc.) are activated randomly until a fixed threshold of actions is reached. In this way, the maximum of pages generated

by the dynamic Website are analyzed. The tool can also be used manually, by recording manual user inputs and actions (realistic session data). A third mode, the guided mode, is also available: the tool only asks user inputs when the automatic mode is insufficient (particular input fields, classification with ambiguities ...). When the traces have been collected, they can be classified and clustered in order to generate the output model. Classification is achieved by using several similarity measures. This is done by calculating a metric evaluating a real number called distance (for example same titles for two pages would produce a distance of 0 for this parameter). Statistics are also used to refine the classification: the dependence of one parameter on another (e.g. title dependency on target) can be tested and from this information refine the equivalence relations. The user can choose the granularity of the produced model by modifying the number of clusters.

Type of algorithm and rules

The reverse engineering is achieved by classifying traces of the analyzed system. Screen classification is done thanks to different parameters: textual similarity, pages generated by the same action, targets identity (same set of signature of the server actions targetted by a screen), identical titles or patterns (text and code), or similarity (according to some textual or structural distance metric). Thanks to these parameters, it is possible to detect instances of a same Web page in the collected traces.

3.2.12 AWT2XIML

General description

The tool AWT2XIML and the corresponding approach developed in [Canf04, Canf05] allows the rendering of Java based UIs on personal wireless devices.

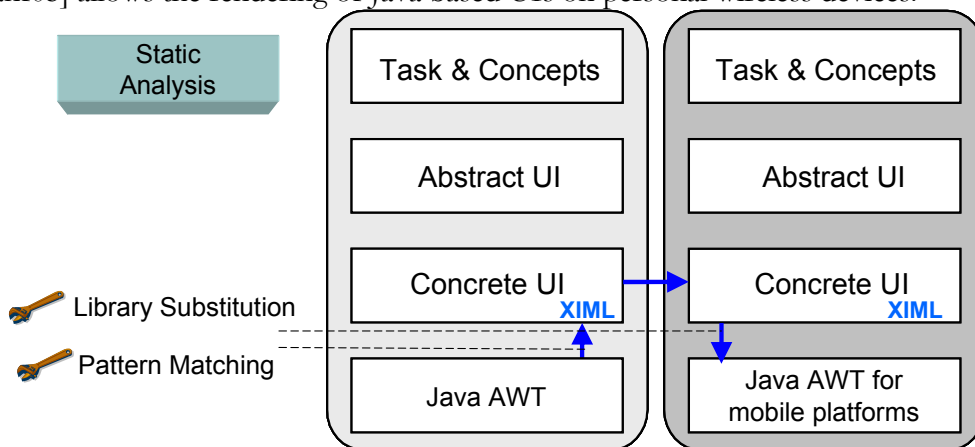


Figure 3-33 AWT2XIML reverse engineering process

Chapter 3 State of the art in reverse engineering

This is done thanks to a semi-automatic reengineering process that transforms the original Java UI in a Java AWT UI reformatted for a small screen. The process is represented in the reference framework on figure 3-33. The reverse engineering is applied on Java AWT final UI to obtain a description at the CUI level. After some transformations, this model is then used in the forward engineering phase to produce Java AWT adapted for mobile platforms.

Principles

In the reverse engineering phase, AWT2XIML analyses the structure of a GUI and produces an XIML model reflecting this structure. By using a GUI editor, adapted UIs can be created by modifying this specification: some modifications are done automatically but a designer has to complete the transformation step.

During the forward engineering phase, files containing the transformation rules for different platform profiles are applied to this XIML file allowing the generation of the Java UI. The forward engineering approach (and link with the old application logic) is based on library substitution and the half-object pattern. The half-object pattern is based on the “classical” Java AWT design pattern and relations, but divides it into two parts, one for the AWT server and the second for the AWT client. By using the half-object pattern, each component object needs to be split in two objects: one half in the application address space, the other half in the user interface address space. The two half objects communicate using a protocol (one of the TCPTE object communication layer).

The main benefit of this approach, compared to bridge patterns with remote peers, is that it reduces the number of remote method invocations (i.e. a method invocation of a container from a server (abstract) half-object is sent to the client half-object that propagates it to its components instead of several methods between server/client for each element of the container). Several methods of the AWT package have also been modified in TCPTE to limit the number of remote calls (i.e. some get methods return fixed values). Each component has a reference to one or more components that represent the server components on the client platform. Method mapping is handled by the client half object component that, for each method invocation, calls one or more platform specific methods on the referenced real components. Similarly, when an event occurs, the client listener sends the generated event to the correspondent concrete listener on server side.

Type of algorithm and rules

Some reverse engineering steps are achieved automatically by pattern matching, but the abstraction process has to be completed by manual modifications.

3.3 Comparison of the approaches

This section summarizes and compares the different approaches presented in this chapter thanks to eight parameters reflecting the scope and methods/techniques used during the process:

- The input UI.
- The output UI (or output model in a reverse engineering process), these two first parameters are used to measure the scope of the approach.
- the abstraction level reached to realize the reengineering/reverse engineering, to distinguish four categories of approaches: transcoding approaches (at FUI), up to an concrete UI description to apply a reengineering to platform similar to the source one, up to the abstract UI level for a very important modification of the UI such as in Morph or AUIDL and up to the task model (usually to merge several systems or for documentation).
- The technique(s) used during the process, which is usually pattern matching or transcoding. Other interesting techniques were also used such as trace analysis.
- The moment when the process takes place, statically (at design time) or dynamically (at run-time), the reengineering is applied on demand.
- The name of the abstract representation language (if any), and if this language was developed to support the reengineering process or if the approach exploited an existing language (marked by a *).
- The fact that several applications can be reengineered (and merged) or studied during the process (multi-application parameter). This parameter allows redistributing the UI in a better way as more information and possibilities are available to the tool or even to achieve retasking.
- The level of human intervention (none, low, moderate or high) in the process. The human intervention can happen at three different moment: during the reverse engineering (by resolving ambiguities that the tool can not resolve or by giving input values to the system in order to achieve the process), after the reverse engineering (by modifying/refining the results obtained by the tool in a specific editor or by finishing the process “by hand”), or before the process (by annotating the coding or by recording traces of execution).

Name	Input	Output	Abstraction level	RE Technique
Morph	Legacy systems	Motif, Java, windows UI	AUI	Pattern Matching
AUIDL	Cobol	IBM 3270	AUI	Pattern matching,

Chapter 3 State of the art in reverse engineering

				AST, control flow analysis
Cellest	Legacy systems	WML, XHTML	AUI/ T&C	Trace analysis
Csaba	Cobol	Windows UI	FUI	Wrapping
PUC	Complex appliance	Pocket PC/PDA UI	AUI	Hardware modification
Visual Tap	Any	Any (visual/ Braille/ vocal)	FUI	Video Signal Recognition / Pattern Matching
Digestor	HTML	HTML	FUI	Transcoding
MTA	HTML	WML	FUI	Transcoding
PIMA	Any	Any	CUI	Pattern Matching
Webrevenge	HTML	Task Model	T&C	Pattern Matching
Reweb	HTML	HTML	CUI	Pattern Matching
Ware	HTML	UML diagrams	CUI	Pattern Matching- Control Flow Analysis
Tamex	HTML	HTML	T&C	Pattern Matching
ADAPT	HTML	HTML	FUI	Transcoding
VoiceXML Transcoder	HTML	VoiceXML	FUI	Transcoding, Annotation
RE end-user..	Click's internal language	UIML, task model, documentation	CUI and T&C	Pattern Matching
Revangie	HTML	Application Model	CUI	Trace Analysis
AWT2XIML	Java	Java for mobile platforms	CUI	Pattern Matching/library substitution

Name	Static/ Dynamic	Language	Multi-application	Human Intervention
Morph	Static	/	No	Low
AUIDL	Static	AUIDL	No	Moderate
Cellest	Both	/	Yes (babel)	High
Csaba	Dynamic	No	No	High

Chapter 3 State of the art in reverse engineering

PUC	Static	/	No	High
Visual Tap	Dynamic	Universal Protocol	No	No
Digestor	Dynamic	/	No	No
MTA	Dynamic	/	No	No
PIMA	Both	/	No	Low
Webrevenge	Static	CTT *	No (sites)	No
Reweb	Static	DMS Domain*	No	Moderate
Ware	Both	IRF, RDB, UML *	No (sites)	Moderate
Tamex	Dynamic	/	Yes	Moderate
ADAPT	Dynamic	No	No	No
VoiceXML Transcoder	Dynamic	VXPL	No	No
RE end-user..	Static	UIML *	No(sites)	Low
Revangie	Dynamic	/	No(sites)	No-High
AWT2XIML	Static	XIML *	No	High

Table 3-1 Comparison of the re(verse)engineering approaches

Only two approaches are able to merge the results of the reverse engineering into one new application (Cellest and Tamex). Webrevenge, click's HTML reverse engineering, Revangie and Ware are also able to combine the results of the individual reverse engineering of Web pages into one global output (a task model, or UML diagram representing the entire site), but these processes do not offer a forward engineering process to use these amalgamated results.

The approaches dedicated to the Web reverse or reengineering are often dynamic approaches by nature, as the problem they solve is to make a requested HTML UI accessible to another (class of) platform(s), without having to reverse the Web site entirely. This can be understood for two reasons: first, the reengineering of HTML UIs would require a lot of resources if every HTML page had to be processed (and maybe not used). Moreover, the static approach can be forsaken as dynamic reengineering approach can give results that are satisfying enough in a relatively short period of time.

Usually, human intervention is needed in the reverse or reengineering process, as the results of an automatic process often require human expertise to resolve ambiguities or modify the model more logically. Indeed, the semantic structure of a UI can only be partially recovered, and human intervention will always be needed to enhance the

quality of the reverse engineered model. There is a particular case in the Revangie process: the human intervention level varies between none and high as the tool can be used in different modes. According to the authors, the mode that gives the best results is the mixed mode, reinforcing this idea that models generated by an automatic reverse engineering should always be “corrected”.

Finally, there is no reverse engineering approach of Web pages up to the abstract UI level (or an amodal representation of the UI). Most of these current approaches dedicated to Web are transcoding approaches peculiar to a couple of source-target platform, or tools allowing only the recovery of documentation. Web reverse engineering and reengineering approaches have a limited scope (e.g. Tamex is only for the couple (HTML, HTML) or AWT2XIML is for the couple (AWT,AWT-mobile)) and are achieved thanks to rigid rules and heuristics, allowing no or few control in the process.

3.4 Conclusion

It can be observed that a relatively important amount of reverse engineering approaches exist, each of these aiming either a particular source language, or if approaches have a similar source language (section 3.2), they have different objectives. Thanks to this chapter, we conclude that the conceptual reference framework is particularly useful to express the starting and finishing points as well as the path of a UI reverse engineering approach. This state of the art also shows that a wide diversity of approaches has been introduced and that the reference framework exhibits three values [Beau00]: explanatory, comparative and exploratory. It facilitates the understanding of their differences and similarities. This framework is also exploratory in that it identifies undercovered techniques: for instance, none of the above approaches really considers multiple levels of abstraction in the UI reverse engineering process. When some level is reached, it is usually unique and for a particular purpose only. Therefore, the reusability of the results is questionable. Similarly, the transformations which have been applied during the process are not always made explicit. We tried as much as possible to make them explicit in terms of abstractions, reifications, and translations, thus ensuring a common comparison point.

The next chapter describes the semi-formal notation elaborated to express the reverse engineering rules starting from various languages in a common representation. The approach presented in the chapter 5, 6, 7 and 8 has been developed to overcome the limitations identified in this state of the art and are based on the notation defined in chapter 4.

Chapter 4 Notation for reverse engineering derivation rules

The state of the art which has been established in the previous chapter identified a need for a common representation of the rules which have been used in the UI reverse engineering process. To address this shortcoming and our requirement for expressing our own effective design knowledge for reverse engineering, this chapter will introduce a notation for reverse engineering rules. For this purpose, this chapter is dedicated on model-to-model mappings, and represents a part of the conceptual comprehension step. This step is then followed by an operationalization step to analyze its implementation. The objective is to express reverse engineering rules in a standardized notation so that it can be operationalized and reused for other types of operationalizations. The chapter is decomposed into six sections. The first section introduces the mapping problem and specifies objectives of this chapter; the second section contains some basic mathematical definitions for graphs and trees supporting the notation; the third part defines tree-operations used in the notation and the fourth section provides some examples of mapping rules between two meta-models. The fifth section provides some examples of rules implementations and finally, the conclusion summarizes the common subproblems shared in the reverse engineering of the different language considered in this chapter, by classifying derivation rules according to their objective.

4.1 Introduction

4.1.1 Objectives and method

The objectives of this chapter is to define a notation able to represent formally the derivation rules in a standard formalism from any source language studied in this thesis to the CUI or AUI model expressed in UsiXML 1.4.6. This notation allows to represent mappings between two meta-models, a source meta-model (e.g. WML 1.1 language meta-model) and a target meta-model (e.g. UsiXML 1.4.6 CUI meta-model). The only conditions are that the source language is declarative, representable as a tree and that the control flow of the source code is straightforward. Therefore, this notation could be reused for other types of UI not analyzed in this thesis, such as XUL or cHTML. Using a unique formalism allows finding out similarities, and so identifying common parts of the mappings that can be generalized by enlarging their scope to groups of objects or across languages.

The method followed in this chapter is decomposed into four steps, which are depicted on figure 4-1 with their corresponding sections and appendices.

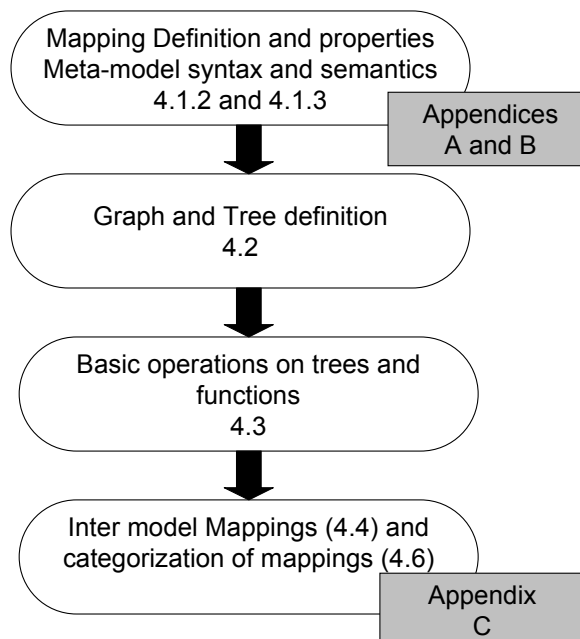


Figure 4-1 Structure of the chapter

- First, the mapping concept and the properties of mappings elaborated in this thesis are defined, so as the meta-models syntax and semantic (see the two next sections

4.1.2 and 4.1.3). Meta-models used in the mappings can be found in appendices A and B. These first definitions are the basis of the proposed notation.

- Secondly, some mathematical concepts related to graph and trees are specified (4.2). As instances (CUI or AUI models or declarative FUI) of meta-model can be represented in a tree structure, these concepts will be used in the notation to express the mappings between the meta-models.

- The third step consists in the definition of operations used in the notation, based on the mathematical definitions from previous section, allowing manipulating the trees and applying functions on nodes of the tree (section 4.3). Some of these operations are specific to the target meta-model (sections 4.3.1 and 4.3.3) and some of them can be applied on both (section 4.3.2).

- Finally, after the elaboration of all the material, the derivation rules are defined (section 4.4). This section only contains some examples of inter-model mappings, the complete list of mappings (inter and intra-model) can be found in appendix C. This chapter is ended by the identification of common subproblems for the reverse engineering of the various source models/languages studied in this thesis (section 4.6).

4.1.2 Mapping and notation definition

This section defines the mapping problem for the reverse engineering of UIs, and precises definitions as some terms are often used with different meanings in this field of research. Model-to-model mapping is a well-known problem in literature [Lope05, QVT03, Czar03, Capl02]. This new research field is strongly supported by the OMG group [OMG02] which has launched several research project based on this subject.

A *mapping* is defined in [Lope05] as the correspondence between elements of two metamodels, or in [QVT03] as the specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel. The notation developed in this thesis represents mappings between two models, a source model conforming to a language meta-model (appendix B) and a target model, the UsiXML CUI or AUI conforming to its meta-model (appendix A).

Following the QVT research group, mappings are to oppose to *relations* which are multi-directional transformation specifications. Relations are not executable in the sense that they are unable to create or alter a model: they can however check two or more models for consistency against one another. Typically relations are used in the specification stages of system development, or for checking the validity of a mapping. Mappings are transformation implementations. Unlike relations, mappings

Chapter 4 Notation for reverse engineering derivation rules

are potentially uni-directional and can return values. Mappings can refine any number relations, in which case the mapping must be consistent with the relations it refines. Both, relations and mappings, are considered as transformations [QVT03].

Mappings also have several attributes. A mapping is said to be *complete* if it preserves basic elements and relationships [Lope05]. It is a *direct* mapping if the metamodels are expressed in the same formalism [Capl02] and *indirect* in the other case. Finally, a mapping can be *unidirectional* (or *forward* mappings), if mappings can only be applied with defined source and target or *bidirectional* (*backward* mappings) if source and target can be swapped (by inverting the mappings) [Czar03].

The notation developed here can be considered as a direct complete and unidirectional mapping between two meta-models:

- The mappings are *direct* as the source and target meta-models are both expressed as class diagrams using the UML notation [uml]. For this purpose, each source language subject to reverse engineering has been abstracted in a meta-model which can be found in appendix B.
- The mappings are *unidirectional* (or *forward*), as the inversion of the mappings can not be used for generation.
- The mappings are complete as it keeps the elements and the relationships of the source model (subject to transformations) unchanged.

This notation is founded on those developed in [MCB99,02] on hypergraph data models, on graph theory [Gri94] and more particularly on trees [Ros99]. These transformations are either applied on a same tree T_T , or between two distinct trees, T_S and T_T , the first representing the source tree (instance of the HTML meta-model for example) and the second the target (instance of the UsiXML meta-model).

In this notation, a UI is described as a rooted and directed tree, where the hierarchy-branching represents a containment relation between the parent and the child element(s) (the parent contains the child element). Nodes of the tree represent the different elements composing the UI. The node name represents the widget class to which an element belongs, such as **checkbox**, **table** ... Each node can possess between 0 and n attributes (representing the attribute of the UI element such as its color, size, etc...). Referring to elements, attributes or functions is done in the same way as in a dotted notation, following the expression UIElement.attribute (for example Window.backgroundColor = white for an attribute or textComponent.

Chapter 4 Notation for reverse engineering derivation rules

parentNode = table for a function applied to the textComponent node; functions can also be called in the following manner parentNode (textComponent)).

Most of the rules are structured in the following manner:

$\forall x \in \text{SourceTree} : x = \text{elementA} \wedge (x.\text{attributeX} \neq \text{NULL} \vee x.\text{parentNode} = \text{"elementB"}) \rightarrow \text{AddNode}(\text{id}, \text{"elementC"})$

which means that for any node x belonging to a particular source tree, such that its node class (or name) is equal to “elementA” and either its “attributeX” exists or the parent node of the node x is equal to “elementB”, then a node “elementC” is added in the target tree. Conditions are linked thanks to logical operators AND(^), OR(∨) and can be negated by the NOT (!) operator. The different operations and functions required in this notation are defined in section 4.3.

4.1.3 Meta-model definition

Meta-models from appendix A represent the structure and composition of the UsiXML language at CUI and AUI levels. Similarly, the meta-models from appendix B represent the structure and composition of the HTML 4.0, WML 1.1, VoiceXML 2.0 and Windows resource files languages. The representation of the structure here means that every possible containment relations between elements are represented on these UML class diagrams. The complete list of attributes for each element is also represented on these meta-models. These diagrams include thus the same information contained in a DTD (put aside the type of the attributes). Most of these meta-models have been constructed thanks to the languages DTD (except for the resource files, the meta-model have been specified thanks to documentation).

The syntax of these meta-models is the UML class diagram graphical syntax. Elements are represented by classes, and their attributes are represented thanks to class attributes. Only two UML relations are used in the meta-models, inheritance relations (normal arrow) and aggregation relations (arrows ended with a rombus). Aggregation relations are used to describe that an element may contain another type of element, e.g. element <a> may contain an element (but not the contrary) in HTML 4.0. Generalization relations are used to facilitate the reading of meta-models by grouping elements with similar attributes and similar aggregation relations, but are not existing entities from the language.

4.2 Mathematical definitions

This section contains several mathematical definitions ([Cal77], [Gri94], [Ros99]) from the graph and tree theories used later to describe the derivation rules. As each meta-model can be represented as a tree-structure, these concepts are important as they allow us to express formally rules to define the mappings.

Graph definitions

- A **simple graph** $G=(V,E)$ consists of V , a nonempty set of vertices, and E , a set of pairs of distinct elements of V , called edges.
- A **directed graph** (V,E) consists of a set of vertices V and a set of edges E that are couples of elements of V . In a directed graph, vertices are called nodes and directed edges are called arcs.
- Two vertices u and v in directed graph G are called **adjacent** in G if $\{u,v\}$ is an edge of G . The vertex u is called the initial vertex of (u,v) and v is called the terminal or end vertex of (u,v) .
- A **Walk/Path(μ)** from x to y : Let x,y be vertices in a graph $G=(V,E)$. An x - y walk is a finite alternating sequence of vertices and edges from G , starting at vertex x and ending at vertex y and involving the n edges $e_i=\{x_{i-1}, x_i\}$ where $0<i<n+1$. The length of a walk is n , the number of edges in the walk. When the graph is simple, a path can be denoted by its vertex sequence.
- A **connected graph** is a graph where there is a walk from any edge to any other edge in the graph.
- A **simple walk** is a type of walk where no vertex occurs more than once.
- A **circuit** is a walk that begins and ends at the same vertex.

Tree definitions

- A **tree** is a connected undirected graph with no simple circuit, i.e. without walk beginning and ending at the same vertex and where no vertex occurs more than once.
- A **directed tree** is a directed graph (cf. supra) which would be a tree if the directions on the edges were ignored. Some authors restrict the phrase to the case where the edges are all directed towards a particular vertex, or all directed away from a particular vertex.
- A **root** is a node $v_i \in V$ such that $v_i \leq v_j \forall v_j \in V$.
- A **rooted tree** is a directed graph, where each arc is directed away from the root.
- The **preorder** t associated to G is a reflective and transitive relation $V \rightarrow V$ such that $\forall v_i, v_j \in V : v_i \leq v_j$ iff $v_i = v_j$ or $\exists \mu (v_i, v_j)$.

- A **tree traversal** is the process of visiting each node in a tree data structure. Tree traversal, also called **walking the tree**, provides for sequential processing of each node in what is, by nature, a non-sequential data structure. Such traversals are classified by the order in which the nodes are visited. In our case, the traversal is done from left to right, e.g. the first declared child of a node is analyzed before the others (in a depth-first order). Therefore, the preorder definition can be extended : $\forall v_i, v_j \in V : v_i \leq v_j$ iff $v_i = v_j$ or $\exists \mu (v_i, v_j)$ or $(\text{parent}(v_i) = \text{parent}(v_j))$ and v_i (on the left of) is declared before v_j .

Example:

```

<x1>
  <x2>
    <x31/>
    <x32/>
  </x2>
</x1>
```

On this small tree, the preorder as defined before this definition is $x1 < x2 < x31 = x32$, i.e. the two children $x31$ and $x32$ are on the same level, there is no preorder distinction for these two nodes. With the notion of tree traversal, the sequence of nodes is classified in this manner: $x1 < x2 < x31 < x32$. The node $x31$ comes before $x32$ as it has been declared before this second node.

- The **parent** of a node v is the unique node u such that there is an arc from u to v . In this case, the node v is said to be the **child** of u .
- A **leaf** is a node $v_i \in V$ such that $v_i \geq v_j \forall v_j \in V$
- The **ancestors** of a node other than the root are the nodes in the path from the root to this node, excluding the node itself and including the root (an ancestor of $v_j \in V$ is a node $v_i \in V$ such that $v_i \leq v_j$.)
- The **descendants** of a node v are those nodes that have v as an ancestor (a descendant of $v_i \in V$ is each $v_j \in V$ such that $v_i \leq v_j$).
- Each node is named, and this name represents the **class** of the element. Classes are types of elements belonging to the languages meta-models.
- An **attribute** is piece of information related to a specific node of the tree. Each attribute has one name and one or more associated values. Attributes names are defined in the languages meta-models.

4.3 Operations on trees

Several trees are used in this notation: T_t stands for the target tree, representing the concrete user interface. The notation is also valid to represent derivation rules up to abstract user interface, but all the rules of this chapter are specified for the concrete level. T_H corresponds to tree representing the UI specified in HTML, T_W for WML and T_V for voiceXML.

4.3.1 Basic operations on T_t

Several basic operations on the target tree are defined in this section. These operations are basic as they are needed to construct the resulting tree. There are three types of operations (i.e., adding, removing and modifying) which are applied to the three types of elements (node, attribute and arc) in the target tree. There is a fourth operation –CloneNode - for the elements of type node. This operation allows copying a node into another new node with each of its attributes.

In our notation, every arc is of the type “parent of” where the first parameter (source node) indicates the parent node and the second the child of the source node.

Each operation receives one or several nodes in parameters, but it is thanks to the identifier attribute of these nodes that the function is applied as the identifier is the unique way to reference a node without ambiguities.

The operations are specified by their name, followed by several parameters. Then, the arguments in and out are represented in natural language, so as the preconditions of these operations.

■ AddNode (class, id)

Args in: class name, identifier.

Args out: node (id) of class (class) created in the target tree.

Pre: id is unique; class belongs to the AUI or CUI meta-model.

Allows adding a node in the target tree. The first parameter indicates the class of the node and the second the identifier of the node.

■ RemoveNode (id)

Args in: identifier.

Args out: node (id) removed from the target tree.

Pre: identifier (id) exists in the target tree

Allows removing a node with identifier id in the target tree. This operation removes all the attributes belonging to this node, and all the children of this node are also removed.

■ ModifyNode (id,newclass,newid)

Args in: identifier, new class name (newclass), new identifier (newid).

Args out: node (id) is modified into a node of class newclass, with its id equal to newid.

Pre: class (newclass) belongs to the CUI or AUI models; id exists in the target tree ; newid is unique.

Allows modifying a node with identifier (id) in the target tree. The first parameter indicates the identifier of the node to modify, the second the new class of the node and the last the new identifier of the node.

■ **CloneNode (idSource, idClone,[tree])**

Args in: identifier (idSource), new identifier (idClone), tree name (tree).

Args out: copy of a subtree starting at node (idsource) in a new node (idClone) in the tree (tree).

Pre: idSource exists, idClone does not exist, tree is a valid tree name.

Allows creating a node with identifier idClone which is a copy of the source node having as identifier idSource. Every child node (and its attributes) of this source node is also copied. Arcs starting from this node are copied too (i.e. the arcs between the source node and its children) and thus an entire sub-tree can be reproduced thanks to this operation. The last parameter indicates the tree in which this node is cloned. If this parameter is not specified, the node is cloned into the same tree than the source node. The new node (idClone) is not linked with the tree, it has to be appended explicitly by another operation.

■ **AddAttribute(Nodeid, name, value)**

Args in: identifier (Nodeid), attribute name (name), attribute value (value).

Args out: attribute (name) with value (value) is added to node (nodeid).

Pre: (Nodeid) identifier exists, name is a valid attribute name from the CUI or AUI meta-model.

Allows adding an attribute to an element having as identifier Nodeid in the target tree. The first parameter indicates the identifier of the node to which the attributes will be added, the second one refers to the name of the attribute and the third one to the value of this attribute.

■ **RemoveAttribute(Nodeid, name)**

Args in: identifier, attribute name.

Args out: attribute from node (nodeid) removed.

Pre: identifier (Nodeid) exists, name is an attribute name belonging to the attributes of node (Nodeid).

Allows removing an attribute from a specific node with identifier Nodeid in the target tree. The first parameter indicates the identifier of the targeted node, the second the name of the attribute to remove.

■ **ModifyAttribute(Nodeid, name, newname, newvalue)**

Args in: identifier, attribute name, new attribute name, new attribute value.

Chapter 4 Notation for reverse engineering derivation rules

Args out: attribute (name) modified into another attribute (newname) with a new value.

Pre: identifier (Nodeid) exists, attributes names (name and newname) are a valid attribute names from the CUI or AUI meta-model, name is an attribute name belonging to the attributes of node (Nodeid).

Allows modifying an attribute in the target tree belonging to the node with identifier Nodeid. The first parameter indicates the identifier of the node, the second the name of the attribute, the third the new name of the attribute and the last the new value of the attribute.

■ AddArc (idSource,idTarget)

Args in: source identifier (idSource), target identifier (idTarget).

Args out: arc added between source node (idSource) and target (node idTarget).

Pre: identifiers exists and no arc exists between the two nodes (idSource, idTarget)

Allows adding an arc in the target tree. The first parameter represents the identifier of the source (parent) and the second one the identifier of the target of the arc.

■ RemoveArc (idSource, idTarget)

Args in: source identifier (idSource), target identifier (idTarget).

Args out: arc removed between source node (idSource) and target (node idTarget).

Pre: identifiers exists and an arc exists between the two nodes (idSource, idTarget)

Allows removing an arc in the target tree. The first parameter indicates the identifier of the source node and the second one the identifier of the target node of the arc.

■ ModifyArc (idSource, idTarget, newTarget)

Args in: source identifier (idSource), target identifier (idTarget), new identifier (newTarget).

Args out: the arc between the two nodes (idSource and idTarget) is removed, and a new arc between node (idSource) and node (newTarget).

Pre: identifiers exists, an arc exists between the two nodes (idSource, idTarget) where the node (idsource) is the source of the arc and the node idTarget) is the end of the arc; the node (newTarget) is not the target of any other arc.

Chapter 4 Notation for reverse engineering derivation rules

Allows modifying an arc in the target tree. The first parameter indicates the identifier of the source node and the second the target of the arc. The third parameter represents the identifier of the new target of the arc.

4.3.2 Derived operation (on T_H , T_W , T_V and T_T):

This section contains common derived functions (returning a value), which are based on the basic operations and the mathematical definitions of section 4.2. A small description is given for each operation to explain in natural language what the operation does. In and out arguments with their type followed by the precondition(s) are also mentioned for each function.

ParentNode (x): Returns y which is the parent of node 'x' (x.id). Equivalent to preceding (cf. defs)

Args in: node x

Args out: node y such that an arc exist between x and y where y is the source and x the target of the arc.

Pre: $x \in V$; x is not the root of the tree.

isLeaf (x): Returns true if node 'x' (x.id) is not the source of any parent arc. Equivalent to leaf (def)

Args in: node x.

Args out: boolean, equal to false if there is an arc starting from x.

Pre: $x \in V$

Sibling (x) : Returns the set of nodes $x_1 \dots x_n$ in which every node has the same parent node as 'x' (x.id), x excluded.

Args in: node x.

Args out: set of nodes having the same parent node as node x, x excluded.

Pre: $x \in V$; x is not the root of the tree.

Dist (x,y) : Returns the number of nodes between two sibling nodes x (x.id) and y (y.id), y included. For example, if x,z and y are the children of a node w and are specified in this order, the function Dist(x,y) will return 2.

Args in: nodes x and y.

Args out: integer representing the number of nodes between two sibling nodes.

Pre: $x,y \in V$; x and y have the same parent node.

Chapter 4 Notation for reverse engineering derivation rules

isTheAbstractionOf (itao(x)): Return y (belonging to T_t) such that y is the result of an abstraction of x ($x.id$). A relation is automatically added for each creation of nodes in the target tree that links an element from the source tree with the node created in the target tree. The function $isTheAbstractionOf(x)$ returns the node – already processed– belonging to the target tree corresponding to the node x from the source tree

Args in: node x .

Args out: node y from the target tree which is the abstraction of node x .

Pre: $x \in V$; node x has already been processed, i.e. a relation exist between element x and an element of the target tree.

isTheReificationOf (itro(x)) Returns y (belonging to $T_{H/V/W}$) such that y is the result of a reification of x ($x.id$). $Itro(x)$ is the inverse function of $itao(x)$.

Args in: node x .

Args out: node y from the source tree which is the the reification of node x .

Pre: $x \in V$

SiblingBefore(x) Return all the sibling nodes of node x ($x.id$) that are declared before ‘ x ’ (following the tree traversal defined in this method)

Args in: node x .

Args out: set of nodes that have the same parent node as node x and that are declared before node x .

Pre: $x \in V$; node x is not the root of the tree and not a leaf.

IsInPath(a,x) Returns true if the node ‘ a ’($a.id$) is contained in a path between the root and node x ($x.id$)

Args in: nodes a and x .

Args out: boolean; equal to true if node a exist in $\mu(\text{root},x)$.

Pre: $x, a \in V$; x is not the root of the tree.

IsInPath(a.attribute=value,x) Returns true if the node ‘ a ’ ($x.id$) with one of its attributes set to a particular value is contained in a path between the root and node x ($x.id$)

Args in: nodes a (with an attribute and its value) and x .

Args out: boolean; equal to true if node a with one of its attribute set to a specific value exist in $\mu(\text{root},x)$.

Pre: $x, a \in V$; the attribute of a is a valid attribute; x is not the root of

Chapter 4 Notation for reverse engineering derivation rules

	the tree.
NearestInPath(x,a)	Returns a node of class x which is the nearest of node a (a.id) in the path between the root and the node x; return false if node x is not in the same path as node a (a.id).
<u>Args in:</u>	nodes x and a.
<u>Args out:</u>	boolean; equal to false if x does not exist in $\mu(\text{root},a)$ or a node of class x such this node is exists in $\mu(\text{root},a)$ and that the number of arcs is minimal between x and a (if other instances of x exists in the same path).
<u>Pre:</u>	$x, a \in V$; a is not the root of the tree.
PathLength(x,y)	Returns the number of arcs between x.id and z.id such that $z = \text{NearestInPath}(x,y)$. If there is no path between x and y, the function returns infinity.
<u>Args in:</u>	nodes x and y.
<u>Args out:</u>	infinity if y does not exist in $\mu(\text{root},x)$ or an integer representing the amount of arcs between x and the node returned by $\text{NearestInPath}(x,y)$.
<u>Pre:</u>	$x,y \in V$; x is not the root of the tree.
isUIElement(x)	Returns true if the node x (x.id) belongs to a predefined set: [body,table,td,tr,input,fieldset,textarea,img,area,applet,embed,marquee,bgsound,select,option]}
<u>Args in:</u>	node x.
<u>Args out:</u>	Boolean, true if class x belongs to a specific set.
<u>Pre:</u>	$x \in V$
isAudio(x)	Returns true if the node x (x.id) possesses an attribute src containing an extension of an audio file, i.e. from the set [“.wav”, “.mp3”, “.aif”, “.au”, “.iff”, “.mid”, “.mod”, “.x3m”, “.voc”, “.mpa”, “.pcm”, “. Xml”, “.mp2”, “.pcm”, “.ra”, “.rm”, “.rpm”]
<u>Args in:</u>	node x.
<u>Args out:</u>	Boolean, true if class x belongs to a specific set.
<u>Pre:</u>	$x \in V$.
CountInPath(x,y,z)	Returns the numbers of nodes of class z in the path (x.id,y.id)
<u>Args in:</u>	nodes x, y and z.

Chapter 4 Notation for reverse engineering derivation rules

	<p><u>Args out:</u> integer representing the number of nodes of class z in $\mu(x,y)$.</p> <p><u>Pre:</u> $x, y \in V$; x is not the root of the tree</p>
ChildNodes(x)	Returns the set of nodes that have as parent node the node x ($x.id$)
	<p><u>Args in:</u> node x.</p> <p><u>Args out:</u> set of nodes that are the targets of arcs starting from node x.</p> <p><u>Pre:</u> $x \in V$; x is not a leaf.</p>
LeftSibling(x)	Returns the nearest node of x ($x.id$) in the set of nodes $siblingBefore(x.id)$. Returns null if x has no siblings before it.
	<p><u>Args in:</u> node x.</p> <p><u>Args out:</u> node y such that y belongs to $siblingBefore(x)$ and $dist(x,y)=1$, or null if $siblingBefore$ is empty.</p> <p><u>Pre:</u> $x \in V$.</p>
RightSibling(x)	Returns the nearest node x ($x.id$) in the set of siblings nodes without the set $siblingBefore$. Returns null if x has no siblings after it.
	<p><u>Args in:</u> node x.</p> <p><u>Args out:</u> node y such that y belongs to $(sibling(x)-siblingBefore(x))$ and $dist(x,y)=1$, or 'null' if $(sibling(x)-siblingBefore)$ is empty.</p> <p><u>Pre:</u> $x \in V$.</p>
CountInChildNodes(z,x)	Returns the numbers of nodes of class z that have the node x ($x.id$) as same parent node
	<p><u>Args in:</u> nodes z and x</p> <p><u>Args out:</u> integer representing the number of nodes of class z in the set $childnodes(x)$.</p> <p><u>Pre:</u> $x \in V$; x is not a leaf node.</p>

4.3.3 Remarks

- The id of an element is unique and is computed by counting the number of nodes created in the target tree (T_t) before the creation of the current node. The function $NodeAmount(T)$, which is present in each derivation rule implying the creation of nodes in the target tree, means the amount of nodes created at this point of the process in the target tree, and thus ensures that each node has a unique identifier.
- In a function call, constant values are indicated between quotation marks and computed values are indicated without quotation marks.

Chapter 4 Notation for reverse engineering derivation rules

- The rules of abstraction are gathered in function of the elements of the source trees. The rules displayed in bold must be executed before the other rules of this group. Rules in bold represent rules which will produce a new node in the target tree.
- When a node is created in the target tree, a relation `isAbstractedInto` is automatically added between the element that triggered the creation of the new node and this new node, allowing keeping a trace of the abstraction.
- Every element of the T_H , T_W and T_V source trees possess potentially an attribute `x.textnode` which is the text contained in the tag `x` (for example, for `<a> link number one `, `a.textnode` is 'link number one'). These attributes are not mentioned in the meta-models from appendix B.
- General rules belonging to inter-graph transformations are applied in a depth-first order.
- Every root element of the source file possesses an added attribute that do not belong the CUI specification, `x.filename` that can be used for tree merging operations (appendix C).
- Some tags are annotated with an 'optional' or a 'skipnode' marker, because the conditional behaviour of a VoiceXML UI can not be represented in UsiXML. These nodes are recorded in a comment and thus the choice is let to the designer to keep/discard the tags.

4.3.4 Groups of operations

These groups of operations are composed of several `addNode`, `addArc` and `addAttribute` operations allowing specifying components of the UsiXML language with one function call. As these components are repetitive, these shortcuts have been designed to clarify and decrease the size of the set of derivation rules (section 4.4 and appendix C). These groups are mostly used to define transitions, i.e. dialog component indicating the source and the target of a transition in the UI (between windows, vocal forms etc...).

An example of the resulting tree generated by the first group of operations, `AddGraphTr (source, target)`, is given below, so as its XML representation:

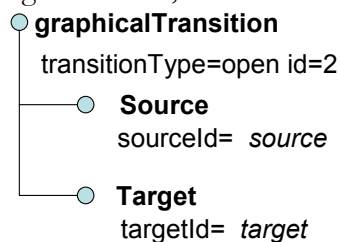


Figure 4-2 Subtree representing a `graphicalTransition`

Chapter 4 Notation for reverse engineering derivation rules

```
<graphicalTransition transitionType="open" id="2">
  <source sourceId=source />
  <target targetId=target />
</graphicalTransition>
```

which represent a transition in the graphical modality at the CUI level in UsiXML 1.4.6. Parameters of the functions are *source* and *target*, represented in italic on figure 4-2. Only these two elements vary in the declaration of a graphical transition in UsiXML.

Group name	Corresponding derivation rules																				
AddGraphTr (<i>source</i> , <i>target</i>)	<table border="1"> <tr><td>AddNode("graphicalTransition",idGrTr) where idGrTr = NodeAmount(T_t)</td><td>^</td></tr> <tr><td>AddAttribute(idGrTr, "transitionType", "open")</td><td>^</td></tr> <tr><td>AddAttribute(idGrTr, "id", idGrTr)</td><td>^</td></tr> <tr><td>AddNode("Source", idSource) where idSource = ∑ node ∈ T_t</td><td>^</td></tr> <tr><td>AddAttribute(idSource, "sourceId", <i>source</i>)</td><td>^</td></tr> <tr><td>AddNode("Target", idTarget) where idTarget = ∑ node ∈ T_t</td><td>^</td></tr> <tr><td>AddAttribute(idTarget, "targetId", <i>target</i>)</td><td>^</td></tr> <tr><td>AddArc(0,"1", idGrTr)</td><td>^</td></tr> <tr><td>AddArc(idGrTr, idSource)</td><td>^</td></tr> <tr><td>AddArc(idGrTr, idTarget)</td><td>^</td></tr> </table> <p>Allows adding GraphicalTransitions in only one operation. A GraphicalTransition is composed of a node, child of the "CUIModel" node, and possesses two children, one representing the source and the second representing the target of the transition.</p>	AddNode("graphicalTransition",idGrTr) where idGrTr = NodeAmount(T _t)	^	AddAttribute(idGrTr, "transitionType", "open")	^	AddAttribute(idGrTr, "id", idGrTr)	^	AddNode("Source", idSource) where idSource = ∑ node ∈ T _t	^	AddAttribute(idSource, "sourceId", <i>source</i>)	^	AddNode("Target", idTarget) where idTarget = ∑ node ∈ T _t	^	AddAttribute(idTarget, "targetId", <i>target</i>)	^	AddArc(0,"1", idGrTr)	^	AddArc(idGrTr, idSource)	^	AddArc(idGrTr, idTarget)	^
AddNode("graphicalTransition",idGrTr) where idGrTr = NodeAmount(T _t)	^																				
AddAttribute(idGrTr, "transitionType", "open")	^																				
AddAttribute(idGrTr, "id", idGrTr)	^																				
AddNode("Source", idSource) where idSource = ∑ node ∈ T _t	^																				
AddAttribute(idSource, "sourceId", <i>source</i>)	^																				
AddNode("Target", idTarget) where idTarget = ∑ node ∈ T _t	^																				
AddAttribute(idTarget, "targetId", <i>target</i>)	^																				
AddArc(0,"1", idGrTr)	^																				
AddArc(idGrTr, idSource)	^																				
AddArc(idGrTr, idTarget)	^																				
ConstrBox(name, type,[idbox])	<table border="1"> <tr><td>AddNode (<i>name</i>, <i>idbox</i>)[where idbox = NodeAmount(T_t)]</td><td>^</td></tr> <tr><td>AddAttribute (<i>idbox</i>, "type",type)</td><td>^</td></tr> <tr><td>AddAttribute (<i>idbox</i>, "isEnabled", "true")</td><td>^</td></tr> <tr><td>AddAttribute (<i>idbox</i>, "isVisible", "true")</td><td>^</td></tr> <tr><td>AddAttribute (<i>idbox</i>, "name", <i>idbox</i>)</td><td>^</td></tr> <tr><td>Add Attribute (<i>idbox</i>, "id", <i>idbox</i>)</td><td>^</td></tr> </table> <p>Allows adding a "construction" box node with 5 attributes. These nodes are replicates of boxes and needed in an intermediate step during the recovery of the layout of the UI. Therefore, they are named <i>bux</i>, <i>bax</i>, or <i>bix</i> to record the fact that they have already been processed and the step in which they have been processed.</p>	AddNode (<i>name</i> , <i>idbox</i>)[where idbox = NodeAmount(T _t)]	^	AddAttribute (<i>idbox</i> , "type",type)	^	AddAttribute (<i>idbox</i> , "isEnabled", "true")	^	AddAttribute (<i>idbox</i> , "isVisible", "true")	^	AddAttribute (<i>idbox</i> , "name", <i>idbox</i>)	^	Add Attribute (<i>idbox</i> , "id", <i>idbox</i>)	^								
AddNode (<i>name</i> , <i>idbox</i>)[where idbox = NodeAmount(T _t)]	^																				
AddAttribute (<i>idbox</i> , "type",type)	^																				
AddAttribute (<i>idbox</i> , "isEnabled", "true")	^																				
AddAttribute (<i>idbox</i> , "isVisible", "true")	^																				
AddAttribute (<i>idbox</i> , "name", <i>idbox</i>)	^																				
Add Attribute (<i>idbox</i> , "id", <i>idbox</i>)	^																				
AddAudiTr (<i>source</i> , <i>target</i>)	<table border="1"> <tr><td>AddNode("AuditoryTransition",idGrTr) where idAuTr =NodeAmount(T_t)</td><td>^</td></tr> <tr><td>AddAttribute(idAuTr, "transitionType", "open")</td><td>^</td></tr> </table>	AddNode("AuditoryTransition",idGrTr) where idAuTr =NodeAmount(T _t)	^	AddAttribute(idAuTr, "transitionType", "open")	^																
AddNode("AuditoryTransition",idGrTr) where idAuTr =NodeAmount(T _t)	^																				
AddAttribute(idAuTr, "transitionType", "open")	^																				

	AddAttribute(idAuTr, "id", idAuTr)	^
	AddNode("Source", idSource) where idSource = \sum node \in T _t	^
	AddAttribute(idSource, "source", <i>source</i>)	^
	AddNode("Target", idTarget) where idTarget = \sum node \in T _t	^
	AddAttribute(idTarget, "target", <i>target</i>)	^
	AddArc(0, "1", idAuTr)	^
	AddArc(idAuTr, idSource)	^
	AddArc(idAuTr, idTarget)	^
	Allows adding an AuditoryTransitions in only one operation. An AuditoryTransition is similar to a graphicalTransition . It is composed of a node, child of the "CUIModel" node, and possesses two children, one representing the source and the second representing the target of the transition.	
AddCUIDiag- Cont(source, target, symbol)	AddNode("CUIDialogControl", idCDC) where idCDC = NodeAmount(Tt)	^
	AddAttribute(idCDC, "symbol", symbol)	^
	AddAttribute(idCDC, "id", idCDC)	^
	AddNode("Source", idSource) where idSource = \sum node \in Tt	^
	AddAttribute(idSource, "source", <i>source</i>)	^
	AddNode("Target", idTarget) where idTarget = \sum node \in Tt	^
	AddAttribute(idTarget, "target", <i>target</i>)	^
	AddArc(0, "1", idCDC)	^
	AddArc(idCDC, idSource)	^
	AddArc(idCDC, idTarget)	^
	Allows adding CUI Dialog Control relation in only one operation. It is composed of a node, child of the "CUIModel" node, and possesses two children, one representing the source and the second representing the target of the relation. The CUI Dialog Control possesses an attribute symbol that specifies the type of control (sequential, concurrent,... elements)	

4.4 Inter-tree mappings

After the definition of the basic operations, and functions used in this notation, some examples of mappings between two trees (or meta-models) are given in this section. Another category of mappings exists, intra-tree mappings, which are mappings applied on the target tree uniquely. This second category of mappings can be found in appendix C.

The sequence in which the derivation rules are applied – and their corresponding section in this chapter - is depicted on figure 4-3.

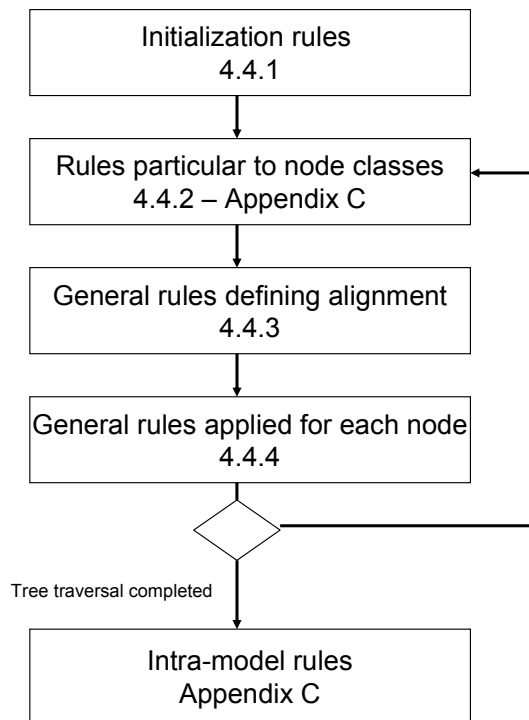


Figure 4-3 Sequence of derivation rules

The first set of rules (4.4.1) represents the initialization step. When the process of transformation starts, these rules are executed before any other derivation rule. Then, rules particular to node types (4.4.2) are applied. These derivation rules aims precise node classes, and maps them to node classes from the target meta-model. After these derivation rules, general rules are applied, which are used for the detection of the hierarchy (4.4.3) and the alignment properties of elements (4.4.4). General rules means that they are not specific to a particular node type. When the tree traversal is completed, intra-model rules are applied. These rules allow to correct the target tree, to recover the layout and to apply retargeting operations.

4.4.1 Initialization

The abstraction process starts with these 8 derivation rules. The root node in the target tree is created thanks to this initialization step. This root node contains information about the date of creation, the schema version of the UsiXML specification and the URL of the UsiXML schema used to specify this UI.

AddNode("cuiModel", "1")	^
AddAttribute("1", "name", "1")	^
AddAttribute("1", "id", "1")	^

AddAttribute ("schemaVersion", "1.4.6")	^
AddAttribute ("creationDate", "date")	^
AddAttribute ("xsi:schemaLocation", "http://www.UsiXML.org/spec usiXML-cui.xsd")	^
AddAttribute ("xmlns:xsi", " http://www.w3.org/2001/XMLSchema-instance ")	^
AddAttribute ("xmlns", "http://www.usiXML.org")	

4.4.2 Particular to node classes

This type of rules contains 46 groups of mappings. For each group, the first mapping in bold corresponds to the detection of a node in the source tree that causes the creation of a node in the target tree. The title of each group of rules is numbered (G1..G46) and followed by the class name of the source element of the mapping. The possible class names corresponding to this element in the target model is indicated between brackets in the group title. The complete list is given in appendix C, and this section only contains some commented group of rules. Some other commented examples are given in chapter 6, 7 and 8.

G4 - td (box/cell)

$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border=0$	$\rightarrow \text{AddNode}(\text{"box"}, \text{idbox})$ where $\text{idcell} = \text{NodeAmount}(Tt)$
$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border>0$	$\rightarrow \text{AddNode}(\text{"cell"}, \text{idcell})$ where $\text{idcell} = \text{NodeAmount}(Tt)$
$\forall x \in T_W : x = td$	$\rightarrow \text{AddNode}(\text{"cell"}, \text{idcell})$ where $\text{idcell} = \text{NodeAmount}(Tt)$
$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border=0$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"type"}, \text{"horizontal"})$
$\forall x \in T_{H/W} : x = td$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"isEnabled"}, \text{"true"})$
$\forall x \in T_{H/W} : x = td$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"isVisible"}, \text{"true"})$
$\forall x \in T_{H/W} : x = td$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"name"}, \text{idcell})$
$\forall x \in T_{H/W} : x = td$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"id"}, \text{idcell})$
$\forall x \in T_{H/W} : x = td$	$\rightarrow \text{CheckAligement}(x, \text{idcell})$
$\forall x \in T_H : x = td \wedge x.width!=NULL$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"width"}, x.width)$
$\forall x \in T_H : x = td \wedge x.height!=NULL$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"height"}, x.height)$
$\forall x \in T_H : x = td \wedge x.bgimage=NULL$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"bgimage"}, x.background)$
$\forall x \in T_H : x = td \wedge x.bgcolor!=NULL$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"bgcolor"}, x.bgcolor)$
$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border>0$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"xIndex"}, b)$ where $b = \sum_{\text{id}} \text{SiblingBefore}(x)$
$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border>0$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"yIndex"}, c):$ $y = \text{NearestInPath}(\text{tr}, x) \wedge c = \sum_{\text{tr}} \text{SiblingBefore}(y)$
$\forall x \in T_W : x = td$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"xIndex"}, b)$ where $b = \sum_{\text{id}} \text{SiblingBefore}(x)$
$\forall x \in T_W : x = td$	$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"yIndex"}, c) : y = \text{NearestInPath}(\text{tr}, x) \wedge$ $c = \sum_{\text{tr}} \text{SiblingBefore}(y)$
$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border=0$	$\rightarrow \text{AddArc}(i.id, x.id)$ where $i = \text{itao}(\text{NearestInPath}(\text{"tr"}, x))$

Chapter 4 Notation for reverse engineering derivation rules

$$\forall x \in T_H : x = td \wedge \text{NearestInPath}(\text{table}, x).border > 0 \rightarrow \text{AddArc}(i.id, idbox) \text{ where } i = \text{itao}(\text{NearestInPath}(\text{"table"}, x))$$

$$\forall x \in T_W : x = td \rightarrow \text{AddArc}(i.id, idbox) \text{ where } i = \text{itao}(\text{NearestInPath}(\text{"table"}, x))$$

This first group of rules defines the derivation rules for table cells. There are three bold rules (i.e. creating a node in the target tree), as the cell can be

1. a part of a **table** without border (derived then into a **box**)
2. a part of a **table** with borders (derived in a cell)
3. a part of a WML **table** (also derived in a cell).

In the first of these three cases, an attribute **type=horizontal** is automatically added as the content of the box will be displayed horizontally by default in HTML. The five next attributes are common for both languages (HTML and WML) and for each of the 3 cases: the attributes are **isEnabled**, **isVisible**, **name**, **id** and **alignment** attributes, as the rule can be applied on $T_{H/W}$ trees and has the only condition that the node name is **td**. The four following attributes are for HTML source trees only, but for both cases (i.e. with border greater or equal to 0) and add the **width**, **height**, **bgcolor** and **bgimage** attribute in the UsiXML specification if they are present in the source tree. The next attribute defines the values for the **xIndex** and **yIndex** in the CUI model. This rule is only applied if the parent **table** has a border greater than 0 (case number 2). These two attributes are computed by counting the number of siblings **td** (and **tr** nodes respectively) before the current node. The same occurs for the WML node, but without the border-condition as **tables** are never derived into **boxes** in this language.

Finally, the hierarchy is constructed (by adding an arc), either by linking the element to the abstraction of the nearest **table** - for the WML language or in the case of a parent table with border greater than 0- or to the abstraction of the nearest row (**tr**) in the case of **boxes**.

G6 – fieldset (box)

$$\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{Addnode}(\text{"box"}, idbox) \text{ where } idbox = \text{NodeAmount}(Tt) \rightarrow \text{AddAttribute}(idbox, \text{"type"}, \text{"horizontal"})$$

$$\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(idbox, \text{"name"}, idbox)$$

$$\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(idbox, \text{"isEnabled"}, \text{"true"})$$

$$\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(idbox, \text{"isVisible"}, \text{"true"})$$

$$\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(idbox, \text{"id"}, idbox)$$

$$\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{CheckAligment}(x, idbox)$$

$$\forall x \in T_H : x = \text{legend} \wedge \text{NearestInPath}(\text{fieldset}, x) \wedge x.textnode \neq \text{NULL} \rightarrow \text{AddAttribute}(i.id, \text{"borderTitle"}, x.textnode) : i = (\text{itao}(\text{NearestInPath}(\text{fieldset}, x)))$$

$$\forall x \in T_H : x = \text{legend} \wedge \text{NearestInPath}(\text{fieldset}, x) \wedge x.textnode \neq \text{NULL} \wedge x.align \neq \text{NULL} \rightarrow \text{AddAttribute}(i.id, \text{"borderTitleAlign"}, x.align) : i = (\text{itao}(\text{NearestInPath}(\text{fieldset}, x)))$$

$$\forall x \in T_W : x = \text{fieldset} \wedge x.title \neq \text{NULL} \rightarrow \text{AddAttribute}(idbox, \text{"borderTitle"}, x.title)$$

Chapter 4 Notation for reverse engineering derivation rules

In the next example, the rules for WML and HTML source tree are again very similar. If a **fieldset** node is detected, a **box** is created in the UsiXML specification with the **type=horizontal** attribute. As for the previous example, five attributes are common to both languages and are automatically added.

In an HTML tree, if a **legend** node is present in the descendant of the **fieldset** node, its **textnode** and **align** attribute are used to set the value of the **borderTitle** and **borderTitleAlign** attributes of the **box** in the target tree.

For a WML tree, the **borderTitle** information can be found in the **title** attribute of the **fieldset** node.

G28 – Form (vocalForm)

$\forall x \in T_V: x = \text{form} \rightarrow \text{Addnode}(\text{"vocalForm"}, \text{idform})$ where $\text{idform} = \text{NodeAmount}(Tt)$
$\forall x \in T_V: x = \text{form} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"id"}, x.\text{name})$
$\forall x \in T_V: x = \text{form} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"id"}, \text{idform})$
$\forall x \in T_V: x = \text{form} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"name"}, x.\text{name})$
$\forall x \in T_V: x = \text{form} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"name"}, \text{idform})$
$\forall x \in T_V: x = \text{form} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"currentValue"}, x.\text{expr})$
$\forall x, z \in T_V: x = \text{form} \wedge z = \text{grammar} \wedge z \in \text{childnodes}(x) \rightarrow \text{AddAttribute}(\text{idform}, \text{"isOrderIndependent"}, \text{"true"})$

The group of rules G28 is only applicable to the VoiceXML language (T_V). It derives a **form** node into a **vocalForm** element in the CUI model. If the **name** attribute is present, its value will be used to set the values of the **name** and **id** UsiXML attributes. In the other case, the **name** and **id** attributes are computed by counting the number of node already created in the target tree. The value of the **expr** attribute is copied in the **currentValue** attribute in the target tree. Finally, if a **grammar** node exists in the children of the **form**, then the attribute **isOrderIndependent** is set to **true** in the CUI model, as a grammar defined at the level of the **form** will be active until the **form** node is closed. The information related to the **grammar** are added later (in G38) to the **vocalForm**.

G43 – goto (vocalNavigation)

$\forall x \in T_V: x = \text{goto} \rightarrow \text{Addnode}(\text{"vocalNavigation"}, \text{idgoto})$ where $\text{idgoto} = \text{NodeAmount}(Tt)$
$\forall x \in T_V: x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"id"}, \text{idgoto})$
$\forall x \in T_V: x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"name"}, \text{idgoto})$
$\forall x \in T_V: x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"NavigationType"}, \text{"goto"})$
$\forall x \in T_V: x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"isBridgeable"}, \text{"false"})$
$\forall x \in T_V: x = \text{goto} \wedge x.\text{nextitem} \neq \text{NULL} \rightarrow \text{AddAudiTr}(\text{idgoto}, x.\text{nextitem})$

This last group of rules is specific to VoiceXML trees. It derives a **goto** element into a **vocalNavigation** node. **Name** and **id** are automatically associated with the element, so as the **navigationType** and **isBridgeable** attributes which are set to **goto** and **false**

Chapter 4 Notation for reverse engineering derivation rules

respectively. Finally, the `nextItem` attribute is used to add an `auditoryTransition` to the CUI model (see group of functions in section 4.3.4 for `AddAudiTr`).

4.4.3 General rule defining alignment

This rule is represented as a function, and they can be applied for every element. `ElementId` represents the current element in the target tree to which the attribute is added.

CheckAlignment(x,elementId)

$\forall x \in T_{H/W} : \text{IsInPath}(\text{center},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"middle"})$
$\forall x \in T_{H/W} : \text{IsInPath}(\text{div.align=center},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"middle"})$
$\forall x \in T_{H/W} : \text{IsInPath}(\text{div.align=right},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"right"})$
$\forall x \in T_{H/W} : \text{IsInPath}(\text{div.align=left},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"left"})$
$\forall x \in T_{H/W} : x.\text{align=left} \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"left"})$
$\forall x \in T_{H/W} : x.\text{align=right} \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"right"})$
$\forall x \in T_{H/W} : x.\text{align=center} \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"middle"})$
$\forall x \in T_{H/W} : x.\text{valign=bottom} \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueVertical"}, \text{"bottom"})$
$\forall x \in T_{H/W} : x.\text{valign=top} \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueVertical"}, \text{"top"})$
$\forall x \in T_{H/W} : x.\text{valign=middle} \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueVertical"}, \text{"middle"})$
$\forall x \in T_{H/W} : \text{IsInPath}(\text{div.valign=middle},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"middle"})$
$\forall x \in T_{H/W} : \text{IsInPath}(\text{div.valign=top},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"top"})$
$\forall x \in T_{H/W} : \text{IsInPath}(\text{div.valign=bottom},x) \rightarrow \text{AddAttribute}(\text{elementId}, \text{"glueHorizontal"}, \text{"bottom"})$

This function checks if the element `x` is in the same path as an alignment modifier (`div`, `center`, `align` attributes), and if it is the case, an attribute is added in the `UsiXML` specification to reflect the horizontal position of the element.

4.4.4 General rules applied for each node

These rules are applied once for each node creation (the `AddNode` operation) in the 46 first groups of rules from section 4.4.2 (and appendix C), except if an `AddArc` is specified in the group of rules. In this case, the `AddArc` operation overrides these general hierarchy construction rules.

The aim of these rules is to check if the current source element is embedded in a container (`table`, `table cell`, `table row` or `fieldset` for HTML/WML and `vxml`, `menu`, `initial`, `form`, `block`, `subdialog` for voiceXML) and reproduces the same structure, i.e. link the element to its corresponding container, in the target tree. If this is not the case, the new element is appended to (defined as child of) the first vertical `box` after the `window` node. The first set of rules can be applied if the source tree is an HTML or WML file.

Chapter 4 Notation for reverse engineering derivation rules

$$\forall x \in T_{H/W}, \exists y, z \in T_t: y = \text{itao}(x) \wedge \text{IsInPath}(\text{table}, x) \wedge (\text{PathLength}(\text{table}, x) < \text{PathLength}(\text{tr}, x)) \wedge (\text{PathLength}(\text{table}, x) < \text{PathLength}(\text{fieldset}, x)) \wedge (\text{PathLength}(\text{table}, x) < \text{PathLength}(\text{td}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{table}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a table and if no tr, fieldset or td node is nearer than the table. If the table is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z, (abstraction of the table in the target tree).

$$\forall x \in T_{H/W}, \exists y, z \in T_t: y = \text{itao}(x) \wedge \text{IsInPath}(\text{td}, x) \wedge (\text{PathLength}(\text{td}, x) < \text{PathLength}(\text{tr}, x)) \wedge (\text{PathLength}(\text{td}, x) < \text{PathLength}(\text{fieldset}, x)) \wedge (\text{PathLength}(\text{td}, x) < \text{PathLength}(\text{table}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{td}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a td node and if no tr, fieldset or table node is nearer than the td node. If the td node is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z, (abstraction of the td in the target tree).

$$\forall x \in T_{H/W}, \exists y, z \in T_t: y = \text{itao}(x) \wedge \text{IsInPath}(\text{fieldset}, x) \wedge (\text{PathLength}(\text{fieldset}, x) < \text{PathLength}(\text{tr}, x)) \wedge (\text{PathLength}(\text{fieldset}, x) < \text{PathLength}(\text{table}, x)) \wedge (\text{PathLength}(\text{fieldset}, x) < \text{PathLength}(\text{td}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{fieldset}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a fieldset and if no tr, table or td node is nearer than the fieldset. If the fieldset is the nearest element, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z, (abstraction of the fieldset in the target tree).

$$\forall x \in T_{H/W}, \exists y, z \in T_t: y = \text{itao}(x) \wedge \text{IsInPath}(\text{tr}, x) \wedge (\text{PathLength}(\text{tr}, x) < \text{PathLength}(\text{table}, x)) \wedge (\text{PathLength}(\text{tr}, x) < \text{PathLength}(\text{fieldset}, x)) \wedge (\text{PathLength}(\text{tr}, x) < \text{PathLength}(\text{td}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{tr}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a tr node and if no fieldset, table or td node is nearer than the tr node. If the fieldset is the nearest element, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z, (abstraction of the tr node in the target tree).

$$\forall x \in T_{H/W}, \exists y, z \in T_t: y = \text{itao}(x) \wedge \text{IsInPath}(\text{table}, x) = \text{false} \wedge \text{IsInPath}(\text{td}, x) = \text{false} \wedge \text{IsInPath}(\text{tr}, x) = \text{false} \wedge \text{IsInPath}(\text{fieldset}, x) = \text{false} \rightarrow z = \text{NearestInPath}(\text{window}, y) \wedge \text{AddArc}(\text{itao}(z).\text{id}, y.\text{id})$$

Checks if the node x is not in the same path as a table, td, tr, fieldset node. If it is the case, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z, (abstraction of the window in the target tree).

$$\forall x \in T_{H/W}, \exists y, z, w \in T_t: y = \text{itao}(x) \wedge x \in \text{childNodes}(\text{body}) \wedge z = \text{itao}(\text{body}) \wedge w = \text{box} \mid w \in \text{childNodes}(z) \wedge \text{siblingsBefore}(w) = \text{NULL} \rightarrow \text{AddArc}(w.\text{id}, y.\text{id})$$

Checks if the node x is not in the same path as a table, td, tr, fieldset node. If it is the case, then an arc is added between the node y (abstraction of the node x in the target tree) and the node w, (first box in the target tree that has no sibling before it).

$$\forall x \in T_w, \exists y, z \in T_t: y = \text{itao}(x) \wedge x \in \text{childNodes}(\text{card}) \rightarrow z = \text{itao}(\text{card}) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Chapter 4 Notation for reverse engineering derivation rules

Checks if the node x has a card as parent node. If it is the case, then an arc is added between y (abstraction of x) and the abstraction of the card node

An example of the application of these derivation rules is given below, in a tree representation (figure 4-4).

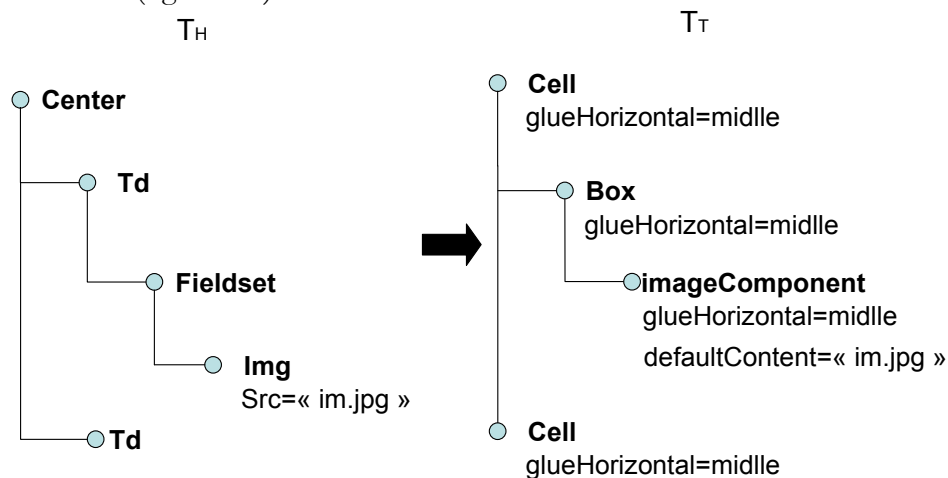


Figure 4-4 Hierarchy detection rules example

The excerpt of the source tree, T_H , is composed of a **center**, **td**, **fieldset**, **img** and **td** nodes. Thanks to the set of rules from section 4.4.3, it can be verified that all the elements of the source tree are in the same path as the **center** node. Therefore, they all inherit from the `glueHorizontal=middle` attribute. The **td**, **fieldset** and **img** nodes are derived in the target tree as **cells**, **box** and **imagecomponent** (by applying rules sets G4, G6 and G15). Then the rule sets from section 4.4.4 are used to define the hierarchy. The **fieldset** node has in its path a **td** node, and is thus appended to the abstraction of this **td** node (i.e. the first **cell** in the target tree). The **img** node has in its path a **td** and **fieldset**, but the path length between the **img** and the **fieldset** is shorter than the distance between the **img** and the **td** node, and is therefore appended to the abstraction of the **fieldset** (the **box** in the target tree). Finally, the last **cell** is not in the same path as any other element of the tree (except the **center** node), and is therefore put at the first level of the target tree.

The second set of rules presented in this section constructs the hierarchy for trees containing VoiceXML 2.0 files, by applying rules similar to the HTML and WML trees.

Chapter 4 Notation for reverse engineering derivation rules

$$\forall x \in T_V, \exists y, z \in T_t : y = \text{itao}(x) \wedge x \in \text{childNodes}(vxml) \rightarrow z = \text{itao}(vxml) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x has a $vxml$ node as parent node. If it is the case, then an arc is added between y (abstraction of x) and the abstraction of the $vxml$ node.

$$\forall x \in T_V, \exists y, z \in T_t : y = \text{itao}(x) \wedge \text{IsInPath}(\text{initial}, x) \wedge (\text{PathLength}(\text{initial}, x) < \text{PathLength}(\text{menu}, x)) \wedge (\text{PathLength}(\text{initial}, x) < \text{PathLength}(\text{form}, x)) \wedge (\text{PathLength}(\text{initial}, x) < \text{PathLength}(\text{block}, x)) \wedge (\text{PathLength}(\text{initial}, x) < \text{PathLength}(\text{subdialog}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{initial}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as an initial node and if no menu, form, block or subdialog node is nearer than the initial node. If the initial is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z , (abstraction of the initial in the target tree).

$$\forall x \in T_V, \exists y, z \in T_t : y = \text{itao}(x) \wedge \text{IsInPath}(\text{menu}, x) \wedge (\text{PathLength}(\text{menu}, x) < \text{PathLength}(\text{initial}, x)) \wedge (\text{PathLength}(\text{menu}, x) < \text{PathLength}(\text{form}, x)) \wedge (\text{PathLength}(\text{menu}, x) < \text{PathLength}(\text{block}, x)) \wedge (\text{PathLength}(\text{menu}, x) < \text{PathLength}(\text{subdialog}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{menu}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a menu and if no initial, form, block or subdialog node is nearer than the menu. If the menu is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z , (abstraction of the menu in the target tree).

$$\forall x \in T_V, \exists y, z \in T_t : y = \text{itao}(x) \wedge \text{IsInPath}(\text{form}, x) \wedge (\text{PathLength}(\text{form}, x) < \text{PathLength}(\text{initial}, x)) \wedge (\text{PathLength}(\text{form}, x) < \text{PathLength}(\text{menu}, x)) \wedge (\text{PathLength}(\text{form}, x) < \text{PathLength}(\text{block}, x)) \wedge (\text{PathLength}(\text{form}, x) < \text{PathLength}(\text{subdialog}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{form}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a form and if no initial, menu, block or subdialog node is nearer than the form. If the form is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z , (abstraction of the form in the target tree).

$$\forall x \in T_V, \exists y, z \in T_t : y = \text{itao}(x) \wedge \text{IsInPath}(\text{block}, x) \wedge (\text{PathLength}(\text{block}, x) < \text{PathLength}(\text{initial}, x)) \wedge (\text{PathLength}(\text{block}, x) < \text{PathLength}(\text{form}, x)) \wedge (\text{PathLength}(\text{block}, x) < \text{PathLength}(\text{menu}, x)) \wedge (\text{PathLength}(\text{block}, x) < \text{PathLength}(\text{subdialog}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{block}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Checks if the node x is in the same path as a block and if no menu, form, subdialog or initial node is nearer than the block. If the block is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z , (abstraction of the block in the target tree).

$$\forall x \in T_V, \exists y, z \in T_t : y = \text{itao}(x) \wedge \text{IsInPath}(\text{subdialog}, x) \wedge (\text{PathLength}(\text{subdialog}, x) < \text{PathLength}(\text{initial}, x)) \wedge (\text{PathLength}(\text{subdialog}, x) < \text{PathLength}(\text{form}, x)) \wedge (\text{PathLength}(\text{subdialog}, x) < \text{PathLength}(\text{block}, x)) \wedge (\text{PathLength}(\text{subdialog}, x) < \text{PathLength}(\text{menu}, x)) \rightarrow z = \text{itao}(\text{NearestInPath}(\text{subdialog}, x)) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

Chapter 4 Notation for reverse engineering derivation rules

Checks if the node x is in the same path as a subdialog and if no block, form, menu or initial node is nearer than the subdialog. If the subdialog is the nearest node, then an arc is added between the node y (abstraction of the node x in the target tree) and the node z , (abstraction of the subdialog in the target tree).

$$\forall x \in T_V, \exists y, z \in T_t: y = \text{itao}(x) \wedge \text{IsInPath}(\text{menu}, x) = \text{false} \wedge \text{IsInPath}(\text{block}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \wedge \text{IsInPath}(\text{form}, x) = \text{false} \wedge \text{IsInPath}(\text{subdialog}, x) = \text{false} \rightarrow z = \text{itao}(\text{vxml}) \wedge \text{AddArc}(z.\text{id}, y.\text{id})$$

If the node x is not in the same path as a subdialog, block, form, menu or initial node, an arc is added between the node y (abstraction of the node x in the target tree) and the abstraction of the vxml node.

The two next sets of rules are uniquely valid for VoiceXML trees. As the modifications of control flows are not represented in the UsiXML specification, several nodes have to be skipped during the transformation process of VoiceXML UI (see chapter 7). The nodes contained in [if-then-else-elseif-catch-filled-disconnect-exit-help-noinput-nomatch-throw-item-rule-tag-ruleref and one-of] are not processed and thus not recorded in UsiXML. Therefore before the application of any set of rules (between G22 and G46), the path of the node is checked to verify if it is not embedded in one of the 17 precited tags. If it is the case, the node is skipped. The same can occur with nodes possessing the cond attribute. The choice is let to the designer to recover those tags or not (see section 7.2.3)

$$\forall x \in T_V: \text{IsInPath}(\text{if}, x) \vee \text{IsInPath}(\text{then}, x) \vee \text{IsInPath}(\text{else}, x) \vee \text{IsInPath}(\text{elseif}, x) \vee \text{IsInPath}(\text{catch}, x) \vee \text{isInPath}(\text{help}, x) \vee \text{isInPath}(\text{exit}, x) \vee \text{isInPath}(\text{disconnect}, x) \vee \text{isInPath}(\text{noinput}, x) \vee \text{isInPath}(\text{throw}, x) \vee \text{isInPath}(\text{nomatch}, x) \vee \text{isInPath}(\text{item}, x) \vee \text{isInPath}(\text{rule}, x) \vee \text{isInPath}(\text{tag}, x) \vee \text{isInPath}(\text{ruleref}, x) \vee \text{isInPath}(\text{oneof}, x) \rightarrow \text{skip node (see remark) } \sim\text{optional}$$

$$\forall x \in T_V: x.\text{cond} \neq \text{NULL} \rightarrow \text{skip node (see remark) } \sim\text{optional}$$

This second set of rules is particular to voiceXML trees. It creates a vocalPrompt node in the target tree when an element possesses a textnode, and does not correspond to a typical node used to express outputs (i.e. audio, field, initial...). In this case, the textnode content is added in the target tree by putting it in the defaultContent attribute of a VocalPrompt node.

$$\forall x \in T_V: x.\text{textnode} \neq \text{NULL} \wedge x.\text{cond} = \text{NULL} \wedge \text{IsInPath}(\text{field}, x) = \text{false} \wedge \text{IsInPath}(\text{prosody}, x) = \text{false} \wedge \text{IsInPath}(\text{audio}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \wedge \text{IsInPath}(\text{filled}, x) = \text{false} \wedge \text{IsInPath}(\text{block}, x) = \text{false} \wedge \text{IsInPath}(\text{s}, x) = \text{false} \wedge \text{IsInPath}(\text{sentence}, x) = \text{false} \wedge \text{IsInPath}(\text{emphasis}, x) = \text{false} \rightarrow \text{skip node (see remark) } \sim\text{optional}$$

$$\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(Tt)$$

$$\rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$$

→ AddAttribute (idprompt, "name", idprompt)
 → AddAttribute (idprompt, "defaultContent", x.textnode)

4.5 Rule implementation

All the rules presented in this chapter can then be implemented, by using a DOM structure to parse the file (libxml.dll in PHP, msxml.dll in VB, xerces in Java...). These libraries propose several basic tree-manipulation functions, such as createNode, createAttribute, deleteNode, etc. and thus facilitate the implementation. Almost all the rules of section 4.4 can be implemented by using a simple condition (if... then). An example of the implementation of the rule about images is given below:

$\forall x \in T_S: x = \text{img} \rightarrow \text{Addnode}(\text{ImageComponent}, \text{idimage})$ where $\text{idimage} = \text{NodeAmount}(Tt)$
 $\forall x \in T_S: x = \text{img} \wedge x.\text{height} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageHeight"}, x.\text{height})$

would be implemented as

```

If x=img { y=createNode("ImageComponent")
If x=img{
    If !(attributeList=NULL) then
        foreach attribute do{
            If ((nextAttribute="height") && !(nextAttribute.value=NULL))
                y.addAttribute ("imageHeight",nextAttribute.value); }
    
```

Functions defined in section 4.3.1 and 4.3.3 are either already defined in the DOM library (such as parentNode, childNodes, etc...) or can be implemented as simple functions. An example in pseudo-code is given here to illustrate the implementation of the function CountInPath (x.id,y.id,z), which return the number of nodes of class z between the two nodes having the x.id and y.id identifiers.

As we can on this example, the tree traversal is achieved by an iterative function (here from node y towards the root).

```

Integer CountInPath(x.id, y.id, z){
    Node par=parentNode(y.id);
    Int count;
    While (par!=x.id && par !=root){
        If (par==z)
            count++;
        par=ParentNode(par);
    }
    If (par==root && root !=z)
        Count=0;
    Return (count)
    
```

In this thesis, the derivation rules have not been implemented automatically thanks to a rule-based system, but have been implemented systematically one by one.

4.6 Conclusion

After writing rules sets to derive elements from HTML, WML, VoiceXML languages (see annex C) and from Windows UI (see chapter 8) into the CUI level, it is possible to state some general conclusions about the reverse engineering of UIs by identifying common issues across the different languages. The reverse engineering of a declarative UI can be characterized by the following categories of rules defining the different sub-problems of the reverse engineering, and more particularly a reverse engineering applied in the UsiXML language.

The firsts two reverse engineering sub-problems are the recovery of the *elements* composing the UI and their *attributes*. This is done thanks to the rules sets from G0 to G46. These rules may be of the type 1-1 (G2 to G5, 7 to 20, 22, 23, 25, 28, 30 to 34, 36, 37, 39, 42 to 46) or 1-n (G1a, 26, 29), n-n (G1b, 24, 27, 35) and n-1 (G6, 21, 40, 41), meaning that an element of the source language may initiate the creation of 1 or n elements in the target language, or that a group of elements may be derived into one element or another group of elements at the CUI level.

Recovery of *attributes* can be achieved following three manners sorted by increasing order of complexity:

- An attribute can be created automatically in the target model, without any counterpart from the source model (e.g. the **isEnabled** and **isVisible** attributes which are mandatory attributes in UsiXML and are most of time added automatically in the resulting tree).
- An attribute can be detected straightforwardly from the code, by extracting its value, possibly apply a function on it, and copy it in the target model (e.g. color attributes in HTML 4.0 are copied in their hexadecimal form in the CUI model, or converted into an hexadecimal value if they are expressed in a literal value such as “blue”).
- An attribute can be the result of a more complex function that is applied on one or several elements from the source tree (e.g. the computation of the horizontal and vertical dimensions of a table in HTML 4.0).

Rules detecting the attributes of elements are contained in the rules sets G1 to G46. The third category of this analysis is the recovery of the *layout* of the UI for graphical UIs (HTML and WML). This question can be subdivided into two subcategories, the first about the division of the page into *boxes* and the second about the *position* of the elements in those boxes. The first sub-problem is resolved by applying the rule sets G22, G52 and G2 to G4 (the last two sets are for the case of tables without borders)

Chapter 4 Notation for reverse engineering derivation rules

and occurs after identifying individual elements. The position of elements can be detected thanks to the rules set of section 4.4.3. This problem was not solved for Windows UIs for both subcategories (see section 8.2 about working hypothesis).

The counterpart of the layout for the vocal modality is the *temporal ordering* of UI components. The sequence in which elements are declared defines the ordering in which they will be played. However, some possibilities exist to modify this straightforward control flow. It is possible to specify conditional behavior (not analyzed, see remark in section 4.4.4 and section 7.2), which may change this sequential specification. It is also possible to define pauses (G42 and G54) and form-active grammars (G23, 25, 27 to 29, 38 and 53) which also modify the control flow.

The fourth sub-problem, which can be linked with layout recovery problem, is the specification of *hierarchy* for the produced model, which is done by identifying the containment relations between elements in the source language (Section 4.4.4, G1a, 3, 4, 5 and 16).

The fifth category of rules is associated with the detection of *dialog* relations, i.e. mostly navigation (as scripts for markup language and the functional core for windows UI are ignored in this analysis and may contain complex dialog operations). Graphical and vocal transitions are defined within the rules set G1b, 11, 15 to 17, 21, 29, 33, 37, 41 and 43.

The sixth category is *multi-model transformations*, across different files or UI specifications. As presented in this chapter, these transformations are used to put different frames or subdialogs in a unique specification (G1a, 31, 47, 48, 48b and 49) but this kind of rules could also be reproduced to define navigation between several windows (by removing URLs and replacing them by ids of targeted windows) and thus *linking* models. This technique could also be used to merge the abstraction of various source UIs into a unique UsiXML specification.

The last category of reverse engineering operations is not related to reverse engineering of UIs in general, but more on the current approach. This group of derivation rule represents *retargeting operations*, ranging from adding more conditions on the derivation of one particular element to the transformation of one element in another type (some examples are given in G51).

Chapter 4 Notation for reverse engineering derivation rules

The seven different sub-problems of the reverse engineering of the various languages studied in this thesis are thus the detection of elements, attributes, layout (or temporal ordering), hierarchy, dialog, the transformations/merging of multiple models and the various retargeting operations.

The main advantage of this chapter consists in the definition and usage of a uniform semi-formal notation to express UI reverse engineering rules in terms of derivation rules. The derivation rules have been chosen because their complexity is at a low level and fulfilled the objectives of our notation. Other alternatives were considered, such as TXL [Malt93, Cord02] or OCL [OCL03], but these approaches were rejected as they were too complex for our purpose, or did not meet all the requirements of the desired notation. The benefit of this notation is that it combines ease of management and expressiveness. In this way, it is possible to render these rules explicit, thus making them permanent, independent of any future implementation of them, and reusable for other classes of similar problems. Beyond this advantage, we have been able to classify these rules, after some systematic handling through several UI languages, into families of rules which address some sub-problem of the general UI reverse engineering problem. In this way, we adopted an approach by reducing the problem into sub-problems, a classic problem solving approach in computer science. We also hope that these families could also be reused in other cases. They are also important for future generalization since each rule could be expanded into a larger one or into a set of rules to address a similar problem, but wider in scope. The notation introduced in this chapter and the rules expressed according to this notation will be used in two tools for supporting the UI reverse engineering process: Vaquita (Chapter 5) and ReversiXML (Chapter 6).

Chapter 5 Reverse Engineering of Web Sites: Vaquita

After being able to define rules for UI reverse engineering and families of rules to address one sub-problem at a time, some selected rules will be incorporated into a first tool for UI reverse engineering. This chapter contains the description of a method and a tool for the reverse engineering of the presentation model for HTML Web Pages. This chapter is divided into five parts: the first section describes the history of the analyzed language, the second is about the working hypothesis, the third section is dedicated to the HTML and XIML meta-models and contains some examples of derivation tables, the fourth section exemplifies the method presented in the chapter and finally, the last section analyzes the shortcomings of this first approach.

5.1 HTML history

The HTML language is a “descendant” of the SGML language as it has been described thanks to this meta-language. SGML is a language allowing describing markup language (such as XML, which is a subset of SGML) in particular those related to the exchange of electronic documents, the management and publication of documents. The SGML has been first defined in the late sixties at IBM research labs, but has only become a true standard (ISO 8879:1986) in the mid-eighties. As the SGML language is very flexible, it has almost not been modified or enhanced. This flexibility has a price, and this price is a very high level of complexity, which has limited and slowed down its adoption in a diversity of environments. The aim of this language was to separate the content of a

document from its presentation. The final intention was to produce several different printed versions of a document coming from only one source. [Xhtm] The HTML language, while following the same principle, has avoided this complexity by defining a small set of markups (tags) necessary to the production of documents and by simplifying the structure of documents. The HTML language is presentation-oriented, i.e. in which markups are defined to display titles, paragraphs, hypertext links... but nothing was defined in HTML to categorize the data contained in documents semantically. In the beginning, HTML was designed for the exchange of scientific documents. The inventor of the HTML, Tim Brenners-Lee, was working in the computer services of CERN when he had the idea to make different scientific documents accessible that would be linked together, instead of static files servers. But its simplicity, and the expansion of the Internet, contributed to the adoption of the format, ensuring HTML a tremendous growth. The fact that the language was based on SGML also facilitated very much its adoption, as it was based on a reliable and well-known language [Addi98]. The HTML only allows specifying document's presentation and is static in the sense that the possible interactions on a page are link-navigation and the sending of information (forms). However, it is possible to put dynamicity in HTML pages by including other languages such as javascripts, VBscripts... in the HTML code.

The first version, as invented by Tim Brenners-Lee, was developed in 1989, but it was a very first draft (he used text files with very few formatting tags). The first version widely accepted by the Internet community was elaborated in 1993. HTML 2.0 came in 1994, and in 1995 the third version of language was released. In September 1995, the version 3.2 of the language became a W3C recommendation. HTML 4.0 was launched in 1997, but became a W3C recommendation only one year later, in 1998. This last version is the most widely used version for the moment. The HTML was also adapted to the XML trends in 1999, by adapting the language to the XML grammar/syntax rules while keeping the same concepts of the HTML 4.0 in the XHTML 1.0 language [Bloo].

5.2 Working hypotheses

For this first study of the HTML language, the analysis of the language is focused on the presentation aspects, i.e. the composition of static web pages, by targeting the reverse engineering of the presentation model. The aim here is to recover an

abstract specification of the different widgets composing an UI, without the aspects relative to the dialog (navigation between pages, modification of the appearance of the UI at run-time etc...). Another limitation of this first study is that the layout of the UI is not recovered in the produced model. The limitation to the presentation model has been chosen for several reasons:

- It allows us to simplify the reverse engineering problem, by analyzing one page at a time.
- It also permits us to ignore embedded scripts (such as javascripts) that would require the analysis of imperative languages, which is another type of reverse engineering.
- Moreover, it is the most widespread model and the less complex to manage.
- It also constitutes the basis for other models (i.e. presentation elements use domain elements, dialog occurs between two or more presentation elements ...) and is therefore an important model in an abstract UI specification.

An important aspect introduced in this research is the flexibility of the reverse engineering process. The designer is able to choose the different reverse engineering rules (rows of derivation tables) to be applied for a particular case. Moreover, it is possible to choose, for some elements, between different transformations (e.g. merge sequential labels, transform radio buttons in drop down list boxes etc...). Thanks to this flexibility, it is possible to recover multiple UI description from one HTML page, and thus *retarget* a UI into a presentation model that suits a particular case: the aim of the reverse engineering could be to reengineer the UI for a monochrome platform, or a PDA with small bandwidth. In those cases, the designer can choose to ignore some types of elements (sounds, images) to decrease the size of the web page, or to select particular transformation rules to save screen space.

The reverse engineering method as applied in this chapter is thus a flexible static analysis of HTML web pages in order to recover the presentation model. The language that has been chosen to express this model is the XIML (see section 5.3). The corresponding level in the Cameleon reference framework is the concrete UI level, as the definition of elements is still dependent on the interaction modality (the graphical modality).

Losses due to the reverse engineering

By limiting the analysis to the presentation model, all the interactive and dynamic behaviours are obviously lost. At the presentation level, losses are of two types: as stated previously, the layout is not analyzed in this first study (only the horizontal layout is recorded for some elements) and the semantic relation (i.e. XIML relations that show that the elements are semantically linked, such as a radio button with its label) are not recovered either.

5.3 HTML and XIML meta-models and derivation tables

The presentation model is written in the eXtensible user-Interface Markup Language [Ximl, Eise00, Eise01], a language developed by Redwhale Software, derived from XML and able to store the models developed in MIMIC [Puer96]. MIMIC is meta-language that structures and organizes interface models. It divides the interface into model components: user-task, presentation, domain, dialog, user and design models. The design model contains all the mappings between elements belonging to the other models. The XIML is thus the updated XML version of this previous language.

The XIML language is mainly composed of four types of components: models, elements, attributes and relations between the elements.

Model: we can distinguish two types of models, the **interface** model and the **model components**. The first is the root of any XIML document and contains the various sub-models (model components) available in XIML. All the types of models do not have to be present and the same type of model-component can exist several times under the same interface model. The model components (task, domain, user, presentation, dialogue, platform, preferences and the general model) contain information specific to a dimension of the interface.

Element: is information describing a model-component. In the case of presentation elements, it is a unit of information describing the visual appearance of a user interface. Each presentation element may contain other presentation elements (until the simple CIO, i.e. indecomposable). For example a **window** is a presentation element which can contain the presentation element **table** which contains itself other presentation elements, etc... A presentation element can refer to an object external to the XIML code, for example an ActiveX control, an image etc... In this case, the attribute **location** can be used in order to specify a URL where the object could be found.

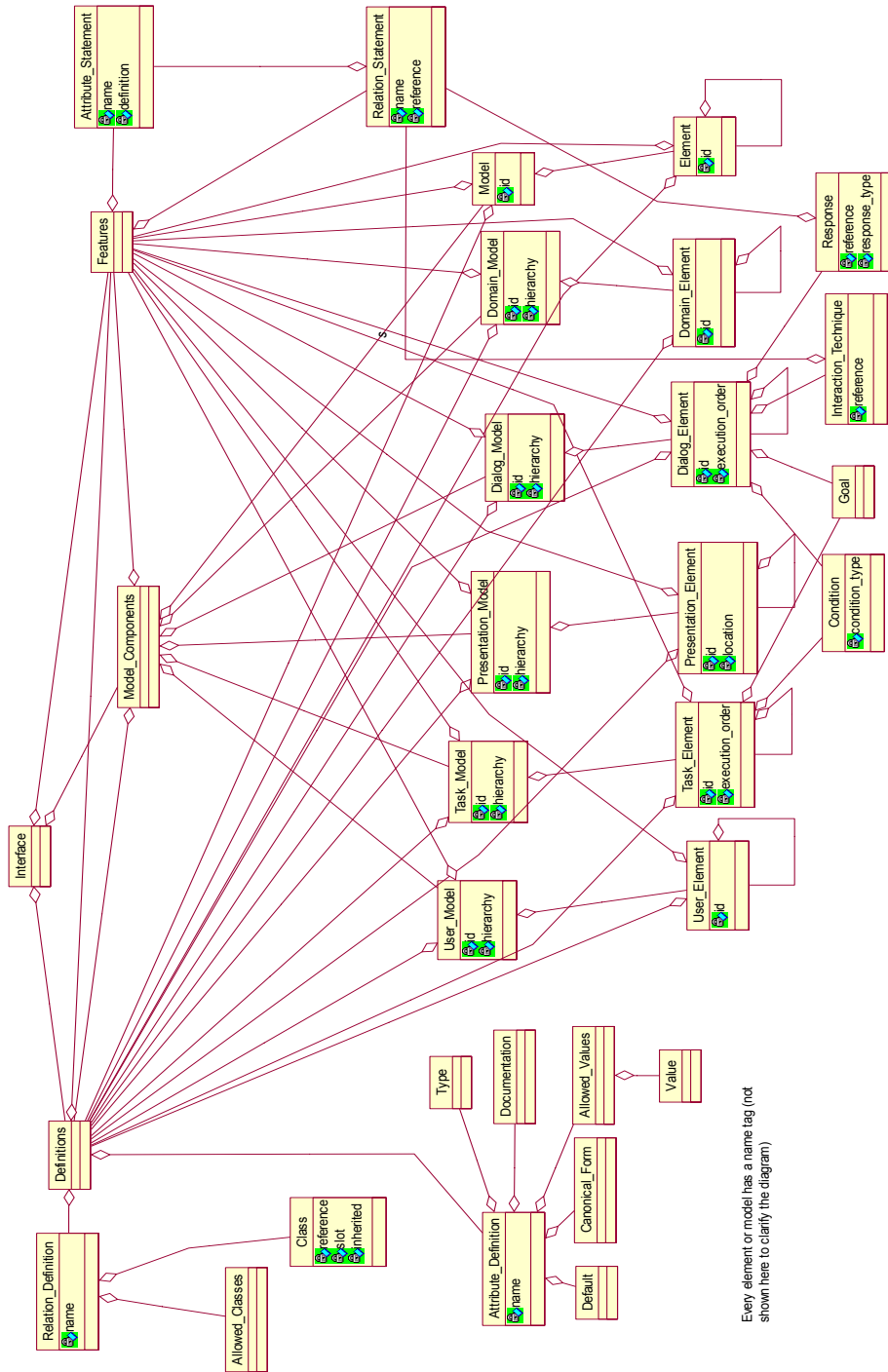


Figure 5-1 The meta-model of XIIML

Attribute: represents a simple unit of declarative information in connection with an interface model, a model-component or an element. It is always necessary to define an attribute for then being able to declare it in a component. The definition of the attribute is composed of the list of its allowed values, the default value of the attribute, its canonical form, documentation (information on what it represents) and of its type.

Relation: defines a link between the elements and/or the models. The element to which the link refers is specified thanks to the **reference** attribute, containing the identifier to which the object is referred. Any relation must be defined before its use (similarly to attributes). This definition contains the classes allowed in the relation.

The presentation model is composed of several embedded elements, which correspond to the widgets of the UI, and attributes of these elements representing their characteristics (colour, size...). The relations at the presentation level are mainly the links between labels and the widgets that these labels describe. The complete meta-model for the XIIML language is shown on figure 5-1. Note that such a meta-model was not provided by the XIIML consortium itself. Therefore, we recomposed it from the Document Type Definition and from the published articles.

A meta-model for the HTML language is given in appendix B. There are five categories of elements for the HTML meta-model: elements specific to the head section (**meta**, **script**...), containers, (such as **forms**, **tables**...) that define hierarchy of elements, formatting tags (such as **b**, **i**, **p**...), lists (**dl**, **ul**...) and atomic tags that cannot contain other tags (**img**, **object**, **button**...).

Some example of derivation tables are given in the rest of this section to illustrate how a HTML file can be reverse engineered into a XIIML presentation model. For each table, the left column represents the XIIML element and attributes and the right column the corresponding HTML tag and its attributes.

Image	img
ID	filename+ <i>IMG</i>
ImageFile	Src=URL
ImageHeight	Height=
ImageWidth	Width=
ImageType	Filename's Extension

ImageHorizSpace	Hspace=
ImageVertSpace	Vspace=
ImageAlternativeText	alt=
ImageBorder	Border=
ImageAlign	align=
ImageLowResFile	Lowsrc=URL (Netscape)

Table 5-1 Derivation table for images

This first derivation table derives an `img` tag into an image in the XIML specification. Most of the attributes are copied in XIML such as they are specified in HTML. A specific attribute, `filetype`, requires a small processing. It contains the file's extension and represents the type of the image (such as `jpg` or `bmp`), allowing designer to filter images based on their types (in the reverse engineering options).

radio button	input type=radio
ID	name=+ <i>RadioButton</i>
RadioBtNumber	Number of radio buttons with the same name attribute
RadioBtDefaultState = checked	Checked

Table 5-2 Derivation table for radio buttons

When an `input` node with its `type` attribute set to `radio` is detected, a `radioButton` is added in the XIML specification. The value of `radioBtNumber` requires a small algorithm to be applied: the entire HTML page is parsed to find out and count other radio buttons with same `name` attribute. The last attribute, `radioBTDefaultState` is set to "checked" only if the "checked" attribute is specified in the HTML code.

extended edit box	textarea
ID	name=+ <i>EDM</i>
EDMNumberOfLines	rows=
EDMNumberOfCols	cols
EDMEnabled = false	disabled
EDMDefaultString	textnode

Table 5-3 Derivation table for edit boxes

A `textarea` tag is transformed in XIML under the form of an `extended edit box`. The `rows` and `cols` of the `textarea` determine the values of the `EDMNumberOfLines` and `EDMNumberOfCols` respectively. The `EDMEnabled` attribute is set to false only

if the `disabled` attribute is present in the HTML code. Finally, the `defaultString` corresponds to the `textnode` of the `textarea` (if it exists).

5.4 Tool support and example

5.4.1 Vaquita

A first tool for the abstraction of Web Pages was developed [Boui01, Vand01, Boui2a]. This tool, named Vaquita for ‘reVerse engineering of Applications by Questions, Information Selection, and Transformation Alternatives’, allows a flexible reverse engineering of web pages written in HTML. This tool is accessible at <http://www.isys.ucl.ac.be/bchi/research/vaquita.htm>. The input of Vaquita is any web page which has to be saved locally and transformed into valid XML. Vaquita transforms this web page into a presentation model, at the concrete UI level in the Cameleon Reference Framework as shown on the figure 5-2. This presentation model can be transformed for another context of use, thanks to different options and heuristics. The tool has been implemented in Visual Basic 6 and is about 10,000 lines of code long. Vaquita can only reverse engineer one page at a time and process it locally (on the designer’s computing platform).

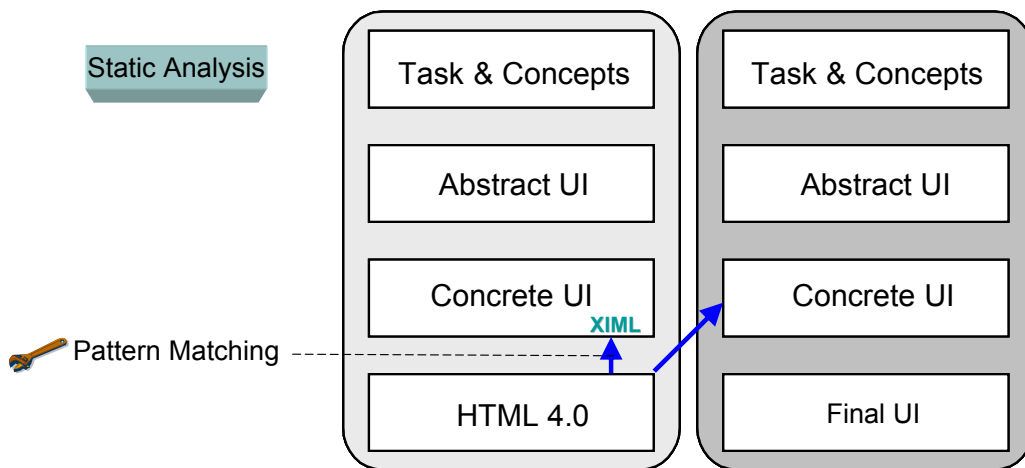


Figure 5-2 Vaquita in the cameleon reference framework

The complete process in which VAQUITA takes part is represented on figure 5-3. The Web page is first saved locally and its XML compliance is checked. If the page does not respect the XML grammar, the HTML page is syntactically corrected, thanks to the TidyCom library [tidy].

The use of this library offers several advantages: first, it makes sure we always have the same code format; secondly it cuts useless tags off (such as empty paragraphs: <p></p>), so that the process time of Vaquita is reduced; and finally it transforms the HTML into the XML format, allowing us to use the msxml3 library [msxml]. This library offers multiple functions for markup languages, such as tag retrieval, creation, modification, etc. The DOM structure is then extracted and the targeted mapping rules are applied. These rules are loaded at the beginning of the reverse engineering process. The groups of mapping rules are stored in a knowledge base, with one separate file for each platform. These mappings are the more probable ones, but designers can modify them easily to customize mapping rules for a specific Web site.

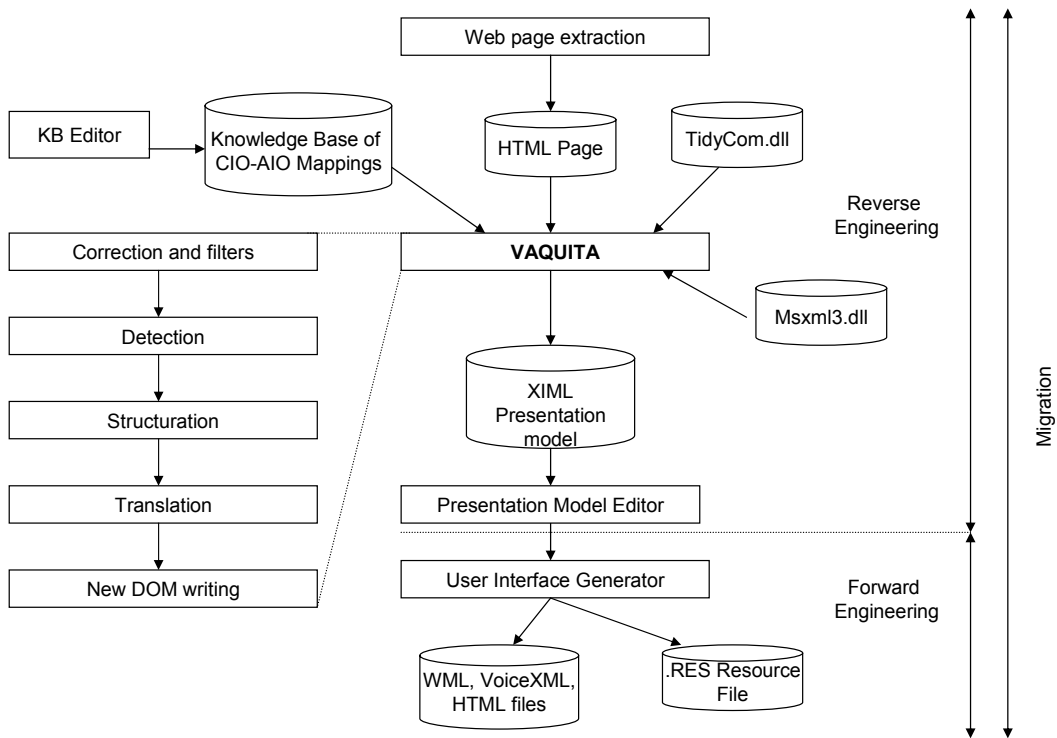


Figure 5-3 The complete reengineering process with Vaquita

Figure 5-4 represents the main screen of Vaquita. This tool is available into two languages, French and English, and the user can switch the language on the option page. The upper part of the screen contains the HTML code, the left part

contains the presentation model in a tree structure and the right part displays the XIML code of an element selected in the tree. This view can be switched with a trees-view that displays both trees, HTML and XIML trees, simultaneously on screen to check the results differently.

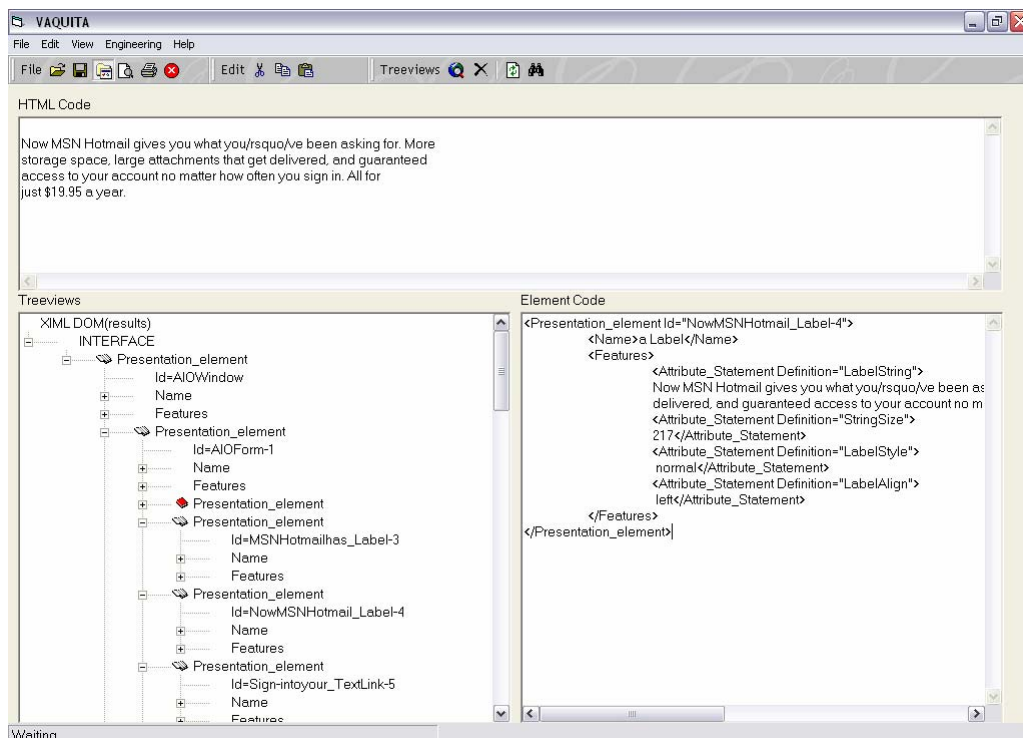


Figure 5-4 Main screen of Vaquita

The figure 5-5 represents the option page of Vaquita. On this page, the designer can choose to detect or ignore HTML elements during the reverse engineering. This allows the designer to directly extract a presentation model suited for a particular platform or context of use. She can for example uncheck sound widgets because the target platform is not equipped with sufficient sound capabilities. He can also remove colour attributes for monochrome platforms. The user can also choose between several reverse engineering heuristics on this option page: he can choose to remove **table** without borders, as this way of structuring the content is particular to HTML. Another example is the folding option which assembles consecutive labels into a label or splits big labels in several smaller labels. All these options ensure a flexible reverse engineering. The designer can modify the process in order to obtain a user interface specification directly adapted for a specific case and save this configuration for a future usage.

This specification can later be used in the forward engineering phase to produce a new UI suited for a particular type of platforms.

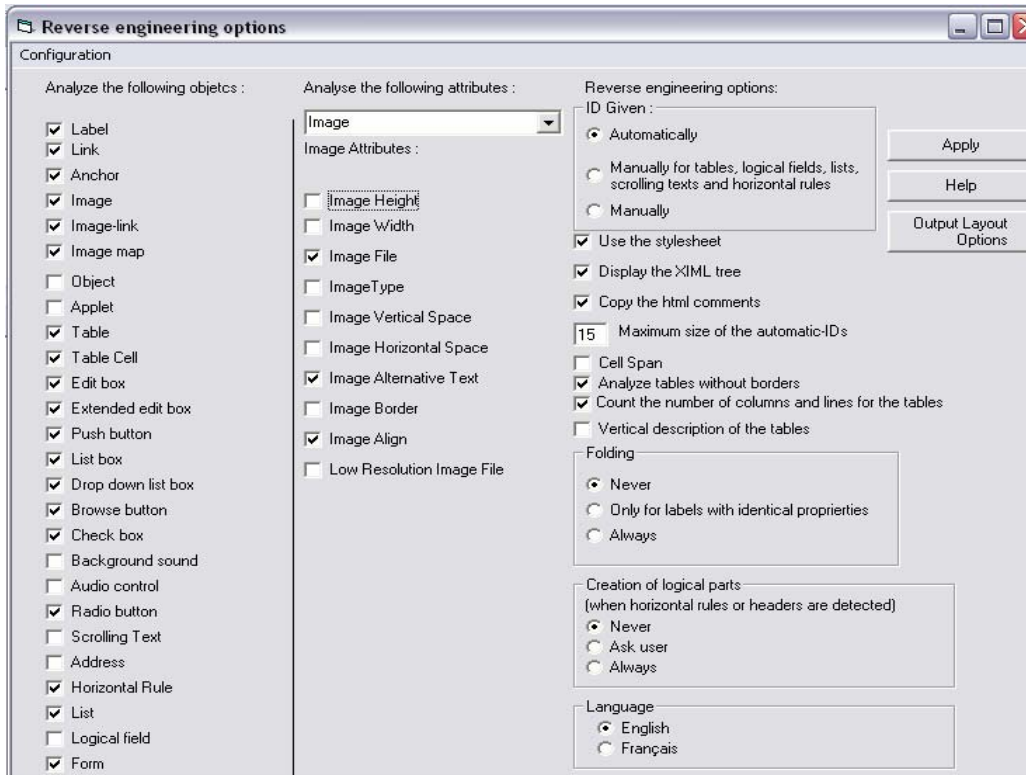


Figure 5-5 Options page of Vaquita

An example of an HTML page reverse engineered with Vaquita is presented in appendix H-1.

5.4.2 Example of a complete reengineering thanks to Envir3D

Forward engineering is supported to transform the XIML specifications resulting from the reverse engineering to a final UI [Boui04].

Thanks to Envir3D [Chow02], it is possible to produce VRML code from a XIML presentation model. Originally, Envir3D generates three-dimensional representations of control room. A drawing panel allows the developer to define the layout of the control room (figure 5-6 illustrates the design of a control room), and the corresponding XIML presentation model is generated from this graphical representation.

Chapter 5 Reverse Engineering of Web Sites: Vaquita

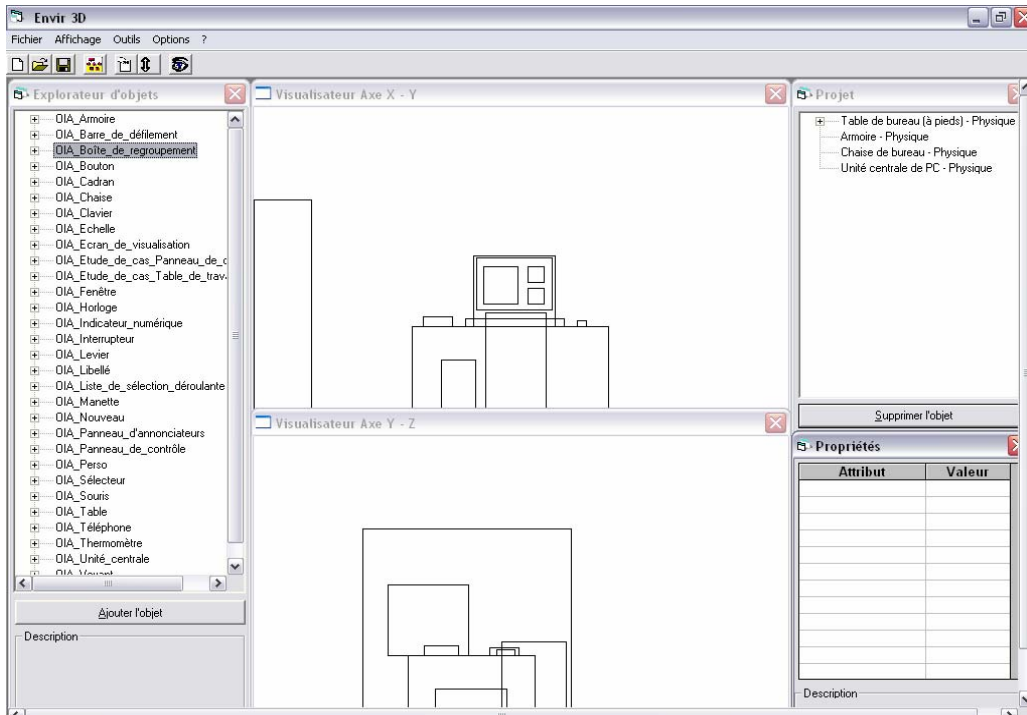


Figure 5-6 Creation of a control room with Envir3D

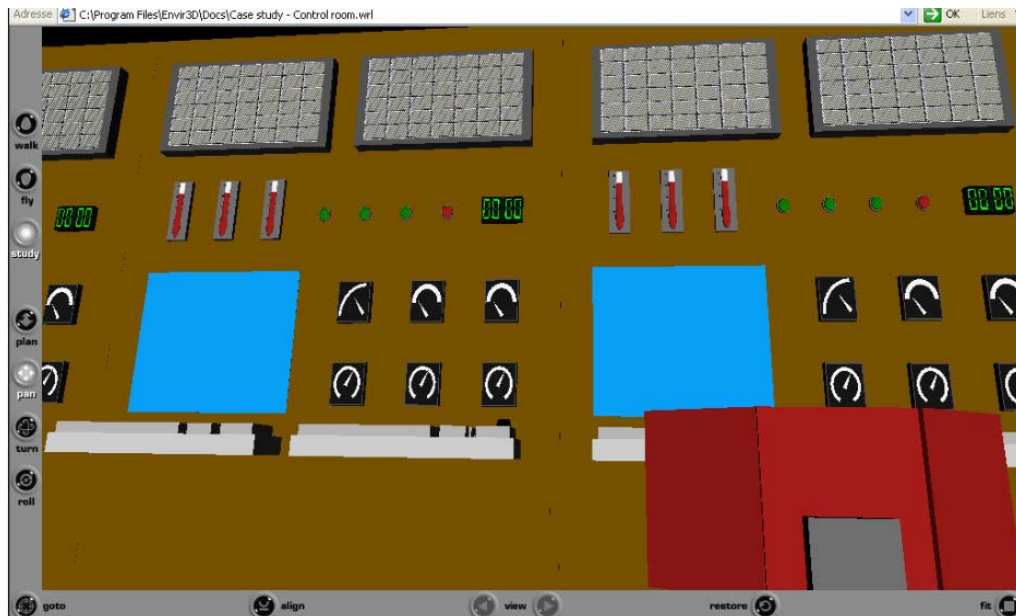


Figure 5-7 Control room generated from a XIIML specification in Envir3D

In a second phase of forward engineering, the presentation model can be used as input to produce a VRML file. Figure 5-7 represents a VRML file representing another control room generated with Envir3D, visualized in a web browser.

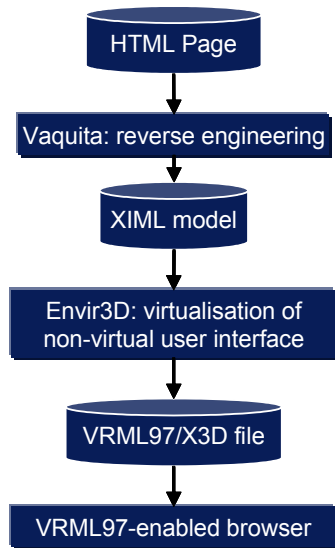


Figure 5- 8 Virtualization process.

Only the second functionality of Envir3D is used in our approach, the generation of a VRML file based on a presentation model. The model-based approach can be exploited to migrate existing non-virtual UIs to the virtual world. This process is called *virtualization*, as it transforms an existing 2D UI into its 3D equivalent. This process is depicted in Fig. 5-8. The combination of reverse and forward engineering thus closes the loop to obtain reengineering, so that UIs can be integrated in desktop virtual environments (DVE).

The figure 5-9 represents a complete reengineering process. The part A of this figure represents the original UI, which the CHI 2001 registration form (it represents a typical Web registration form). The part B is the result of the reverse engineering obtained thanks to Vaquita. The excerpt of the UI description is the XIML specification for a label and a textbox. This specification is translated into an adapted version for Envir3D in part C. During this translation phase, the presentation elements are adapted to elements belonging to the widget set of Envir3D. This operation mostly consists of the addition of absolute positions in the three dimensions as the tool needs the position of elements to generate the UI. A local reference (*location*) had to be added to each element to indicate to the tool where he could find components. This transformed presentation model is then used as input by Envir3D to produce a UI expressed in VRML in the part D of figure 5-9.

Chapter 5 Reverse Engineering of Web Sites: Vaquita

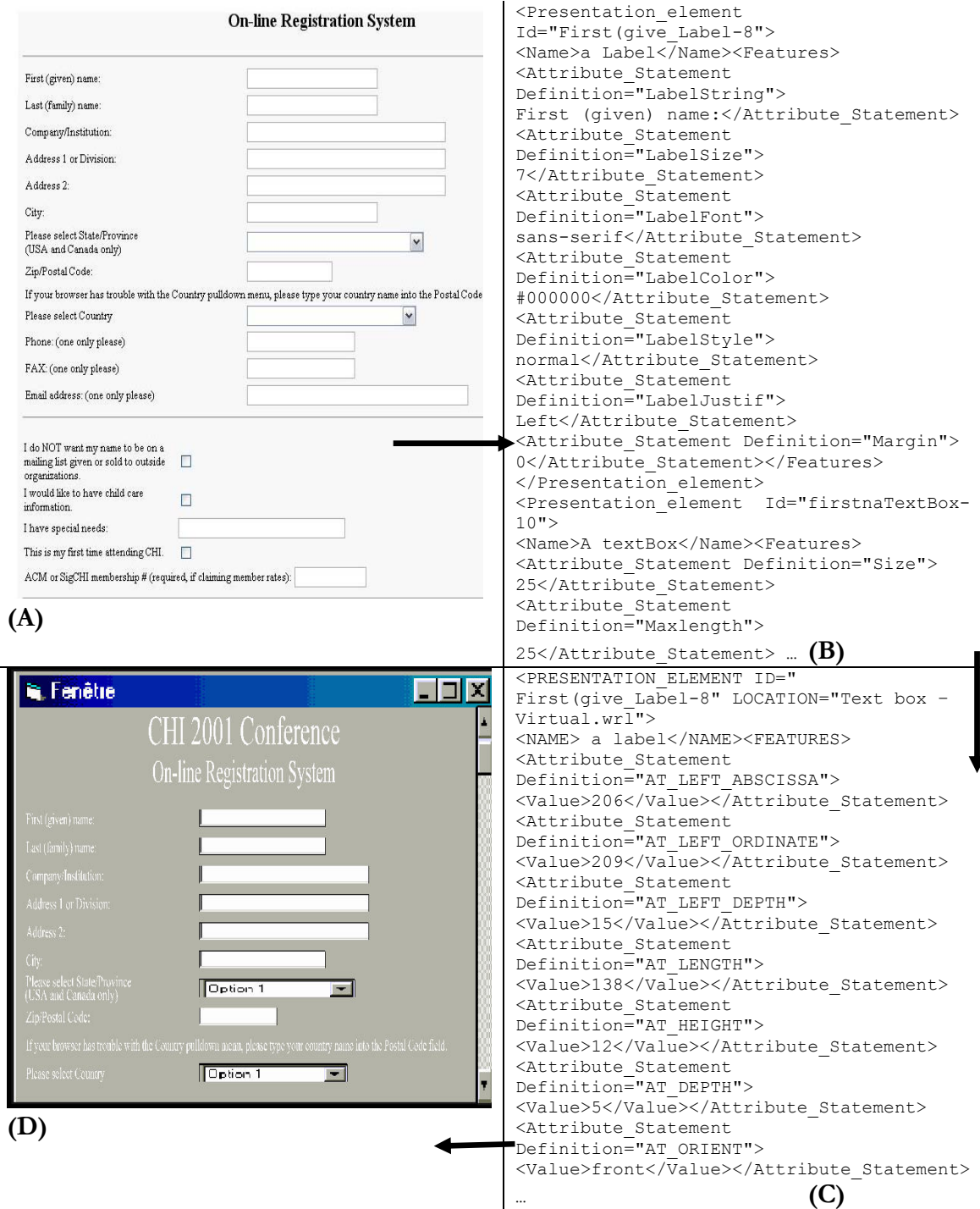


Figure 5-9 The reengineering of a Web Form

5.5 Shortcomings of Vaquita

This first tool suffers from several shortcomings:

1. The output language is XIML, which is a proprietary language of Redwhale Software, a private company. Therefore, users need to sign a licence in order to use this language and it is not possible either to distribute software that use this technology freely without their agreement. This licence will also reduce the spreading of the language, and thus the utility and usage of the tool. In addition, the meta-model of the XIML language (Figure 5-1) also demonstrates that not all aspects related to context of use could be easily modelled. Therefore, the language also suffers from some expressivity shortcomings that we need to overcome.
2. The tool is only able to reverse engineer the presentation of a UI (the composition of the UI, expressed in terms of concrete user interface objects), not the dialog which is the second mandatory model for the regeneration of user interfaces. In XIML, the dialog model is a separate model which is not very precisely defined.
3. Vaquita can only reverse engineer files statically, i.e., at design time. The target Web page has to be saved on the local machine before it is analyzed. This process prevents any dynamic usage where a batch or an on-line system could be used.
4. It is impossible to have a direct access to the Web by giving an URL to reverse engineer. The tool is only able to reverse Web pages one by one which can be a very cumbersome process for the transformation of large Web sites.
5. Finally, Vaquita is not able to recognize the different relations between the elements of the UI description automatically (layout and dialog).

5.6 Conclusion

The subproblems, as defined in section 4.6, of the reverse engineering process covered by this approach are the elements detection, the attributes recuperation, the hierarchy detection task and the retargeting operations. Thus four out of seven requirements are addressed in this first study. Although Vaquita represents already a first significant experience in handling the problem of UI reverse engineering, several limitations have been identified in order to leverage the capabilities in a more sophisticated tool in the next chapter. However, the Vaquita tool developed in this chapter also showed some promises in coupling it with a forward engineering tool (here, a 3D generator for a virtual reality interface).

Chapter 6 ReversiXML

This chapter is the continuation of the last chapter about the reverse engineering of HTML Web pages as it is aimed at addressing the observed limitations of the Vaquita tool, a first attempt to solve the UI reverse engineering in a declarative language. Therefore, it has not the same structure than chapter 5, 7 and 8 as some parts of the problem have already been identified in the previous chapter and as new sub-problems will be handled. This sixth chapter starts by summing up the results of the evaluation of the first approach and tool, leading to the definition of new requirements. The second section redefines the working hypotheses of previous chapters. The third section contains the description of the UsiXML language and some examples of derivations rules. The fourth section describes the tool ReversiXML, its process and its implementation based on the aforementioned derivation rules.

6.1 Evaluation of Vaquita

This first study gave good results, but had to be enhanced to overcome several limitations and to adapt the tool to the needs of the industrial partners in the Cameleon project (a research project aiming to develop context aware UIs, <http://giove.isti.cnr.it/cameleon.html>). Several new requirements have been identified, and the most relevant are listed below:

- First, the user interface description language used in Vaquita is XIML, and as this language is the property of a private company, a new language has been developed within the Cameleon project: UsiXML. The new tool should thus produce UI specifications expressed in this new description language.

- Secondly, the new tool should support on-the-fly reengineering. Instead of statically reverse engineer Web pages, the process of reengineering HTML pages could be done dynamically at the server level. This solution implies the change of the programming language, as Vaquita was written in Visual Basic 6. After analysis, the best language to fit with this requirement is PHP, because:

- The language is free.
- It has numerous advantages to develop online applications connected to databases (e.g. with mySQL).
- The fifth version of the language contains most of the functionalities required by our tool: it is compiled by including a DOM library (libxml2.dll) and tidy (tidy.dll) and facilitates thus the implementation by permitting the invocation of tree-manipulation function built-in the language.
- PHP allows to access other Web pages (to reverse engineer) very easily, as it runs on a server
- Performances of large applications are very good (e.g. more than 20 times faster than Vaquita, written in Visual Basic 6)

- The structure of the new tool has to be completely reviewed in order to adapt the software to this new architecture (see the new architecture evolution in appendix I).

- The new tool should support an automatic selection of the retargeting rules, by recognition of the user's platform.

- The tool should also support reverse engineering up to the abstract UI level from the reference framework [Calv02, Calv03], thus giving the possibility to the user to reverse engineer HTML code at two different levels.

There are more details about the weaknesses of Vaquita compared with the new tool in section 6.4.4.

6.2 Working hypotheses

The working hypotheses have been modified compared to the ones of the previous chapter. The aim here is to recover an abstract specification (at two different levels of abstraction) of the different widgets composing an UI, while taking into account some parts of the dialog (navigation between pages). The reverse engineering of imperative languages is still not covered in this analysis, as it would require a completely different type of reverse engineering, and therefore only pure HTML is reverse engineered (not javascripts, applets...). In this second

research, the layout of the interaction objects will also be recovered while keeping the flexible and retargeting features of the previous reverse engineering approach. The output language for this second research is UsiXML. As this language is still evolving, the version of the language which was selected for this thesis was the version 1.4.6, which was the most up-to-date version of the language at the time of implementation. A second reason motivating why the UsiXML language has been chosen after XIML is that the former language is also compatible with the reference framework introduced in chapter 2, whereas the latter is not.

Another important aspect of this approach is to allow the dynamic application of the process, and thus if a forward dynamic approach is combined, it would allow an on-the-fly reengineering. The reverse engineering method as applied in this chapter is thus a flexible reverse engineering of HTML Web pages at runtime in order to recover the concrete or abstract UI specification expressed in UsiXML.

The fact that this approach is only dedicated to HTML implies that some interaction objects and dialog components are lost, as javascripts are ignored (they are mostly used to apply dynamic effects on Web pages). Solutions for this problem are proposed in the section 6.4.5. By ignoring cascading styles sheets files (.CSS), there is also a loss of styles and presentation aspects of the original Web pages. See the chapter 9 about validation for more details on losses of the reverse engineering process with this approach.

6.3 UsiXML meta-model and derivations rules

6.3.1 UsiXML meta-model and definitions

The source language for this analysis has already been presented in the previous chapter, in section 5.3. The output, the UsiXML language v1.4.6 [Limb04a, Limb04b, Limb04, Luyt04], is an UI specification language developed at UCL. UsiXML (USer Interface eXtensible Markup Language – <http://www.usixml.org>) allows specifying eight different models (see figure 6-1) composing a ninth model to represent the various UI aspects: a task model, a domain model, an AUI model, a CUI model, a mapping model, a context model, a resource model and a transformation model.

Task Model: is a model describing the interactive task as viewed by the end user interacting with the system. A task model represents a decomposition of tasks into

sub-tasks linked with task relationships. Therefore, the decomposition relationship is the privileged relationship to express this hierarchy, while temporal relationships express the temporal constraints between sub-tasks of a same parent task.

Domain Model: is a description of the classes of objects manipulated by a user while interacting with a system. The task and domain models compose the higher level of abstraction of the cameleon reference framework.

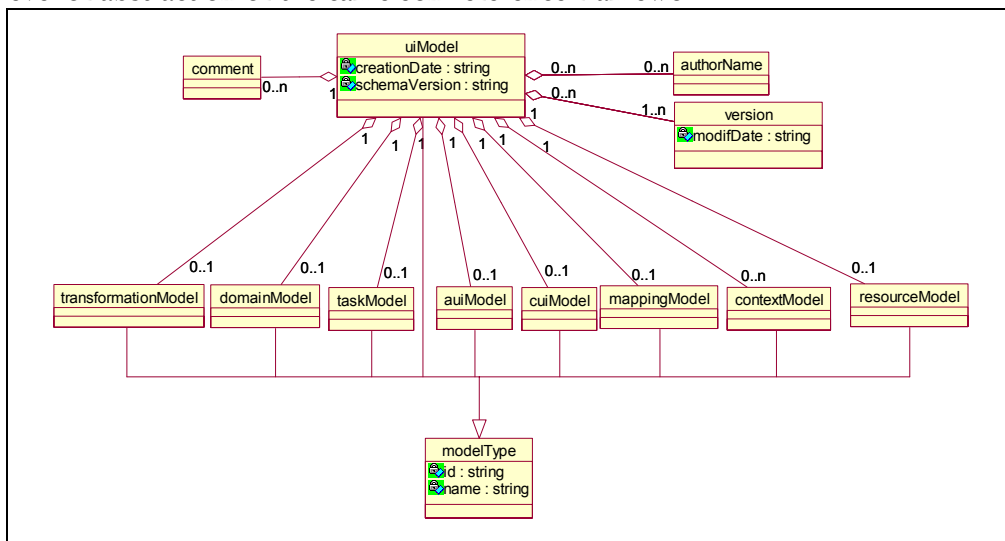


Figure 6-1 Model composition in UsiXML

Mapping Model: is a model containing a series of related mappings between models or elements of models. A mapping model gathers a set of inter-model relationships that are semantically related.

Context Model: is a model describing the three aspects of a context of use in which an end user is carrying out an interactive task with a specific computing platform in a given surrounding environment. A context model consists of a user model, a platform model, and an environment model.

The *user model* is composed of several user stereotypes, described by attributes such as the experience with the system or with the task, the motivation, etc.

The *environment model* describes any property of interest of the global environment where the interaction takes place. The properties may be physical (e.g., lighting or noise conditions) or psychological (e.g., level of stress).

The *platform model* captures relevant attributes related the combination of hardware and software where the user interface is intended to be deployed.

uiModel: is the topmost superclass containing common features shared by all component models of a UI. A uiModel may consist of a list of component models in any order and any number, such as task model, a domain model, an abstract UI model, a concrete UI model, mapping model and context model. A uiModel needs not to include one of each model component. Moreover, there may be more than one of a particular kind of model component in a uiModel.

Abstract User Interface (AUI) model is a canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent of the interaction modality (e.g., graphical, vocal or tactile). The elements used in the logical UI are abstractions of existing widgets.

Concrete User Interface (CUI) model: concretizes an AUI into Concrete Interaction Objects so as to define widgets layout and interface navigation. This interface is now composed of existing UI widgets, but the widgets are independent of any particular toolkit.

Resource Model contains elements (title, tooltip, mnemonic...) specific to a given context (for example, the user's language). Resources are linked to objects of the CUI or AUI model.

Transformation Model permits to specify transformation rules under the form of graph transformation rules, taking advantage of the underlying graph structure of UsiXML. This formalism supports different types of transformations: abstraction (e.g. recovering an AUI model starting from a CUI model), reification (e.g. generating a CUI from a task model and a domain model) and translation (e.g. adapting a CUI for another context of use).

Only two models are used in this chapter, the concrete UI and abstract UI models, and their meta-models are given in appendix A. These models have the common property that they allow specifying the composition (thanks to individual components) and structure (thanks to containers) of a UI, and some dialog or layout relations between elements composing the UI.

The *CUI level* is composed of interaction objects dependent of the modality (graphical or vocal). Elements composing the CUI model are the representation of the most common widgets that users can find in UIs. As this step is the nearest to the final code, elements possess several properties defining their appearance, size, layout etc...

The *AUI level* contains a description independent of the modality, the UI description is based on classes of interaction objects (input, output, navigation, activation, containers) with very few attributes/properties.

6.3.2 Derivation rules

Some example of derivation rules from the final UI code into abstract or concrete UI are shown. The complete list of derivation rules for the concrete UI can be found in appendix C. The first set contains an example of a transformation rule for labels (into outputs), at the AUI level. The rule includes 33 textnode-containers elements in its left parts, and transforms them into two different nodes, an `abstractIndividualComponent` which is the parent of another node, the `output` node, that represent the output facet of the component.

G56 – textnode containers (output facet)

```

 $\forall x \in T_H : x = (\text{p}\text{pre}\text{sub}\text{sup}\text{q}\text{blockquote}\text{b}\text{i}\text{strong}\text{em}\text{strike}\text{s}\text{h1}\text{h2}\text{h3}\text{h4}\text{h5}$ 
 $\text{h6}\text{u}\text{dd}\text{dt}\text{li}\text{tt}\text{address}\text{u}\text{cite}\text{code}\text{dfn}\text{kbd}\text{samp}\text{var}\text{caption}\text{address}) \wedge$ 
 $x.\text{textnode} \neq \text{NULL}$ 
 $\rightarrow \text{Addnode}(\text{"abstractIndividualComponent"}, \text{idaio})$  where  $\text{idaio} = \text{NodeAmount}(T_t)$ 
 $\wedge \text{AddAttribute}(\text{idaio}, \text{"id"}, \text{idaio})$ 
 $\wedge \text{AddAttribute}(\text{idaio}, \text{"name"}, \text{idaio})$ 
 $\wedge \text{Addnode}(\text{idout}, \text{"output"})$  where  $\text{idout} = \text{NodeAmount}(T_t)$ 
 $\wedge \text{AddAttribute}(\text{idout}, \text{"outputContent"}, x.\text{textnode})$ 
 $\wedge \text{AddAttribute}(\text{idout}, \text{"id"}, \text{idout})$ 
 $\wedge \text{AddAttribute}(\text{idout}, \text{"name"}, \text{idout})$ 
 $\wedge \text{AddArc}(\text{idaio}, \text{idout})$ 

```

As stated before, this level is a general description of the UI without any reference to the (graphical) modality, there are very few attributes for the elements (to compare, `textComponents` at the CUI level possesses more than 10 attributes). Only the content of the interaction objects are recorded at this level, without their style attributes and therefore a header is recorded in the same type of element as a list item for example.

For the CUI level, the first example of derivation rule represents the creation of a `window` when a `body` node is detected. The `name` attribute of the `window` takes the `textnode`'s value of the `title` node if it exists; otherwise, its value is the number of nodes in the target tree. The `isEnabled` and `isVisible` (two mandatory attributes of the CUI model) are automatically set to `true`. The rest of the attributes are translated into their corresponding UsiXML attributes. A particular attribute - `filename`- that does not belong to the CUI, is added to record the HTML filename in the specification. It will be used later for frames composition (see set of rules G 47, 48 and 49) and removed in the cleaning of the target tree operations (set of rules G50). After the creation of a `window` node in the UsiXML tree, two `box`

elements are added. The first **box** is of the type vertical as it will contain a vertical sequence of horizontal **boxes** (see layout recovery section 6.4.2.b for more details) and the second **box** is the type horizontal as it will contain the first “row” of elements.

G1a – body (window)

$\forall x \in T_H : x = \text{body} \rightarrow \text{Addnode}(\text{“window”}, \text{idwin})$ where $\text{idwin} = \text{NodeAmount}(T_i)$
 $\forall x \in T_H : x = \text{body} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“id”}, \text{idwin})$
 $\forall x \in T_H : x = \text{body} \wedge \text{title.textnode} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“name”}, \text{title.textnode})$
 $\forall x \in T_H : x = \text{body} \wedge \text{title.textnode} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“name”}, \text{idwin})$
 $\forall x \in T_H : x = \text{body} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“isEnabled”}, \text{“true”})$
 $\forall x \in T_H : x = \text{body} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“isVisible”}, \text{“true”})$
 $\forall x \in T_H : x = \text{body} \wedge x.\text{bgcolor} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“bgColor”}, x.\text{bgcolor})$
 $\forall x \in T_H : x = \text{body} \wedge x.\text{topmargin} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“windowTopMargin”}, x.\text{topmargin})$
 $\forall x \in T_H : x = \text{body} \wedge x.\text{leftmargin} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“windowLeftMargin”}, x.\text{leftmargin})$
 $\forall x \in T_H : x = \text{body} \wedge x.\text{background} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“bgImage”}, x.\text{background})$
 $\forall x \in T_H : x = \text{body} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“filename”}, \text{filename})$
 $\forall x \in T_H : x = \text{body} \rightarrow \text{ConstrBox}(\text{“box”}, \text{“vertical”}, \text{idbox})$ where $\text{idbox} = \sum \text{node} \in T_i \wedge \text{AddArc}(\text{idwin}, \text{idbox})$
 $\forall x \in T_H : x = \text{body} \rightarrow \text{ConstrBox}(\text{“box”}, \text{“horizontal”}, \text{idhbox})$ where $\text{idhbox} = \text{NodeAmount}(T_i) \wedge \text{AddArc}(\text{idwin}, \text{idhbox})$

The second example is the transformation of **img** nodes into **imageComponents**. The URL of the source file of the image is copied in the **defaultContent** attribute in the target tree. The path of the **img** node in the HTML tree is checked in order to find an anchor (**<a>**) node. If it is the case, the image can be used as a link to navigate through the Website, and therefore the **href** attribute of the **<a>** node is copied in the **hyperLinkTarget** attribute in the UsiXML tree. Finally, the **checkAligment** function is triggered, which is a set of functions common for almost every element of the source tree. This special set of rules (see chapter 4) checks if an element is in the same path as an alignment modifier (**div**, **center**, **align** attributes), and if it is the case, an attribute is added in the UsiXML specifications to reflect the horizontal position of the element.

G15 – img (imageComponent)

$\forall x \in T_H : x = \text{img} \rightarrow \text{Addnode}(\text{ImageComponent}, \text{idimage})$ where $\text{idimage} = \text{NodeAmount}(T_i)$
 $\forall x \in T_H : x = \text{img} \wedge x.\text{height} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{“imageHeight”}, x.\text{height})$
 $\forall x \in T_H : x = \text{img} \wedge x.\text{width} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{“imageWidth”}, x.\text{width})$

$\forall x \in T_H : x = \text{img} \wedge \text{IsInPath}(a) \rightarrow \text{AddAttribute}(\text{idimage}, \text{"hyperLinkTarget"}, \text{NearestInPath}(x,a).\text{href}) \wedge \text{AddGraphTr}(\text{idimage}, \text{NearestInPath}(x,a).\text{href})$ $\forall x \in T_H : x = \text{img} \wedge x.\text{border} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"borderWidth"}, x.\text{border})$ $\forall x \in T_H : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"id"}, \text{idimage})$ $\forall x \in T_H : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"name"}, \text{idimage})$ $\forall x \in T_H : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"isVisible"}, \text{"true"})$ $\forall x \in T_H : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"isEnabled"}, \text{"true"})$ $\forall x \in T_H : x = \text{img} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"defaultContent"}, x.\text{src})$ $\forall x \in T_H : x = \text{img} \wedge x.\text{hspace} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageHorizSpace"}, x.\text{hspace})$ $\forall x \in T_H : x = \text{img} \wedge x.\text{vspace} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageVertSpace"}, x.\text{vspace})$ $\forall x \in T_H : x = \text{img} \rightarrow \text{CheckAlignment}(x, \text{idimage})$

The last example of derivation rules is about **table** detection. Two cases can occur, either the **table** has no **border** attribute (or the **border** attribute is set to zero pixel) and it is transformed into a vertical **box**, as it will contain a vertical sequence of rows, or the **border** attribute is greater than zero and it is transformed into a UsiXML **table**. Several common attributes can be derived for the two types of elements. However, the **table** possesses three specific attributes, the **border's width**, and attributes defining the size of the **table** (rows and columns). These attributes are computed: the number of rows – **ySize** - is computed by counting the number of rows (**tr** nodes) that are “direct-child” of the **table**, as several tables can be embedded. The number of columns – **xSize**- is computed by finding the maximum number of **td** nodes that are child of a **tr** node which has itself the current **table** as parent. A maximum value is searched because the number of columns in a **table** does not have to be constant in HTML. Finally, the alignment of the **table** is checked thanks to the group of functions **CheckAlignment** (see previous explanations of G15).

G2 – table (table-box)

$\forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0 \rightarrow \text{Addnode}(\text{"table"}, \text{idtable}) \text{ where } \text{idtable} = \text{NodeAmount}(T_t)$ $\forall x \in T_H : x = \text{table} \wedge (x.\text{border} = 0 \vee x.\text{border} = \text{NULL})$ $\quad \rightarrow \text{Addnode}(\text{"box"}, \text{idtable}) \text{ where } \text{idbox} = \text{NodeAmount}(T_t)$ $\quad \rightarrow \text{AddAttribute}(\text{idtable}, \text{type}, \text{"vertical"})$ $\forall x \in T_H : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"isEnabled"}, \text{"true"})$ $\forall x \in T_H : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"isVisible"}, \text{"true"})$ $\forall x \in T_H : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"name"}, \text{idtable})$ $\forall x \in T_H : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"id"}, \text{idtable})$ $\forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0 \rightarrow \text{AddAttribute}(\text{idtable}, \text{"xSize"}, (\max_{tr}(\sum_{td} \text{sibling}(td) \mid td \in T_s) \wedge \text{NearestInPath}(\text{table}, td) = x))$

$$\begin{aligned} \forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0 &\rightarrow \text{AddAttribute}(\text{idtable}, \text{"ySize"}, (\sum_{tr} \text{sibling}(tr) \mid tr \in T_s \wedge \\ &\quad \text{NearestInPath}(\text{table}, tr) = x)) \\ \forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0 &\rightarrow \text{AddAttribute}(\text{idtable}, \text{"borderwidth"}, x.\text{border}) \\ \forall x \in T_H : x = \text{table} \wedge x.\text{width} \neq \text{NULL} &\rightarrow \text{AddAttribute}(\text{idtable}, \text{"width"}, x.\text{width}) \\ \forall x \in T_H : x = \text{table} \wedge x.\text{height} \neq \text{NULL} &\rightarrow \text{AddAttribute}(\text{idtable}, \text{"height"}, x.\text{height}) \\ \forall x \in T_H : x = \text{table} \wedge x.\text{bgimage} = \text{NULL} &\rightarrow \text{AddAttribute}(\text{idtable}, \text{"bgimage"}, x.\text{background}) \\ \forall x \in T_H : x = \text{table} \wedge x.\text{bgcolor} \neq \text{NULL} &\rightarrow \text{AddAttribute}(\text{idtable}, \text{"bgcolor"}, x.\text{bgcolor}) \\ \forall x \in T_H : x = \text{table} &\rightarrow \text{CheckAlignment}(x, \text{idtable}) \end{aligned}$$

6.4 Tool support: ReversiXML

This section describes the tool in details: the first subsection makes a general description of the tool, the second subsection dissects the reverse engineering process while pointing out some important steps (dialog and layout), the third section compares the tool with Vaquita according to several parameters and the last section describes ongoing issues and future works. The evolutionary architecture of the tool is illustrated in appendix I, and examples of the outputs of the tool can be found in case studies of chapter 9.

6.4.1 Description of ReversiXML

ReversiXML is the evolution of the first tool Vaquita. ReversiXML (accessible at <http://www.isys.ucl.ac.be/bchi/research/reversi/RevXMLUI.php>) is the on-line version of this first tool. It has been developed in PHP 5 and is decomposed into a set of libraries: one for the UI of the tool, a group of libraries for the reverse engineering engine itself and another one for the detection of connected platforms. The new tool has been developed to be used in two different ways: either manually by a designer or automatically by an external user. In the first case, the designer can define the reverse engineering options and choose an output file which will be sent back to his computer. In the second case, the user cannot access the reverse engineering options and he can not see the results of reverse engineering either. The results are directly sent to a forward engineering tool. This second use of the tool is based on a knowledge base of configuration files (reverse engineering options) for different types of platforms. After the detection of the connected user and platform, the best configuration file is selected following these parameters and the reverse engineering is applied according to these options. The interest of this second use is that the tool could be exploited in an on-the-fly reengineering, by using only ReversiXML's engine libraries and thus the current approach in a batch process.

Chapter 6 ReversiXML

The remainder of this section is devoted to an illustrated description of the manual version. In the manual version of the tool, three files can be chosen at the starting screen of ReversiXML (see fig. 6-2): the input file, a reverse engineering configuration file and an output file. The input file can be a local file or an URL. The two other types of files are not mandatory: one can apply a reverse engineering without saving its results and a reverse engineering can be achieved without any configuration file (in this case all the default rules are applied).



Figure 6-2 Start screen of ReversiXML

Configuration files are created on the reverse engineering options page (fig.6-3). As in Vaquita, the user can fine tune the detection of each HTML object or attribute. Each tabbed panel of figure 6-3 corresponds to an HTML element or group of elements that the user can choose to skip or to record in the abstract specification. For each element, the designer can choose to ignore its attributes by unchecking its corresponding check box. Reverse engineering transformations can also be chosen on this page (e.g. transform a radio button group into a drop down list box, consider a sequence of links as a menu, etc.). It is also possible to choose the abstraction level the user wants to reach on this page (abstract or concrete UI specification). Finally, the user can save the file on the server for future usage. After the reverse engineering in the manual version, the result of the reverse engineering- a concrete or an abstract UI model- is then displayed in the browser and saved in a file on the local server, in the UsiXML format (see figure 6-4).

Chapter 6 ReversiXML

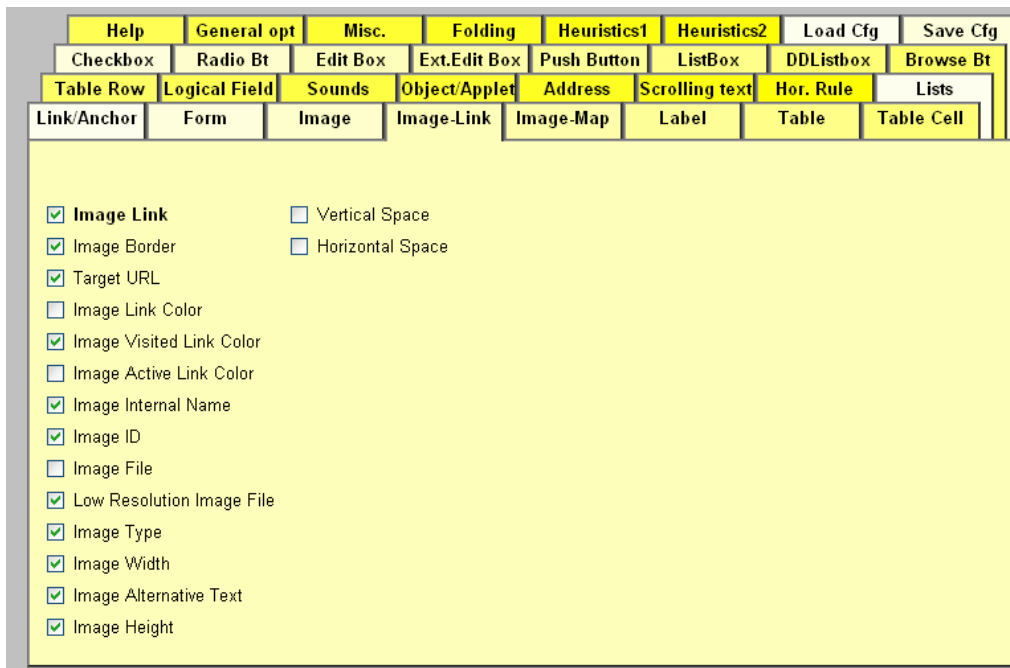


Figure 6-3 Reverse Engineering options

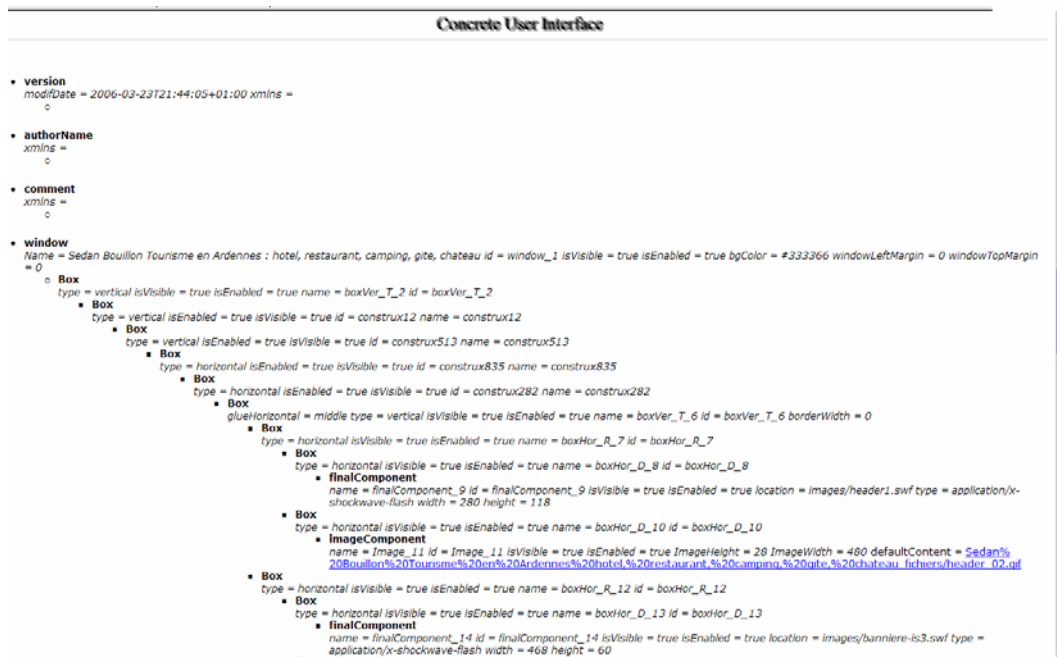


Figure 6-4 The resulting concrete UI model

The three files defined on the previous page are still accessible on the result page, under the form of a link. The user can save the output on his computer by right-clicking on the output-file link. Some other information is also displayed on this results page: some data about the connected platform (ip, operating system, browser type and version, javascripts enabled browser, color-capable platform...), the error buffer of tidy and the list of links of the current Web page. Several statistics are displayed, such as the number of objects of each type, the number of removed elements, etc...

The location of ReversiXML in the Cameleon reference framework is shown on figure 6-5. The tool is able to recover the concrete or abstract UI model starting from a HTML file and to perform some translation operations for another context of use.

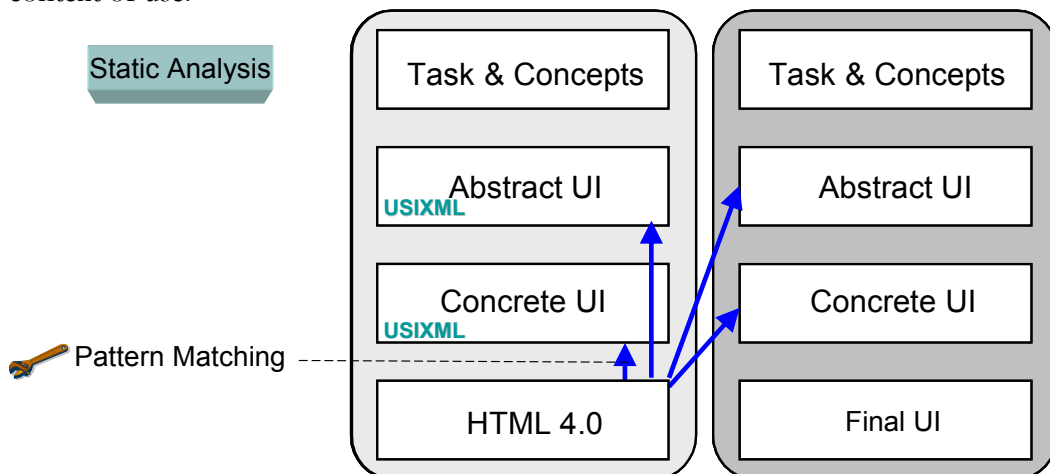


Figure 6-5 ReversiXML in the reference framework

6.4.2 The reverse engineering process

This subsection is divided into three parts: firstly, the reverse engineering is presented in details by showing the different steps of the process. The two next parts are dedicated to two specific steps of the first section, the layout and the dialog recovery, in order to emphasize these two important dimensions of the reverse engineering.

6.4.2.a Steps

The reverse engineering process can be decomposed into 10 steps (see figure 6-6), which are explained below. Not all steps have been completely implemented, but the process and tool structure has been developed so as to accommodate the addition of these modules. The input of ReversiXML is a set of URLs to reverse engineer, called the page pool, and a configuration file.

1. Web Page Extraction

In this first step, four cases can occur: the target file is located on distant server, the file cannot be found, the page has already been analyzed or the file is located on the same server. In the first two cases, ReversiXML saves the Web page from a distant server or from archives.org (a site archiving ancient Websites) on the local server on which ReversiXML runs in order to facilitate the manipulation and to speed up other requests for the same page.

In the last two cases, either the already processed model is sent back to the user/forward engineering tool, or the local page is directly analyzed on the server. The output of this step is a string containing the HTML code of the Web page.

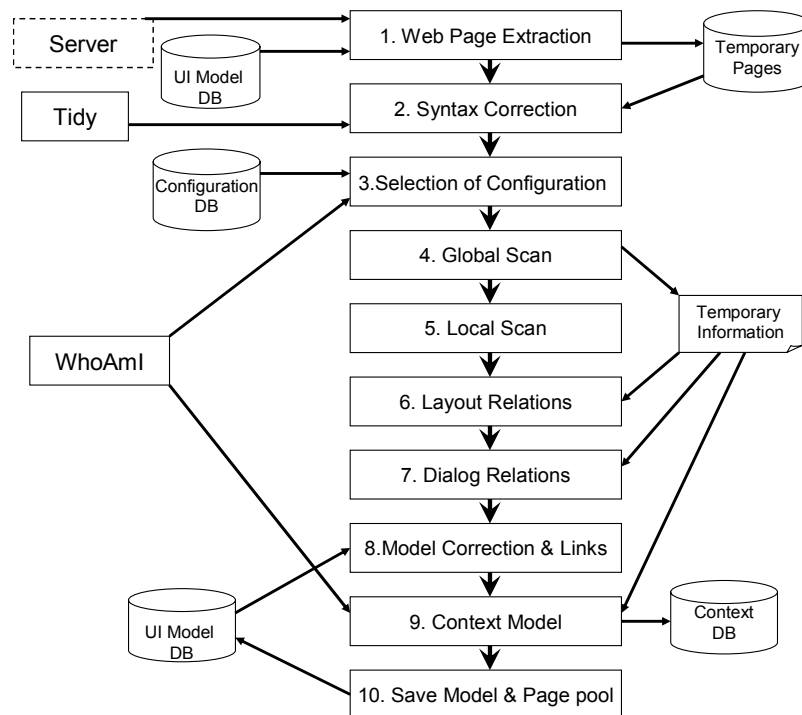


Figure 6-6 The entire reverse engineering process

2. Syntax Correction

The validity of the Web page is then checked. If the Web page does not respect the XML syntax, the HTML code is corrected by tidy [Tidy] in order to obtain a file that can be put in a DOM structure [Dom1]. A correct XML syntax implies that every single tag has to be closed, i.e. a `
` tag has to be rewritten as a `
` tag, and overlapping tags, such as `<a>hello <i>world</i>`, have to be written in the correct order (as in this format: `<a>hello<i>world</i>`). Moreover, Tidy allows to remove empty paragraphs and rename tags with the same meaning (`<i>` and `<emph>` for example). This allows to have the same code format and so to speed up the process by removing useless or redundant tags. The output of this step is a string containing the HTML code reformatted in the XML syntax.

3. Selection of the Configuration file

The input of this step is a configuration file. If no configuration file was chosen, the process continues on step 4. If ReversiXML is used in the automatic mode, a file is chosen by the system. Thanks to the information from the context model and from the WhoAmI recognition system, the best configuration file is selected and loaded by ReversiXML. The WhoAmI recognition system is a set of PHP scripts partially developed in collaboration with IS3 (<http://www.is3.com>) allowing to identify some features of the user and platform connected to a Website. The result of this step is the selection of a set of techniques and reverse engineering options.

4. Global scan of code - Identification of elements

The corrected string containing the HTML file is put in a DOM structure which is analyzed by the tool. In a first step, the code is scanned globally to detect the elements that can be translated, following the options of the configuration file, directly in UsiXML. This first step gives only a partial model that will be completed in steps 4, 5 and 6. During this phase, some hidden attributes that will be used in other steps are defined, such as the vertical and horizontal position, sources and targets for every link on the page or meta-information that can be used to complete the context model. The output is a partial concrete or abstract UI model and a set of “hidden” attributes.

5. Local scan of code – Identification of special elements and attributes

To complete the UsiXML model, some small parts of the page have to be scanned a second time with a particular aim. For example, **tables** at the concrete level have the **numberOfRows** and **numberOfCols** attribute which is computed by counting the number of rows and by finding the maximum value for the number of cells in a same row. This value cannot be added directly in the first scan, as a **table** can contain others, and it is possible to find a cell belonging to the first table in the last line of the HTML code. There are several other cases where local scans are needed, such as image maps, radio buttons etc... because the code relative to these elements is scattered in the input file. The output is an abstract or concrete UI model almost completed.

6. Specification of the UI Layout

In this step, the layout can be computed as all the different objects of the page have been detected and abstracted. See section 6.4.2.b for more details about the deduction of the layout. The output is a complete presentation and dialog model (following our hypotheses) at the abstract or concrete UI level.

7. Specification of the Dialog Relations

After having completed the presentational part of the UI model, the dialog transitions are completed by ReversiXML. Thanks to the information (the link table – see figure 6-10) gathered during the step 4, the relations are added at the end of the model. Only the navigational part of the dialog model is completed by ReversiXML. The output of this step is complete (but maybe containing incorrect results) abstract or concrete UI.

8. Model Correction and Model Links

Some corrections to the produced model are applied during step 8. For example some attributes that do not belong to the CUI model but that were used to help the construction of the output file have to be erased. The step 6 may generate too many **boxes** as the first goal of the algorithm is to represent exhaustively the layout thanks to **boxes** with the maximal accuracy. This can be corrected by identifying useless **boxes**. These **boxes** correspond mainly to lines or columns of a HTML **table** which do not always give useful information for the UsiXML structure. The layout can thus be represented with fewer boxes without loss of accuracy. Empty **boxes** (which correspond to empty cells) can also be removed in the UsiXML specification. But as this method removes about 20% of the useless

boxes, it should be enhanced as the resulting models remain with several useless **boxes** (see more details in appendix J). Following the configuration, some other transformations can be applied during this step, such as for example the transformation of a sequence of links into a menu or the division of the **window** into smaller **windows** for platform with small display capabilities. The concrete or abstract model is then linked with other models created with ReversiXML. The link table (figure 6-10) is scanned to check whether a target has already been reverse engineered. If it is the case, the name of the target of the dialog transition is changed into the id of the target window. This step completes the generation of a correct abstract/concrete UI model, which is now linked with the other models generated during this session.

9. Generation of the Context Model

In this step, the information from the Web page and from the WhoAmI system is used to specify some parts of the context model. This step has not been completely implemented, but a method is given in appendix D to derive some information needed for the context model. The output of this step is partial context model, as all the needed information to complete the context model cannot be found with the current recognition system.

10. Save Results and Check Page Pool

In this final step, the generated model is saved in the UI model knowledge base and the page pool (list of pages to be reverse engineered) is checked to see if some pages are still to be analyzed. If the list is not empty, the process starts again at step 1.

These steps allow to cover the seven sub-problems of the reverse engineering process presented in section 4.6, as the steps 4 and 5 correspond to the elements, attributes and hierarchy recovery tasks, step 6 is related to the layout recuperation, step 4, 7 and 10 fulfill the dialog detection task, step 8 is applied to achieve the multiple specification transformation tasks, step 3, 4 and 5 are necessary to realize retargeting operations.

6.4.2.b Layout recovery

The layout in UsiXML can be expressed in three ways: either by putting elements into boxes (as in XUL or in a grid layout in Java), or by specifying a **glue** attribute that defines the horizontal position of an element compared to its container, or by

defining a relation between two objects. The **box**-layout combined with **glue** attributes have been chosen in ReversiXML to represent the layout of the interaction objects as the automatic generation of relations could let ambiguities in the position of elements.

The **box** mechanism works in this way: when a vertical **box** is defined, the elements contained in the **box** are put below each other. Elements embedded in horizontal **boxes** are put after – on the right – each other. **Glue** attributes are used to define the position of an element compared to its container (centered, right-justified etc...).

In standard HTML it is impossible to put elements on the page precisely and with certainty without **tables**. Usually, elements are positioned thanks to **tables**, and the layout is thus automatically recovered by applying “normal” derivation rules (G3, G4 and G5 in appendix C). But in some cases, other types of elements are used to structure the UI, and in this case, “artificial” **boxes** have to be added in order to keep the original position of the objects.

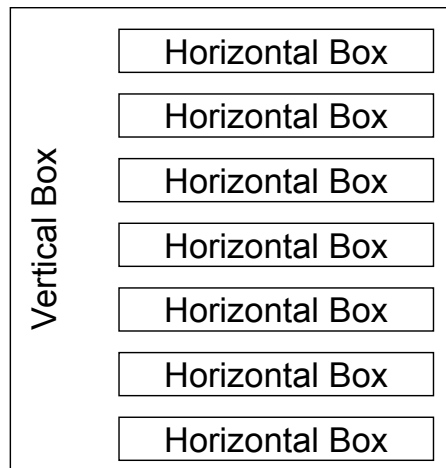


Figure 6-7 General layout of a Web page

The general idea for the recovering of the layout is that an HTML UI is composed of several rows (horizontal **boxes**) contained in a vertical **box** (see figure 6-7). Therefore, a first vertical **box** is always created at the beginning of the analysis, and a horizontal **box** which will contain the first line of elements.

Then, derivations rules are applied, but when a specific element is encountered, (**br,li,ul,dd,dl,ol,h1...h6,center,q,blockquote,pre,hr,div**), a marker is specified in the UsiXML specification. These elements imply the generation of line breaks in the

UI, and thus elements before and after the line break should be displayed on different lines.

In a second phase after the analysis of the complete HTML file, the UsiXML specification is scanned and for each line break marker, a new vertical box is appended to the parent of the marker. Then, all the elements before the marker are put in a new horizontal box and all the elements after the line break (all until another box or line break is encountered or the parent element is closed) are also put into another new horizontal box. In this manner it is possible to represent the fact that the elements should be displayed on different lines. Figure 6-8 represents the transformation of a UsiXML specification containing a line break marker.

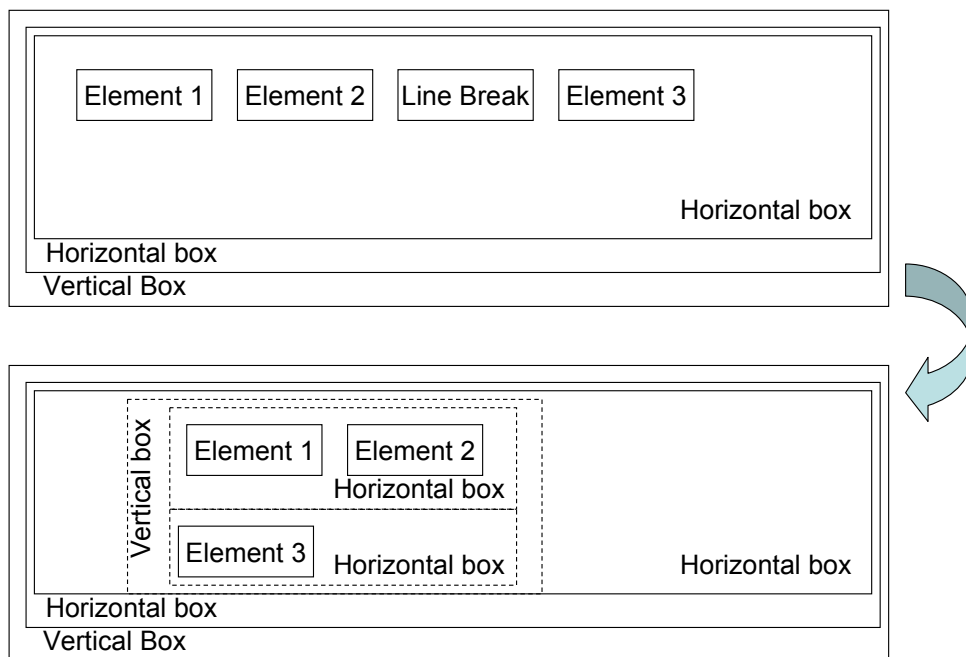


Figure 6-8 Layout recovery

The other manner to position elements in UsiXML is to define `glueHorizontal` and `glueVertical` attributes. Therefore, the alignment of each element is checked and these attributes are added consequently. The alignment of elements is modified when they are embedded in tags such as `<div align=... valign=...>`, `<center>` or `<p align=... valign=...>` or when they possess their own `align` attribute in the HTML code (such as in `<table align=...>`). For these elements, a `glue` attribute is added in UsiXML and it takes the value of the `align` (or `valign`) attribute.

A small example is presented of figure 6-9 which has the same structure as on figure 6-7. The HTML code and its corresponding simplified UsiXML code are shown below.

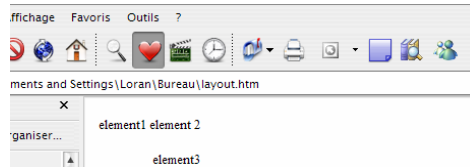


Figure 6-9 Example of layout recovery

HTML code

```
<table>
  <tr>
    <td>
      element1 element 2 <br><p align=right>element3
    </td>
  </tr>
</table>
```

Simplified UsiXML code

```
<box type="vertical">
  <box type="horizontal">
    <box type="horizontal">
      <box type="vertical">
        <box type="horizontal">
          <box type="horizontal">
            <textOutput defautcontent="element 1">
            <textOutput defautcontent="element 2">
          </box>
          <box type="horizontal">
            <textOutput defautcontent="element 3">
            <glueHorizontal="right">
          </box>
        </box>
      </box>
    </box>
  </box>
</box>
```

6.4.2.c Dialog relations

The navigation between the different windows or containers is the central point of the representation of the dialog for the HTML language.

As shown on figure 6-10, the different links contained in a UI are gathered in a table, and are added to the model at the end of the reverse engineering process.

Before the addition to the model, each link's target is verified, to check if it has already been (or will be) reverse engineered during the same session. If it is the case, the URL is removed and replaced by the id of the target window and the former URL is put in a comment.

An example of a graphical transition (dialog relation) is shown below:

```
<graphicalTransition id="gt_29" transitiontype="open">
  <source sourceId="TextLink_170" />
  <target targetId="window_3" />
```

Chapter 6 ReversiXML

```
<!--targetId= "http://www.isys.ucl.ac.be"-->
</graphicalTransition>
```

A graphical transition is characterized by three parameters: its **name**, **source** and **target**. The **name** is composed of the two letters “gt_” and the number of the transition. The **source** is the id of the interaction object which gives access to the next window, and the **target** is the id of the window (or another URL) to which the dialog transition gives access. In this example, the URL has been put in a comment and replaced by the id of the window, as it will be reverse engineered during the same session.

LINKS		
num	Source	Target
1	Image_36	http://www.sedan-bouillon.com/catalogue/
2	Image_41	http://www.sedan-bouillon.com/catalogue/ctg.php?c=10
3	Image_46	http://www.sedan-bouillon.com/catalogue/ctg.php?c=11
4	Image_51	http://www.sedan-bouillon.com/catalogue/ctg.php?c=12
5	TextLink_52	http://www.sedan-bouillon.com/catalogue/ctg.php?c=13
6	TextLink_59	http://www.sedan-bouillon.com/catalogue/prdt.php?c=11&s=24&p=5
7	Image_70	http://www.sedan-bouillon.com/catalogue/prdt.php?c=11&s=24&p=5
8	Image_77	http://www.sedan-bouillon.com/catalogue/commande.php
9	Image_84	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=63
10	Image_91	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=64
11	Image_98	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=53
12	Image_105	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=52
13	Image_115	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=61
14	Image_122	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=54
15	Image_129	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=56
16	Image_136	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=62
17	Image_143	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=59
18	TextLink_147	http://www.sedan-bouillon.com/catalogue/ssctg.php?c=13&s=
19	TextLink_153	http://www.sedan-bouillon.com/catalogue/qui.php
20	TextLink_155	http://www.sedan-bouillon.com/catalogue/contact.php
21	TextLink_157	http://www.is3.fr/

Figure 6-10 Link table

6.4.3 Differences between the two tools

Based on previous work, ReversiXML enlarges the possibilities and scope of the research already conducted during the development of Vaquita. The objective is to develop a flexible reverse engineering tool able to produce dynamically a UI description suited for a particular context of use. Table 6-1 represents the major differences between the two reverse engineering tools of Web pages. First of all, the output language is XIML for Vaquita and UsiXML for ReversiXML. The fact that this language has been developed within the Cameleon Project allows us to distribute ReversiXML without license, which is not the case for Vaquita. ReversiXML allows the reverse engineering of a UI at the concrete or abstract

Chapter 6 ReversiXML

level (and only at the concrete level for Vaquita). Another major difference is the possibility to use the tool manually or automatically. The user can also choose to let ReversiXML apply the most suited reverse engineering rules following the connected platform (thanks to a system able to recognize the current context of use). The use of Vaquita was at design time and entirely manual and the designer had to choose every reverse engineering rule. ReversiXML can also be used on a server and directly process files on the Web, while Vaquita can only reverse engineer local files.

The programming language is completely different (VB 6 for Vaquita and PHP 5 for ReversiXML) and this implies a shorter processing time as PHP is faster than Visual Basic (about 20 times faster in when used in the same conditions, i.e. in a local reverse engineering process). This last parameter is very important in the hypothesis of a dynamic reengineering: users who want to access an ordinary Web page with their mobile platform will not have to wait too long for the regeneration of the UI. The speed difference is mostly due to the fact that the DOM libraries for VB6 (msxml3.dll) uses memory in a non-optimal manner, causing delay in the process, and that the generation of the UI of Vaquita is particularly slow (drawing of the trees).

Vaquita was only able to reverse one page at a time but ReversiXML gives the possibility to process several pages in one request, or even an entire site. This implies the reverse engineering of the navigation model (or dialog). Vaquita was only able to recover the presentation model, but in ReversiXML, a part of the dialog model is also recovered (the navigational structure of the Website).

Another important point is the interoperability with other tools. The number of partners of the XIIML consortium was promising, but only one 'official' tool is under development after four years (put aside enviro3D). This fact prevents us from testing a complete reengineering process on classical 2D UIs. With UsiXML, several tools already exist or are under development. See appendix E for more details about other UsiXML compliant tools. The supported relations are richer in ReversiXML than in Vaquita. Vaquita was only able to establish semantic relations between labels and other interaction objects in an assisted way, while ReversiXML recovers dialog and layout relations automatically.

	Vaquita	ReversiXML
Abstraction Level	Concrete UI	Concrete and Abstract UI
Output language	XIIML	UsiXML
Mandatory license	Yes	No

Selection of RE rules	Manual	Manual and automatic
Use	Local	On-line or Local
Processing Time	1	0.05 (20 times less)
Programming Language	Visual Basic 6	PHP 5
Code Size	About 10,000 LOC	About 12,000 LOC
Site reverse engineering	No	Yes
Reverse engineering	Static	Static/ Dynamic
Models	Presentation	Presentation & Dialog
Application Size	About 800 Kb	About 600 Kb
Context awareness	No	Yes
Data-bases	UI model	UI model, context, configurations
Interoperability with other tools	Orca (ongoing development), Envir3D	TeresaXML, GrafiXML, TransformiXML, FlashiXML...
Semantic Relations	Yes	No (ongoing consideration)
Layout Relations	No	Yes
Dialog Relations	No	Yes
Design time/Run time	Design time	Both
Interactive/Batch	Interactive	Both

Table 6-1 Differences between Vaquita and ReversiXML

6.4.4 Ongoing issues and future work

There are still some issues that are not solved at this point of the development:

Embedded components: an increasing number of pages contain Flash or Java components. The reverse engineering of these components is impossible as they use an imperative language which is compiled. The current method to solve this problem in UsiXML is to create a `finalComponent` with the URL of this component. Another method will be investigated in a near future to detect if the component contains links: when the parser recognizes a flash component, the page will be displayed and the designer will be asked to use the components. If the component contains links, the redirection will be recorded by ReversiXML and put in menu in the place of the component. Obviously, this method will only work for the manual version or ReversiXML (at design time).

Scripts: As stated in the working hypotheses, scripts are not analyzed as their reverse engineering method and objectives are completely different from the

reverse engineering of UIs. However, a similar solution to the embedded components could be used, i.e. link extraction, and in a simpler way: as scripts are not compiled, it could be possible to detect URLs and build a menu with all the links from these scripts. In this manner, the reverse engineered model would preserve the access to the entire navigational structure.

Semantic Relations: these relations between components to specify that they have to be displayed together are not currently supported in the UsiXML language. However this kind of relation is very important for the redistribution of the UI on several windows. For the moment, the **box** mechanism can be used to represent these semantic relations, but it is impossible to distinguish these **boxes** from **boxes** used to structure the UI. This relation will probably be added in the language as in ReversiXML. The detection of this type of relations is also a research subject in [Gaer03] and [Xian06].

Use of statistics to refine the reverse engineering: thanks to the statistics related to the composition of the UI, it would be possible to make a better selection of the retargeting rules. By adding several statistics (such as the measure of the dispersion of widgets, the number of characters, the number of different widgets, etc...), some heuristics could be added to have a more accurate reverse engineering. For example, it would be significant for the elision of paragraphs technique (the first words of the paragraph are transformed into a link that gives access to the rest of the paragraph). The technique should be applied for long text-based pages (measured by the total number of characters), whereas this would not be applied for pages with less text (but maybe with longer paragraphs). Indeed, it will require less time for the user to scroll through the page in this second case and will give more meaningful results than an individual rule like 'transform each paragraph with more than 200 characters into a link'. By taking into account this global information, ReversiXML could produce models of better quality thanks to a reverse engineering based not only on the target platform/user, but also on the composition and overall structure of the source UI (patterns).

Reduction of the amount of boxes: as the algorithm for the generation of **boxes** is very exhaustive, i.e. for each HTML tag modifying the layout a **box** is created without checking if it makes sense, the results should be corrected to minimize the number of **boxes**. Examples or reduction rules are given in appendix J.

The fact that the source code contains too many table tags comes generally from HTML editors such as Microsoft FrontPage or DreamWeaver. These editors offer a WYSIWYG process to build the Web page and this technique adds several 'layout tags' to represent accurately the UI as drawn by the user.

Allow universal Web pages reverse engineering: All the Web pages are not accessible by ReversiXML as the automated extraction mechanism of Web pages via URLs currently suffers from two weaknesses: firstly, the correction and transformation in the XML format thanks to Tidy sometimes generates imperfect files, i.e. not valid XML files that can not be parsed by the libxml2 library. In those cases, the reverse engineering is simply not achieved. Some correction after the tidy-process could be added to the tool in order to avoid this problem. Secondly, some pages are inaccessible because the Web server does not identify ReversiXML as a normal browser. This is a Web crawler protection of Websites to avoid unusual accesses of their content, such as from spamming tools etc... In this case, the code is not available, and the only solution would be to emulate a Web browser in the download module of ReversiXML.

Pursue further implementation: not all the requirements are fully addressed and new features could be added in the implementation of the current version of ReversiXML:

- The architecture could evolve to the version 1.2 as presented in appendix I.
- The elements to recover a partial context model could be added to the tool (see appendix D).
- The selection mechanism of the best configuration file has been tested on a few cases, but a complete database of configuration files should be developed, by associating a configuration file for each existing platform.
- The reverse engineering of entire sites is not completely implemented.

6.5 Conclusion

This chapter decided to address the main shortcomings identified in Vaquita, an early attempt for supporting the UI reverse engineering. For this purpose, new working hypotheses have been adopted and justified that lead to the definition of a series of progressively more complex software architectures, an enhanced version of the UI reverse engineering process, and an implementation of a tool.

This chapter could be considered as a second attempt to address the general requirements identified in Chapters 1 and 3 according to a depth-first approach: while staying on the same problem, we choose to go deeper in the sophistication of the process support, although some other extensions have been identified. Both tools can be compared in terms of the trade-off between the levels of complexity and coverage (figure 6-11).

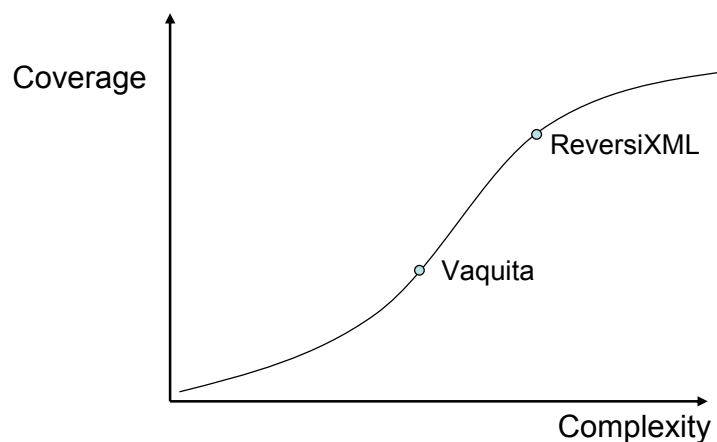


Figure 6-11 Coverage and complexity trade-off

Vaquita had a level of complexity lower than ReversiXML, but also less coverage of the source UI features. With ReversiXML, we reached an optimal level in this trade-off, as further enhancements would require lots of efforts in development compared to few advantages in the coverage (e.g. the evolution to the architecture 1.2 of the tool would bring marginal benefits to the approach but would necessitate important resources for the implementation; similarly, benefits of the reverse engineering of the dialog is disproportioned compared to its “cost”, and would also imply difficulties in the forward engineering).

In the next chapter, we will adopt a breath-first approach to examine other related declarative languages in which a UI could be implemented to identify what are the content of this thesis which remain operational and valid to solve the problem. However, it will not be our goal to systematically develop some software support for each of those cases. Only some selected portions will be subject to some implementation.

Chapter 7 Reverse engineering of other markup- based UI

In order to widen the scope of UI declarative languages to be used as a source for reverse engineering, this seventh chapter is about the reverse engineering of two selected markup-languages: the WML and the VoiceXML languages. The first study of WML is about the recovery of UIs designed for platforms with very limited capabilities (but with the same interaction modality: the graphical channel) while the second, about VoiceXML, is the reverse engineering of UI designed for another computing platform (i.e., a mobile phone or a computing platform equipped with voice synthesis) but with another interaction modality (the vocal channel). The study of WML is based on the bachelor thesis of Xuefeng Cui [Cui05] who has conducted his work under our supervision during the last academic year. Each language study, representing sections of this chapter, is divided into four parts, firstly a brief presentation and history of the language, secondly the scope of the analysis, thirdly the presentation of representative derivation rules and a meta-model of the language and finally some examples of reverse engineering. A tool has also been developed for the reverse engineering of WML, and its results are also presented in this chapter. Finally, a third section concludes this chapter.

7.1 Reverse engineering of WML

7.1.1 The WML language

The growth of internet-enabled mobile phones is much higher than the growth of desktop PC (see chapter table 1-1 figures), and it is obvious that in a few years, this kind of platforms will be more exploited than classical desktop platforms to access the Internet. We are only in the beginning of this process, and therefore the use and proportion of pages written in the WML language seems weak for the moment but will be more important in a near future. To satisfy user needs, the mobile platforms market provides a wide variety of devices with different capabilities such as screen resolution, screen size, number of colours, memory... (see figure 7-1).

The possibilities for developing “mobile” UIs can be characterized by the following constraints (compared to the desktop computer’s UIs):

- 1) Small display and limited user input facilities
- 2) Narrowband network connection
- 3) Limited memory and computational resources.
- 4) Important abeyance time during a connection.
- 5) Unstable connections which can suffer of quick damages.

All these factors imply very simple/short UIs, mostly composed of text, with minimal input of the user.

WML history is shorter than a decade. The WAP forum has been created in 1997 by the most important actors in the mobile phone market (Nokia, Motorola, Ericsson, Microsoft, Phone.com, etc.). The first version of WML was elaborated in May 1998. The version 1.1 has been released in 1999, the version 1.2 in September 1999 and version 1.3 in 2000. Currently, the version 2.0 has been released (since end of 2001) but it has not been really adopted by the majority of WML developers and Web services providers. This new version is based on XHTML and can be used to deliver WML 1.x content to WAP 2 client, achieving backward compatibility with WML 1 in a manner that is transparent to the end user. Another language, HDML (handheld device markup language), was developed previously, but it had too many shortcomings (especially portability /compatibility, it could not be used on every mobile phone) and was abandoned by most of the developers with the adoption of WML.

A competitor of the WML is cHTML (compact HTML). It was released in 1999 and is in fact only a subset of the HTML language (with some specific

Chapter 7 Reverse engineering of other markup-based UI

functionalities) created for “small information devices”. It was never considered as a standard by W3C and is actually mostly exploited in Japan (i-mode). It had several advantages over WML (use a well-known language, website easily adaptable to cHTML, very simple use) but the fact that W3C did not recognize it as a standard limited its expansion.



Figure 7-1 Mobile devices with different capabilities

WML can almost be considered as a subset of HTML too, as the differences between the two languages are very small. A few specific tags have been added, but 70% of the language tags and attributes are the same as in the hyper text markup language. The major difference is the introduction of the card metaphor:

instead of having a single window stored in a file (one file equals one window), a WML file contains a set of cards (a deck) representing several windows. These cards are grouped logically so that a user can quickly access different related topics on a website by avoiding waiting for the loading of every new card (window).

There is also, as in HTML, a scripting language, allowing for applying simple processing on variables. WMLScript is able to take into account some characteristics of the connected platform (and adapt the deck accordingly) and to add some dynamic behaviour to the developed decks.

Finally, the WML language can be summarized by four major functional areas [Cui05]:

- Text presentation and layout - WML includes text and image support, including a variety of formatting and layout commands
- Deck/card organisational metaphor - all information in WML is organised into a collection of cards and decks
- Inter-card navigation and linking - WML includes support for explicitly managing the navigation between cards and decks
- String parameterization and state management - all WML decks can be parameterized, using a state model.

7.1.2 Working Hypothesis

The goal of this chapter is to apply the method developed for the HTML language to another markup-language created for a completely different type of platform (different in terms of capacity and interaction techniques), mobile phones. The analysis of this language is not as deep as for the HTML language and this for three reasons:

1. The language and the different kinds of UIs that can be produced with WML are much less complex than for HTML (i.e. no complex layout, fewer UI elements, no sounds, no embedded components, etc.)
2. The language is currently not so widespread than the HTML language, and thus its importance and its use are smaller than for HTML.
3. There are fewer incentives for the reverse engineering of WML than for the HTML language, as the reengineering of WML UIs towards platforms with more capabilities is less likely than the contrary.

There are several versions of WML (see 7.1.1), and only the version 1.1 has been fully analyzed in this thesis. The version 1.1 is the most popular version (the version 2.0 has been released too recently and can not be fully exploited for the

moment), most WML sites are developed in this version, so as development toolkits and simulators.

As for HTML, the reverse engineering of the dialog is limited to the navigation. The reverse engineering applied in this chapter is a static application of derivation rules. The target model for this study is the UsiXML v1.4.6 CUI only, but the method could be enhanced by adding similar rules to recover the AUI level.

It can be argued that the reverse engineering of this language is not very useful, but it has been analysed in this thesis for three main reasons: firstly, it can be used to redesign existing pages (for example for another mobile platform with slightly different capabilities, e.g. larger screen, some colour-capabilities, widgets availability etc.). Secondly, the CUI model could possibly be used in a translation process to convert it partially or in its totality in a vocal UI, thus resulting in transformation of the former WML UI into a multi-modal UI. Thirdly, this analysis enlarges the scope of the reverse engineered languages and thus the proposed method, by analyzing the adequacy of the notation for this language and how the different issues related to HTML reverse engineering apply to WML (see chapter 4).

Losses due to the reverse engineering process

As the layout and presentation of the UI is very simple, there are very few (if any) losses due to the reverse engineering process at the presentation level. Most of the information contained in a deck is kept in the concrete UI model except some dynamic aspects. As for HTML, only the static part of the UI is represented in the UI model. Variables and states are not recorded in the UsiXML specification, so as WML scripts and tags using the concept of history.

7.1.3 Language meta-model and derivation rules

This section contains the meta-model [Cui05] and some specific rules to the WML 1.1 language. The meta-model (figure 7-2) is characterized by its simplicity, compared to other language meta-models (see appendix B). The root element of the model is a `wml` element, which can contain a `meta`, `template` or `card` node. The UI is contained in `card` elements and can be composed of navigation elements (`+navelements`), `timer`, paragraphs (`p`) or fields (`+fields`). Fields are decomposed in several input elements (`select`, `input...`) and flow elements (`+flow`) that represent formatting tags for the text of the UI, such as `b` or `i`(for bold or italic text), `tables`, `links(a)` and images (`img`). Elements starting with a plus symbol represent super

Chapter 7 Reverse engineering of other markup-based UI

classes, grouping several types of elements sharing similar attributes and aggregation relations.

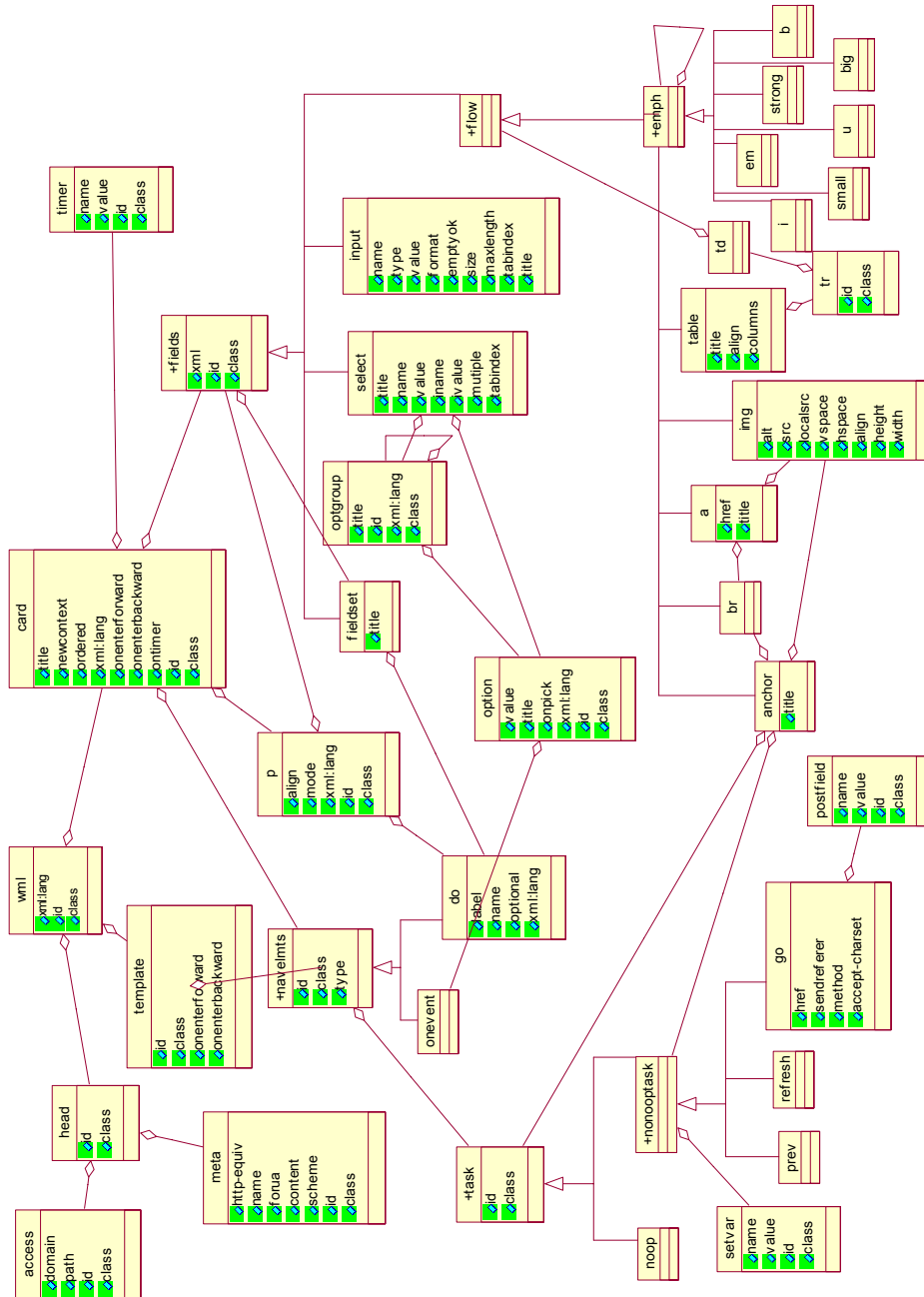


Figure 7-2 The meta-model of WML 1.1

The complete set of derivation rules for WML based on this meta-model is presented in appendix C. There are several rules similar to the derivation rules of HTML 4.0, and therefore most of them have been combined with the HTML rules (combined rules are characterized by a $T_{H/W}$ in the left part of the rule in appendix C). In fact, two cases exist, either the rule is identical and the rules of HTML and WML are simply merged or the WML rule contains less constraints than for the HTML language, and the rules can also be combined if the constraint is that an element or attribute does not exist or is equal to zero (constraint of inexistence).

There are very few rules that are only applicable to the WML language in appendix C. These rules are characterized by the fact that they only contain a T_w in the left part of the rule.

Only four completely specific groups of rules for the WML exist, for elements that do not exist in HTML. There are of course more differences between the two languages but in most of the cases, they are mixed in a group of rules for other languages. These four groups of rules are for the **template**, **card** (replacing windows), **select** (translated here as **comboBox**, **radioButtons** or **checkboxes** following its attributes), and **do/go** nodes. The derivation rules for two of these components are shown in the rest of this section.

The transformation of the **select** node in a **comboBox** (a drop down list box) depends on three conditions: the number of different **options** must be greater than 6, the **options** depending on the **select** node should not possess an **onpick** attribute and only one choice can be made (**multiple** selections are not allowed). In the other cases, the **select** node is translated by a **radioButton**, a **textComponent** or a **checkbox**, but this is done when an option node is detected (G9)

G8 – select (comboBox)

```

 $\forall x,y \in T_w : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$ 
 $\rightarrow \text{Addnode}(\text{"comboBox"}, \text{idcombo}) \text{ where } \text{idcombo} = \text{NodeAmount}(T)$ 
 $\forall x,y \in T_w : x = \text{select} \wedge (x.\text{value} \neq \text{NULL}) \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"varValue"}, x.\text{value})$ 
 $\forall x,y \in T_w : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isEnabled"}, \text{"true"})$ 

```

```

 $\forall x,y \in T_W : x = \text{select } \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"id"}, \text{idcombo})$ 
 $\forall x,y \in T_W : x = \text{select } \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"name"}, \text{idcombo})$ 
 $\forall x,y \in T_W : x = \text{select } \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isVisible"}, \text{"true"})$ 
 $\forall x,y \in T_W : x = \text{select } \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isEditable"}, \text{"false"})$ 
 $\forall x,y \in T_W : x = \text{select } \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isDropDown"}, \text{"true"})$ 
 $\forall x,y \in T_W : x = \text{select } \wedge x.\text{tabindex} \neq \text{NULL} \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"mnemonic"}, x.\text{accesskey})$ 
 $\forall x,y \in T_W : x = \text{select } \wedge(x.\text{multiple}=\text{"false"} \vee x.\text{multiple}=\text{NULL}) \wedge$ 
 $(\text{CountInChildNodes}(x,\text{option})>6) \wedge y=\text{option} \wedge \text{isInPath}(y,x)=\text{true} \wedge y.\text{onpick}=\text{NULL}$ 
 $\rightarrow \text{CheckAligment}(x,\text{idcombo})$ 

```

The **do** node is a “task” node, allowing the user to activate a task when he clicks on word/phrase on the screen. The only task that is considered in this analysis is a navigation task (by using the **go** node to specify the URL of the link) and in this case the **do/go** nodes are translated by a `textComponent`.

G21 – do/go (textComponent)

```

 $\forall x,y \in T_W : x = \text{do} \wedge x.\text{type}=\text{accept} \wedge y=\text{go} \wedge y \in \text{childNodes}(x)$ 
 $\rightarrow (\text{"TextComponent"}, \text{idtext}) \text{ where } \text{idtext} = \text{NodeAmount}(T_t)$ 
 $\forall x,y \in T_W : x = \text{do} \wedge x.\text{type}=\text{accept} \wedge x.\text{label} \neq \text{NULL} \wedge y=\text{go} \wedge y \in \text{childNodes}(x) \rightarrow$ 
 $\text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{"x.label"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"})$ 
 $\forall x,y \in T_W : x = \text{do} \wedge x.\text{type}=\text{accept} \wedge y.\text{href} \neq \text{NULL} \wedge y=\text{go} \wedge y \in \text{childNodes}(x) \rightarrow$ 
 $\text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{"x.label"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"})$ 
 $\text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, y.\text{href}) \wedge \text{AddGraphTr}(\text{idtext}, y.\text{href})$ 

```

7.1.4 Tool support

A tool has been developed in [Cui05] to support the reverse engineering of the WML 1.1 language. The reverse engineering of WML could also be added easily to the ReversiXML tool, as very few rules would be added to the current engine.

Implementation

The tool developed in [Cui05] is composed of a small java program launching an XSLT transformation process. Xpath expressions are also used to compute some attribute values that cannot be directly mapped thanks to XSLT.

The tool has to be used in a command-line style, as shown on figure 7-3, by indicating the XSLT stylesheet to use during the reverse engineering, the source file name and output file name.



Figure 7-3 Command prompt for the WML reverse engineering tool

Examples

Two commented examples of UsiXML CUI models obtained with the tool are available in appendix H-2.

Complete reengineering

A complete reengineering has been realized by using both the WMLtoUsiXML and Grafxml tools. The original WML cards are presented in figure 7-4. The WML and UsiXML code is shown in table 7-1. The reengineered card is the yahoo mobile login page. On this UI, the user has to give its login and password and submit this small form. She can also ask for help by pushing on the help link or navigate to the yahoo home page (the link is not visible on the figure).

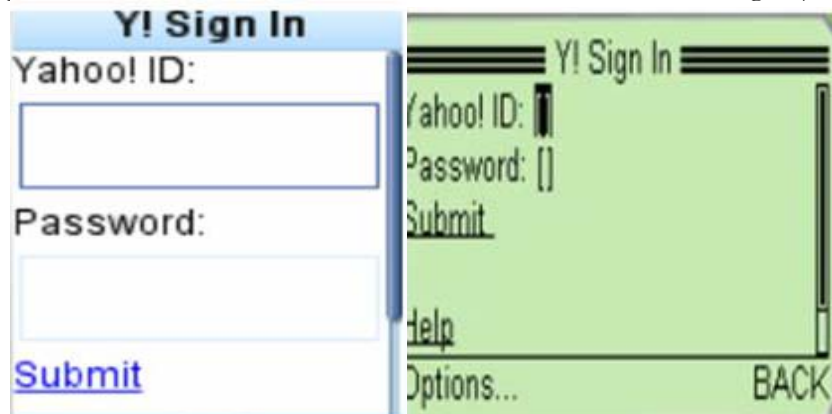


Figure 7-4 Yahoo mobile login page

WML	UsiXML
<pre><wml><head> <meta http-equiv="Cache-Control"</pre>	<pre><CuiModel xmlns="http://www.usiXML.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema</pre>

Chapter 7 Reverse engineering of other markup-based UI

<pre> content="max-age=0" forua="true"/> </head> <template><do type="prev" label="BACK"><prev/></do></template> <card id="c2" title="Y! Sign In"> <onevent type="onenterforward"> <refresh> <setvar name="passwd" value="" /> <setvar name="src" value="sGHW" /> </refresh></onevent> <onevent type="onenterbackward"> <refresh> <setvar name="passwd" value="" /> <setvar name="src" value="sGHW" /> </refresh></onevent> ((1)) <p> Yahoo! ID: <input name="login" value="" format="*m" />Password: <input type="password" name="passwd" value="" format="*m" /> ((2)) <anchor title="OK">Submit <go method="post" href="/raw?"> <postfield name="dp" value="auth"/> <postfield name="s" value="1"/> <postfield name="src" value="\$(src)"/> <postfield name="login" value="\$(login)"/> <postfield name="passwd" value="\$(passwd)"/> </go></anchor>

 Help
 ((3)) Yahoo! Home </p> </card> </wml> </pre>	<pre> ma-instance"> <window id="window_1" borderTitle="Y! Sign In" isVisible="true" isEnabled="true" name="Y! Sign In"> <box id="box_1" isVisible="true" isEnabled="true"> <textComponent id="textComponent_1" glueHorizontal="left" isVisible="true" isEnabled="true" defaultContent="Yahoo! ID:"/> <textComponent id="textComponent_2" glueHorizontal="left" defaultContent="" isEditable="true" isVisible="true"/> <textComponent id="textComponent_3" glueHorizontal="left" isVisible="true" isEnabled="true" defaultContent="Password:"/> <textComponent id="textComponent_4" glueHorizontal="left" isPassword="true" defaultContent="" isEditable="true" isVisible="true"/> ((2)) <textComponent id="textComponent_5" glueHorizontal="left" isVisible="true" isEnabled="true" defaultHyperLinkTarget="/raw?" name="OK" defaultContent="Submit"/> <textComponent id="textComponent_6" ((3)) glueHorizontal="left" isVisible="true" isEnabled="true" defaultHyperLinkTarget="http://wap.oa.yah oo.com/raw?dp=page&pg=help/signin" defaultContent="Help"/> <textComponent id="textComponent_7" ((3)) glueHorizontal="left" isVisible="true" isEnabled="true" defaultHyperLinkTarget="http://wap.oa.yah oo.com/raw?dp=home" defaultContent="Yahoo! Home"/> </box></window> <graphicalTransition id="textLink_1" type="open"> <source sourceId="textComponent_5"/> <target targetId="/raw?"/> ((3)) </graphicalTransition>... </CuiModel> </pre>
--	--

Table 7-1 Reengineering of Yahoo mobile login page

((1)) An entire part of the WML code is not used during the reverse engineering. The `template` node is not reverse engineered as it contains a reference to the history. The other nodes `onevent` are not analyzed too as this type of information is not kept in the UsiXML models.

((2)) This part of the UI contains two labels, “Yahoo! ID” and “Password”, and two textboxes (`<input>` nodes). These elements are mapped as four `textComponents` in the UsiXML specification by applying the rule sets G7 (for inputs) and G17 (for labels). The last `textComponent` possess the `isPassword`

attribute as it is the translation of the password field (`<input type=password ...>` in WML).

(3) The two links from the WML code are transformed into `textComponents` in the CUI model (G17 rule set). The corresponding `graphicalTransitions` are also added at the end of the model.

The result of the forward engineering, based on the specification of table 7-1, is shown on figure 7-5. As the Graftxml is not able to generate a final UI for the moment, only the preview window is available.

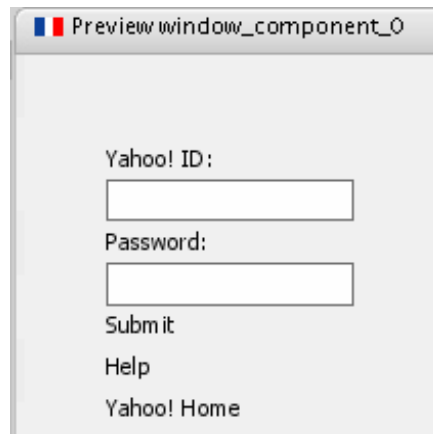


Figure 7-5 Preview UI generated with Graftxml

7.2 Reverse engineering of VoiceXML 2.0

7.2.1 The VoiceXML language

The VoiceXML language is a very young language, created one year after the WML language. VoiceXML [Roui04] is a markup language for representing human-computer dialogs, allowing developing vocal application rapidly. VoiceXML assumes a specific voice browser (such as Vocal ADK, Voxeo or Websphere) with audio output/input capabilities.

The first research conducted on vocal UIs started in 1995 at AT&T Research Labs. In 1999, AT&T and Lucent had developed two incompatible dialects of the Phone Markup Language (PML), while Motorola had its own language (VoxML). Other companies such as IBM, with the SpeechML, also started research on this subject. They decided to organize the VoiceXML Forum, in order to standardize these separate languages. In August of 1999, the forum produced VoiceXML 0.9, combining the best features of the earlier languages. After the publications of the

VoiceXML language, the forum had a tremendous growth, and several comments of the new participants helped to produce the VoiceXML 1.0 in March 2000. In May 2000, the W3C accepted the language as standard as more and more companies were using it and were involved in the development of this markup language. In October 2001, a first version of the VoiceXML 2.0 was submitted (aiming mostly at correcting errors of the first version), and became a W3C candidate recommendation in January 2003 [Voic04].

The importance of the language is increasing constantly, as the number of “potential platforms” grows accordingly. It is estimated that there are over 1.5 billion of phones in the world, which is much more than personal computers. Moreover, the technology related to VoiceXML (such as text-to-speech systems or vocal recognition systems) is becoming more sophisticated and this will thus promote the use of this language.

VoiceXML is different from the two previous studied markup languages, as several particularities were introduced and are listed below:

- VoiceXML integrates conditional tags or event catcher tags (predefined or user-defined events). Moreover, the EcmaScript language is strongly incorporated in the VoiceXML language and allows developers to modify control flows during the execution of the UI without having to call a complex function.
- It also allows the use of libraries and functions (e.g. by using the **subdialog** tag), promoting their use and the development of modular applications.
- The use of two interaction devices: the user can interact by using voice or the keyboard of the telephone instead of mouse and keyboard inputs.
- The language introduced the mixed-initiative in the VoiceXML forms. When a form is put in the mixed-initiative mode, users can fill fields of the form in any order, and do not have to wait for a specific question to give their answer.

7.2.2 Working Hypothesis

The aim of this section is to apply the method developed for the HTML language to another markup-language created for a different modality. The targeted output of this study is the CUI level expressed in UsiXML v1.4.6, recovered thanks to a static analysis of VoiceXML code. The recovery of dialog is limited to the navigation between VoiceXML applications.

Similarly to WML, the analysis of this language is not as deep as for the HTML language and this for several reasons:

Chapter 7 Reverse engineering of other markup-based UI

1. The UsiXML language provides more support for graphical UIs than for vocal UIs, and several aspects of vocal UIs have to be ignored during the reverse engineering (see the section “losses due to reverse engineering”).
2. The language is not so widespread as the HTML language, and therefore there are fewer needs for reverse engineering with this language.
3. There are also fewer incentives for the reverse engineering of VoiceXML than for the HTML language, as the produced models can not be reused for several targets in the forward engineering phase. This is due to the nature of vocal UIs, as these UIs are developed to be rapid and simple (such as weather forecast, restaurant reservation etc.).

The version that has been analyzed in this section is the second version of the VoiceXML language. Anyway, versions are very similar (as the difference between 1.0 and 2.0 are mostly corrections and some additions to take some particularities of platforms into account). As for the preceding analysis, the question of the interest of the reverse engineering of the VoiceXML can be asked. Three main points of interest can be highlighted.

- First of all, some particular specifications of vocal UI could be reused for redesign, or for the forward engineering towards another platform with limited capabilities. A potential target could be WML as this language is also used to design small and simple UIs.
- Secondly, a more interesting use could be the integration of a (partial) vocal UI into a multi-modal UI (graphical-vocal UI). This kind of UIs will probably be more important in the future, mixing HTML’s information presentation capabilities with some vocal commands/forms. We could imagine that some parts of a vocal UI could be reused and integrated with the “graphical part” of a CUI model (similarly to the VoiceXML transcoding approach in chapter 3).
- Finally, this enlarges the scope of the method proposed in this thesis, showing how the reverse engineering process can be generalized while pointing out some limits of the method (see next section).

Losses due to the reverse engineering process

The reverse engineering of VoiceXML in UsiXML specifications causes several losses, especially because conditional behaviours and complex control flows can not be represented in the output language. Therefore, a part of the structure of the VoiceXML UI can not be recovered, such as conditions, events catching or

the repetition of instructions. This implies a less detailed analysis of the language and the loss of complex dialogs.

Tags that modify the control flow or that modify values of variables (representing usually states of the UI) will not be taken into account, as the UsiXML language has not been designed to represent imperative languages and consequently such structures. Two attributes also exist, **cond** and **count**, which allow applying a condition on a specific tag. The value of **cond** is evaluated before the (attribute-) owner tag is activated, and **count** represents the number of times the preceding prompt has to be activated before the (attribute-) owner tag becomes active. Again, these two common conditions are not represented in a UsiXML specification for the moment. The VoiceXML language allows defining grammars in a UI specification but it is also impossible to represent this information in a UsiXML model. It is possible to record the path of an external grammar, but there is no structure to accommodate the definition of an internal grammar in a UsiXML specification. A solution could be to transform an internal grammar into an external grammar by putting the content of the VoiceXML grammar in an independent file. The complete list of tags that are ignored is given below (table 7-2) so as the category of concept to which they belong.

Tags	Category
if, then, else, elseif	Conditional tags
catch, help, exit, disconnect, noinput, nomatch, throw,	Event catcher tags
Item,rule,tag,ruleref,one-of	Grammar definition tags
Count=..., cond=...	Conditional attributes
Value, var	State variables or user inputs

Table 7-2 Tags ignored during reverse engineering

Two solutions are possible for the elements depending on those tags: either ignore them or keep it in the UsiXML specification, but under the form of comments. In this latter case, the designer will be able to select the elements to be kept in the model (even without conditions) or the one that can be discarded. He could also put the conditions again manually after a forward engineering phase.

7.2.3 Language meta-model and derivation rules

This section contains the meta-model of the voiceXML 2.0, the portion of interest of the CUI level in UsiXML for vocal languages and some commented example of derivation rules from the VoiceXML to UsiXML CUI level.

Chapter 7 Reverse engineering of other markup-based UI

Contrary to WML, VoiceXML and HTML are two completely different languages, and rules cannot be combined. 24 element-derivations rules, two general rules and two target-tree rules are needed for the reverse engineering of this language. A partial list is given in chapter 4 and is completed by appendix C. The meta-model of the VoiceXML 2.0 language is shown on figure 7-6, but it is a lightened version, as some simplifications have been made (all the possible containment relations are not shown in this version of the model) in order to make it readable. The complete version can be found in appendix B. The root of this meta-model is a `vxml` element, which can contain meta information (`meta` and `metadata`), link, property, events handlers (`+event handler`), containers and input (`+container`) or executable content (`+executable content`). The VoiceXML UI is embedded in a node belonging to the containers class. This superclass (preceded by a + symbol) groups logical containers (`block`, `initial`) and form-input elements (`field`, `record`, ...) as they share common attributes and properties. The `event handler` superclass contains several predefined event (`help`, `noinput`...) so as user-defined catchers (`catch`). Finally, `executable content` is the class for the rest of UI components. It contains tags allowing modifying control flow (`if`, `then`, `else`, `return`...) so as output nodes (`prompt`, `audio`). Executable contents and event handlers are characterized by the fact that these elements can not embed another element of the same class, contrary to containers.

The portion of interest of UsiXML for the reverse engineering of vocal UIs is shown on figure 7-7.

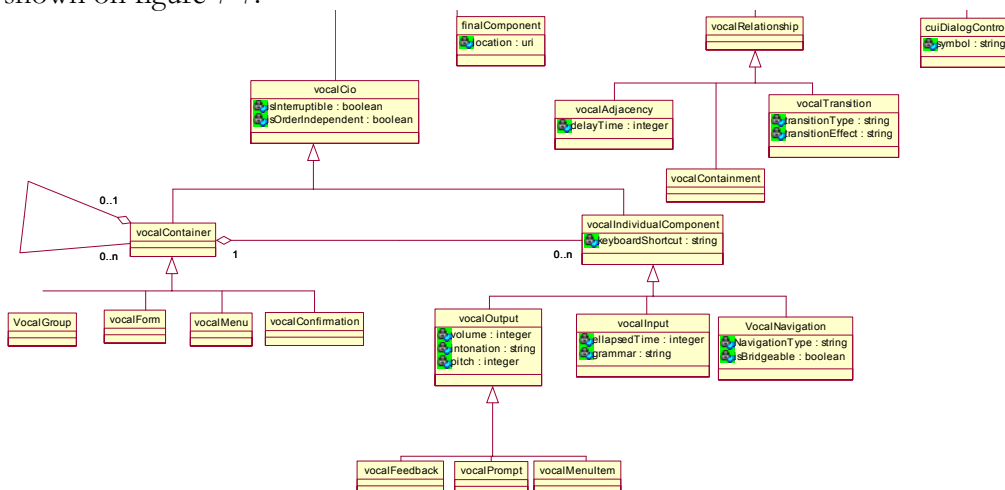


Figure 7-7 Portion of the CUI used to represent vocal UI

Chapter 7 Reverse engineering of other markup-based UI

We can consider that there are three main categories of elements: containers, individual components and relations. Containers are subdivided into four categories, **VocalForms** (contains inputs), **VocalMenu** (contains a series of **menuitems** to be selected), **VocalConfirmation** (contains a confirmation task represented by a simple input) and **VocalGroup** (contains elements logically related). **VocalIndividualComponents** are divided into three types: inputs, outputs (prompt for simple outputs, items when the output belongs to a menu and feedback to represent responses of the system) and navigation. A more complex attribute exists for this last category, the **isBridgeable** attribute. When set to true, it represents that the current document remains active when the navigation is triggered. All these elements inherit the **isInterruptible** and **isOrderIndependent** attributes. The first means that the user is able to interrupt a process by saying a “hotword” (defined in browser). The second represent the mixed-initiative concept. When set to true, the input fields can be filled at any moment during the execution of the vocal UI. It can be applied to containers or to inputs. Finally, there are four possible relations between vocal elements: adjacency to represent a delay between two elements, containment, transition to represent navigation and **CUIDialogControl** to specify some specific control flows such as the element A has to be played before element B etc... Only a part of the figure 7-7 is used during the derivation from VoiceXML to UsiXML. Three relations (**vocalTransition**, **CUIDialogControl** and **vocalAdjacency**) are needed during the reverse engineering, three containers (**vocalGroup**, **vocalForm** and **vocalMenu**) and five individual components: **vocalPrompts**, **vocalMenuitems**, **vocalInputs**, **vocalNavigation** and **finalComponents**. Most of the elements of a VoiceXML UI can be represented by **vocalGroups**, **vocalPrompts** and **vocalInputs**.

The current specification of the UsiXML language puts the **grammar** of components as an attribute of **vocalInputs** whereas in the VoiceXML language **grammars** can be defined for several categories of elements (**forms**, **fields** ...). Some losses of information are due to this fact, but it has been minimized by making inherit **form grammars** (i.e. applying to a set of components) to every element contained in the form in the UsiXML specification.

Some representative examples of derivation rules are given in the rest of this section.

The first example is about the derivation of a **field** node into a **vocalInput** in UsiXML. When a **field** node is detected, two nodes are created in the target tree. The first node is a **vocalPrompt** representing the **prompt** element contained in the

field element and the second node is a `vocalInput` corresponding to the `field`. The `prompt` is first processed to follow the sequence question-answer (whereas tags are specified in the inverse order in VoiceXML). There can be three sources for the attribute `grammar` in the UsiXML specification:

1. The `field` element possesses a `type` attribute, and its value is copied in the UsiXML's `grammar` attribute
2. The `field` is embedded in a `form` that possesses a `form-grammar`, and in this case the `isOrderIndependent` attribute is set to `true` and the `form grammar` is copied into the `vocalInput`'s `grammar`.
3. The `field` possesses a `grammar` node (this case is not processed in this rules set but in G38).

If the `field` element possesses two `grammars` (one defined in the element and the other in the `form`), the element's `grammar` (last `grammar` reference encountered) is recorded.

G24 – field (vocalInput)

$\forall w, x \in T_V: w = \text{field} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1)$
 $\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(T_i)$
 $\forall w, x \in T_V: w = \text{field} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$
 $\forall w, x \in T_V: w = \text{field} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idprompt})$
 $\forall w, x \in T_V: w = \text{field} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{bargain} \neq \text{NULL} \wedge x.\text{bargain} \neq \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"isInterruptible"}, \text{"true"})$
 $\forall w, x \in T_V: w = \text{field} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, x.\text{textnode})$

$\forall x \in T_V: x = \text{field} \rightarrow \text{Addnode}(\text{"vocalInput"}, \text{idfield}) \text{ where } \text{idfield} = \text{NodeAmount}(T_i)$
 $\forall x \in T_V: x = \text{field} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"id"}, \text{idfield})$
 $\forall x \in T_V: x = \text{field} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"name"}, x.\text{name})$
 $\forall x \in T_V: x = \text{field} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"name"}, \text{idfield})$
 $\forall x \in T_V: x = \text{field} \wedge x.\text{type} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"grammar"}, x.\text{type})$
 $\forall x \in T_V: x = \text{field} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"currentValue"}, x.\text{expr})$
 $\forall x \in T_V: x = \text{field} \wedge x.\text{type} = \text{NULL}, \exists y, z \in T_V: y = \text{NearestInPath}(\text{form}, x) \wedge z = \text{grammar} \wedge z \in \text{childNodes}(y) \wedge z.\text{textnode} \neq \text{NULL} \wedge \text{grammar} \notin \text{childNodes}(x) \rightarrow \text{AddAttribute}(\text{idfield}, \text{"grammar"}, z.\text{textnode}) \wedge \text{AddAttribute}(\text{idfield}, \text{"isOrderIndependent"}, \text{"true"})$
 $\forall x \in T_V: x = \text{field} \wedge x.\text{type} = \text{NULL}, \exists y, z \in T_V: y = \text{NearestInPath}(\text{form}, x) \wedge z = \text{grammar} \wedge z \in \text{childNodes}(y) \wedge z.\text{src} \neq \text{NULL} \wedge \text{grammar} \notin \text{childNodes}(x) \rightarrow \text{AddAttribute}(\text{idfield}, \text{"grammar"}, x.\text{src}) \wedge \text{AddAttribute}(\text{idfield}, \text{"isOrderIndependent"}, \text{"true"})$

Chapter 7 Reverse engineering of other markup-based UI

The set of rules G34 transforms an `audio` element into a `vocalPrompt`. The `audio` element is used for two purposes, either play an audio file (such as `.wav`) or to render the value of variable. In the first case, the `src` attribute is used to set the `defaultContent` of the `vocalPrompt`, in the second case the value (a variable's name) of the `expr` attribute is used to set the `defaultContent`.

G34 – audio (vocalPrompt)

$\forall x \in T_V : x = \text{audio} \rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idaudio}) \text{ where } \text{idaudio} = \text{NodeAmount}(T_t)$ $\forall x \in T_V : x = \text{audio} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"id"}, \text{idaudio})$ $\forall x \in T_V : x = \text{audio} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idaudio})$ $\forall x \in T_V : x = \text{audio} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"defaultContent"}, x.\text{expr})$ $\forall x \in T_V : x = \text{audio} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"defaultContent"}, x.\text{src})$ $\forall x, p \in T_V : x = \text{audio} \wedge p = \text{prompt} \wedge p.\text{bargain} = \text{false} \wedge \text{IsInPath}(p, x) = \text{true} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isInterruptible"}, \text{"false"})$

In this last set, the `link` element is transformed into a `vocalNavigation` element. The `isBridgeable` attribute is set to `false` directly as when such type of `link` is triggered, the source document becomes inactive. If a `dtmf` value is specified (`dtmf` is the keyboard's number of the phone that will activate this element) in the source tree, the `keyboardShortcut` takes that value. Finally, if the `next` attribute has a value, an `auditoryTransition` is added from this element to the target of the `next` attribute (see chapter 4 for the definition of `AddAudiTr` which is a group of `AddNode` operations).

G33 – link (vocalNavigation)

$\forall x \in T_V : x = \text{link} \rightarrow \text{Addnode}(\text{"vocalNavigation"}, \text{idlink}) \text{ where } \text{idlink} = \sum \text{node} \in T_t$ $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{"id"}, \text{idlink})$ $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{"name"}, \text{idlink})$ $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{"NavigationType"}, \text{"link"})$ $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{"isBridgeable"}, \text{"false"})$ $\forall x \in T_V : x = \text{link} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idlink}, \text{"currentValue"}, x.\text{expr})$ $\forall x \in T_V : x = \text{link} \wedge x.\text{dtmf} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idlink}, \text{"keyboardShortcut"}, x.\text{dtmf})$ $\forall x \in T_V : x = \text{link} \wedge x.\text{next} \neq \text{NULL} \rightarrow \text{AddAudiTr}(\text{idlink}, x.\text{next})$

7.2.4 Example

Two examples of reverse engineering of VoiceXML files to UsiXML at the CUI level are given in appendix H-3.

7.3 Conclusion

The analysis of the two languages addressed almost all the reverse engineering sub-problems identified in section 4.6. Indeed, the elements, attributes, and hierarchy detection categories are covered totally by both approaches. The layout for WML and sequential ordering for VoiceXML are also reverse engineered completely. Multiple tree transformations have also been developed to address the problem of UI distributed on multiple files. The dialog detection problem is partially achieved, as only the navigation between UI has been taken into account. Finally the category of retargeting operations has not been covered for the two languages.

This chapter demonstrated that it is not only feasible to address the UI reverse engineering problem for other platforms equipped with other UI languages (we selected two cases with different platforms and different interaction modalities), but several fundamental aspects of previously introduced concepts remain valid and applicable, in particular the reference framework, the notation for expressing the derivation rules, the user interface description language (here, UsiXML), and the process based on families of sub-problems which have been found similar to previous ones to some extent. Identifying these stable aspects is an important conclusion if one desires to address a particular UI reverse engineering which has not yet been addressed before, but which could be considered as comparable in terms of the characteristics of the source language (said to be declarative).

If a new UI declarative language emerges which could be assimilated to those which have been considered so far, it is expected that these fundamental aspects will remain stable over time as they will be still applicable.

Chapter 8 Reverse engineering of resource files

This eighth chapter is on the reverse engineering of another type of declarative UI, Windows resource files. This chapter expands the scope of the proposed method, by adding the reverse engineering of a non-markup language. However, Windows resource file can be considered as declarative UI specifications, but the manner to describe UIs is quite different from markup languages style. This study is based on the bachelor thesis of Julien Marion [Mari05] who has conducted his work under our supervision during the academic year 2004-2005. The student followed the method elaborated in the previous chapters in order to achieve this reverse engineering study. As for the previous chapter, this study is decomposed into four parts: the first section contains a brief description of the language, the second section defines the working hypothesis, the third is a description of representative derivation rules and a meta-model of the language and the last one concludes this chapter. An example of reverse engineering is also available in appendixes.

8.1 Windows resources files

Resources files are additional binary data of Windows applications which can be stored directly in an application (.exe files) or in libraries (.dll files). They contain a wide range of elements, including elements necessary for the UI. Resources can be easily recovered thanks to decompilation programs (such as ResourceHacker [Resh02] or Restorator [Rest06]) in order to apply quick maintenance or minor

changes. A Windows resource file contains several different components that are not directly needed in the functional core of an application, such as accelerator tables, bitmaps, cursors, menus, dialog boxes, icons, string tables and version information... Resource files (.res) are compiled resource script files (.rc) which are a textual specification of the resources.

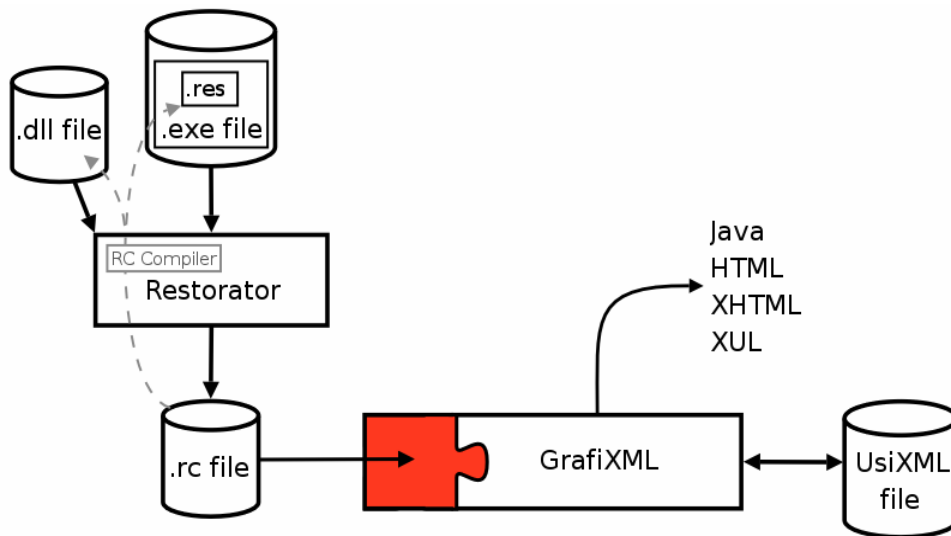
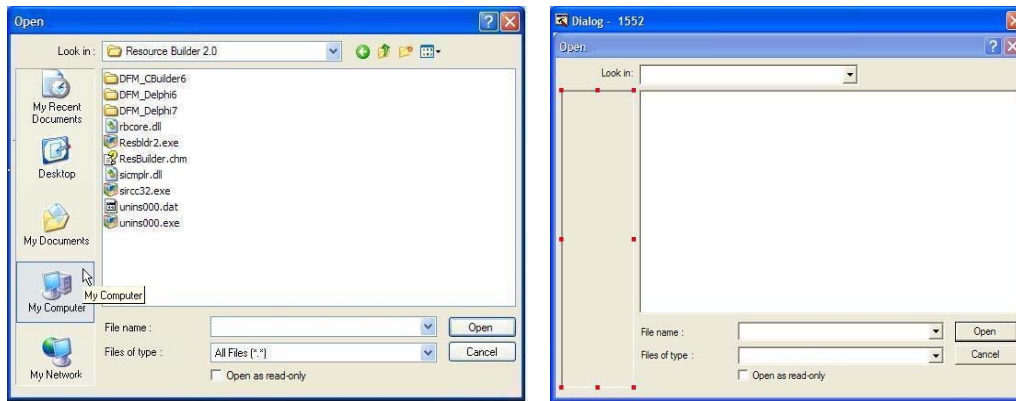


Figure 8-1 The envisioned reengineering process

The envisioned reengineering process starts by extracting the resource script file from a .dll or .exe file thanks to a decompiler (for example Restorator on figure 8-1). Once the .rc files have been extracted and saved to a local directory, the reverse engineering process can be launched. The process can be divided into two steps: the scan and the process phase. In the scan phase, the relevant and available resource scripts are detected, i.e. only resources in relation to the UI. Then the relevant .rc files are selected to be analyzed and the reverse engineering can be launched by applying derivation rules described in section 8.3. After the reverse engineering, the CUI model expressed in UsiXML can be edited in GrafiXML and then used to generate new UI code in HTML, XHTML, XUL or Java.

The portion of interest of resource scripts is reduced to dialog and menu components, as other elements are graphical components not useful for the reverse engineering of an abstract specification. An example of resource script is shown on fig 8-2.

Chapter 8 Reverse engineering of resource files



```
1552 DIALOGEX 0, 0, 370, 237
STYLE DS_MODALFRAME | DS_CONTEXTHELP | WS_POPUP | WS_VISIBLE | WS_CLIPCHILDREN | WS_CAPTION | WS_SYSMENU
CAPTION "Open"
LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
FONT 8, "MS Shell Dlg"
{ CONTROL "Look &in :", 1091, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 4, 7, 57, 8,
0x00001000
CONTROL "", 1137, COMBOBOX, CBS_DROPDOWNLIST|CBS_OWNERDRAWFIXED|CBS_HASSTRINGS|WS_CHILD|
WS_VISIBLE|WS_VSCROLL |WS_TABSTOP, 66, 4, 174, 300
CONTROL "", 1088, STATIC, SS_LEFT | WS_CHILD, 248, 4, 80, 14
* CONTROL "", 1184, "ToolBarWindow32", 0x50012B4C, 4, 22, 58, 208, 0x00000200
CONTROL "", 1120, LISTBOX, LBS_NOTIFY | LBS_NOINTEGRALHEIGHT | LBS_MULTICOLUMN | WS_CHILD | WS_BORDER |
WS_HSCROLL, 66, 22, 300, 156
CONTROL "File &name :", 1090, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 67, 187, 71, 8
CONTROL "", 1152, EDIT, ES_LEFT | ES_AUTOHSCROLL | WS_CHILD | WS_VISIBLE | WS_BORDER | WS_TABSTOP, 144,
184, 164, 12
CONTROL "", 1148, "ComboBoxEx32", 0x50210042, 144, 184, 164, 150
CONTROL "Files of &type :", 1089, STATIC, SS_LEFT | SS_NOTIFY | WS_CHILD | WS_VISIBLE | WS_GROUP, 67, 203, 71, 8
CONTROL "", 1136, COMBOBOX, CBS_DROPDOWNLIST | WS_CHILD | WS_VISIBLE | WS_VSCROLL | WS_TABSTOP, 144,
201, 164, 100
CONTROL "Open as &read-only", 1040, BUTTON, BS_AUTOCHECKBOX | WS_CHILD | WS_VISIBLE | WS_TABSTOP, 144, 217,
160, 8
CONTROL "&Open", 1, BUTTON, BS_DEFPUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 316, 184,
50, 14
CONTROL "Cancel", 2, BUTTON, BS_PUSHBUTTON | WS_CHILD | WS_VISIBLE | WS_GROUP | WS_TABSTOP, 316, 200, 50,
14 }
```

Figure 8-2 Resource script of a dialog box

The upper left part represents the dialog box that will be decompiled, the upper right part represents the view of this dialog box in a decompilation tool and the bottom part is the textual specification for this dialog box.

The reverse engineering of UI abstract specifications based on resource files can be characterized by five major difficulties:

1. Resource script files do not always contain the entirety of the attributes of interaction objects. Some attributes are defined at runtime and it is impossible to recover this information in a static analysis of the resource script. Moreover, values of some attributes are sometimes modified before the first display of a UI, and the specified values are never used during the execution of the application. Thus the produced model is a wrong specification of what is really displayed to the user.

Chapter 8 Reverse engineering of resource files

Again, it is impossible to detect those attributes, and the reverse engineering will only be based on the known/available information.

2. Some values are also modified dynamically (e.g. change of color) in the main program's code, but as these are specified in the functional core, these aspects can not be reverse engineered and are simply ignored.

3. The use of shortcut notations in the specifications, where some attributes or objects are combination of other "basic" attributes/objects. This implies knowledge of all the attributes or objects defined by these "super-elements" and a constitution of a "dictionary" of shortcut notations. An example of this is `DEFPUSHBUTTON"&Find Next",1,205,5,65,14,WS_GROUP` which is equivalent to `CONTROL"&Find Next",1,BUTTON,BS_DEFPUSHBUTTON|WS_CHILD|WS_VISIBLE| WS_GROUP|WS_TABSTOP,205,5,65,14`. As shown, the DEFPUSHBUTTON possesses 4 default attributes compared to the classical button

4. Resource files also allow using hexadecimal shortcut notation. A hexadecimal number is specified to represent a unique combination of attributes. For example, on figure 8-1 in the row of the fourth control, the number 0x50012B4C is equivalent to the styles `WS_CHILD`, `WS_VISIBLE`, `WS_TABSTOP`, `CCS_NOESIZE`, `CCS_NOPARENTALIGN`, `CCS_NODIVIDER`, `TBSTYLE_TOOLTIPS`, `TBSTYLE_WRAPABLE`, `TBSTYLE_CUSTOMERASE` and `TBSTYLE_FLAT`, and the number 200 is equivalent to the extended style `WS_EX_CLIENTEDGE`. For the extended combo box (row of 8th control), the number 0x50210042 is equivalent to `WS_CHILD`, `WS_VISIBLE`, `WS_VSCROLL`, `WS_TABSTOP`, `CBS_DROPDOWN` and `CBS_AUTOHSCROLL`.

This other type of shortcut notation implies decoding the hexadecimal number to find out which attributes are defined thanks to this number.

5. The layout of components is described thanks to absolute positions and dimensions. Moreover, these positions are not described thanks to pixels, but are based on a measure proportional to characters size. They can be converted into pixel thanks to an approximation (see section 8.3). Another difficulty of these absolute positions is that elements are often overlapping, for example a label having a width equal to window's width could be specified, accompanied by an overlapping checkbox on the same vertical position but with different horizontal position. A final difficulty is that elements dimensions are not always displayed according to the specified size, e.g. size of drop down list boxes is equal to the size of displayed items when the list is "opened" during execution. All these factors imply a very complex solution to divide a Windows UI into boxes.

These difficulties can be categorized according to the reverse engineering dimensions presented in section 4.6. The difficulties number 1, 2, 3 and 4 are classified in the

attribute detection problem. But difficulty n°2 and 3 can also be categorized into two other dimensions, n°2 in the dialog recovery problem and n°3 for the element detection issue. The fifth difficulty belongs to the layout recovery aspect, and also to the hierarchy detection problem, as position of elements have to be analyzed to detect if an element is embedded into another.

8.2 Working hypothesis

The abstraction level chosen for this reverse engineering is the concrete UI model recovered thanks to a static analysis. However, the method presented here could be easily extended to the abstract UI model. The reverse engineering is only based on resource files, i.e. no other section of code of the application is analyzed. The scope of the analysis is thus limited to the presentational aspects of resource files and the dialog is thus not analyzed, as the transitions between dialog boxes and transformations or modifications of the UI are managed in the functional core. Only transitions starting from menus could be reverse engineered, but it has been chosen to ignore the dialog completely.

The reverse engineering of the layout of the UI is also not recovered. Coordinates of UI elements are given in absolute positions and its reverse engineering would require a complex algorithm to recover the box composition of an UI. This algorithm should compare vertical and horizontal positions, calculate the area of objects, and then group them by using a smallest-distance criterion. However, there would be no guarantee that these results would be correct, and a human analysis/correction would be required after the reverse engineering process (see section 8.1 about layout detection problems).

Losses due to the reverse engineering

In addition to the dialog and layout that are not analyzed, some other type of information is also lost during the reverse engineering process. The entire UI is reverse engineered without colors, as colors in Windows resource files are often defined at the system level (i.e. the user defines the colors of fonts, backgrounds etc...) for all the dialog boxes used in the system.

Another type of losses is that Windows UI allows to be defined very precisely, as almost every graphical aspect can be customized, and this implies a high number of attributes. Obviously, all these attributes are not mapped in UsiXML, and therefore several graphical details are lost during the reverse engineering process (for example 3D style – sunk edge buttons).

8.3 Derivation rules and resource meta-model

Following the method developed in previous chapters, a meta-model has been defined to represent resource files structure in a UML class diagram. This meta-model is in fact a model based on a logical representation of resource files, as names and attributes of resource files can be relatively cryptic (e.g. WS_EX_DLGMODAL FRAME has been replaced by dialogModalFrame in the window entity).

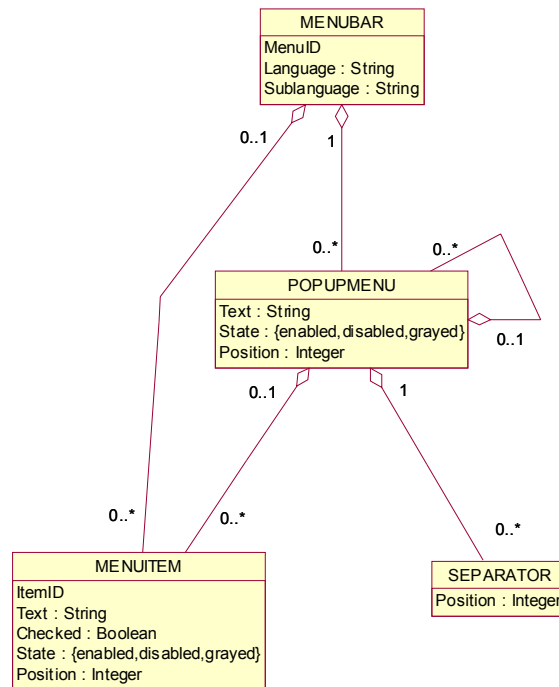


Figure 8-3 Menu meta-model

Another advantage of using this intermediary step is that attributes of elements have a significance following their position in the code, e.g. the five first integer numbers after the name (or last numbers of the line) define the id, vertical position, horizontal position, width and height of the object. In the logical representation, these values are copied into named attributes, so that they can be used in the derivations rules. It also allows converting hexadecimal numbers (see section 8.1) in attribute names. Using this intermediate representation facilitates and clarifies the reading of a resource file. The two meta-models [Mari05] representing menus (fig. 8-3) and windows (appendix B) are thus based on this logical representation.

Chapter 8 Reverse engineering of resource files

The first model about menus contains only four types of elements: a **menubar** containing the menu specification, **popupmenus**, menu items and **separators**. The subject of the second model (appendix B) is the window/dialog box meta-model representation. This model contains almost all the “classic” interaction that we can find in windows application, such as button, combo boxes, check boxes, labels etc...

Derivation rules were designed to represent transformations from a tree structure to another. Functions and basic operation are specifically expressed to accommodate such type of structure. But, resource files code is not represented in a tree as for markup languages and some transformation have to be made before the application of derivation rules. Figure 8-4 represents a typical menu and its corresponding resource script.

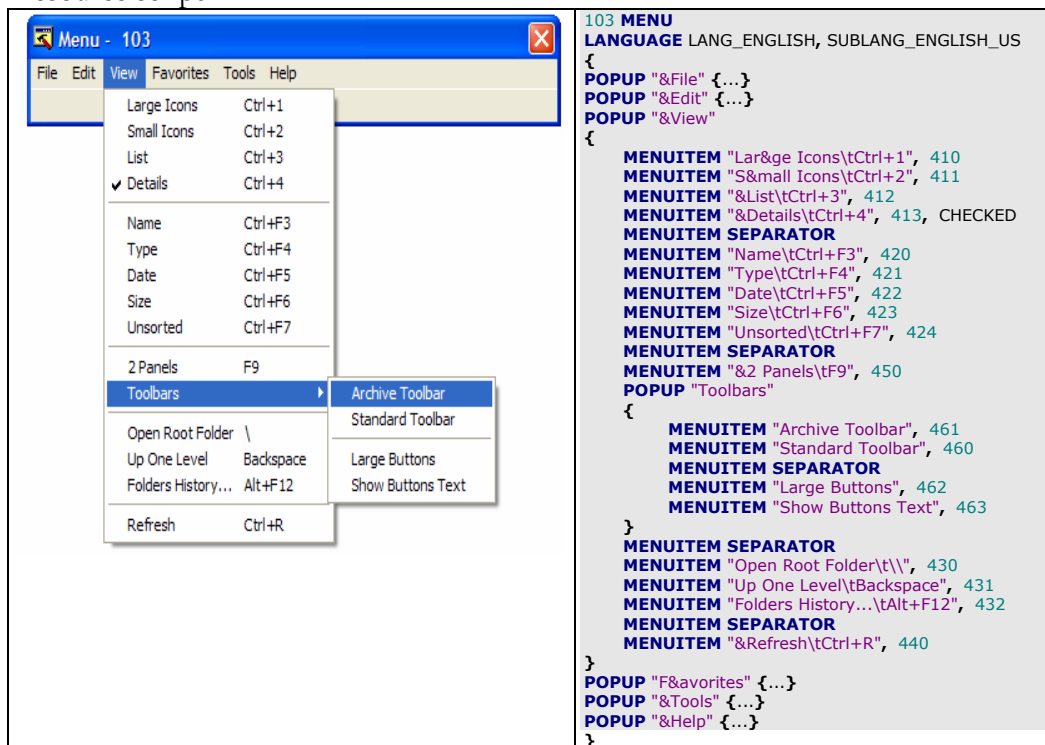


Figure 8-4 Example of a menu in a resource script

As one can see, a hierarchical structure can still be derived from the code. The containment relation is represented thanks to brackets ({}) or the “begin” and “end” keywords instead of an opening and closing markup as for HTML, WML and VoiceXML. The containment between two elements in a UI is not always shown in

this manner (see section 8.1), but it can already give information about some containment relations. Each line of code represents a node, and also an element of the UI, that can be the subject of the derivations rules. An exception to this rule is that entities following a **menu** or a **dialog** (such as the language on fig. 8-4) have to be considered as attributes of the **menu** or **dialog**. Once these transformations have been applied, the derivation rules can be used as in the previous chapters.

The entirety of the derivation rules has not been specified in this thesis, but this section shows the feasibility of the specification of derivation rules for resource files. The remainder of this section contains several examples of rules.

```

∀ x ∈ TR: x = DIALOG → Addnode (“DialogBox”, iddgbox) where iddgbox = NodeAmount(Tt)
∀ x ∈ TR: x = DIALOG → AddAttribute (iddgbox, “id”, iddgbox)
∀ x ∈ TR: x = DIALOG ∧ x.dlgId !=NULL → AddAttribute (iddgbox, “name”, x.dlgId)
∀ x ∈ TR: x = DIALOG ∧ x.dlgId =NULL → AddAttribute (iddgbox, “name”, iddgbox)
∀ x ∈ TR: x = DIALOG → AddAttribute (iddgbox, “isVisible”, “true”)
∀ x ∈ TR: x = DIALOG ∧ x.disabled !=true → AddAttribute (iddgbox, “isEnabled”, “true”)
∀ x ∈ TR: x = DIALOG ∧ x.disabled =true → AddAttribute (iddgbox, “isEnabled”, “false”)
∀ x ∈ TR: x = DIALOG ∧ x.height !=NULL ∧ x.fontSize !=NULL
→ AddAttribute (iddgbox, “width”, ((x.width x 3304)/(275 x x.fontSize) + 10))
∀ x ∈ TR: x = DIALOG ∧ x.width !=NULL ∧ x.fontSize !=NULL
→ AddAttribute (iddgbox, “height”, ((x.height x 274)/(21 x x.fontSize) + 35))
∀ x ∈ TR: x = DIALOG ∧ x.caption =true → AddAttribute (iddgbox, “defaultContent”, x.caption)
∀ x ∈ TR: x = DIALOG ∧ x.topMost =true → AddAttribute (iddgbox, “isAlwaysOnTop”, x.topMost)
∀ x ∈ TR: x = DIALOG
→ ConstrBox (BoxId, “vertical”) ∧ AddArc(iddgbox, BoxId) where BoxId = NodeAmount(Tt)

```

This first example derives a **dialog** element into a **dialogBox** in the target tree. If the **disabled** attribute is set to false, the **isEnabled** attribute is set to true in the target tree. As sizes of dialog boxes in Windows is proportional to the width and height attribute and the font size, a relatively complex operation is done to recover the size in pixels (e.g. height is calculated by the following formula $(274 \times \text{height} / 21 \times \text{fontsize}) + 35$, which is the unit used in the UsiXML model. The **caption** and **topmost** attributes define the **defaultContent** and **isAlwaysOnTop** attributes respectively in the UsiXML tree. Finally, a vertical **box** is created and appended to the **dialogBox** node in the target tree. This box will contain the rest of the UI (as for rule set G1 for windows in HTML/WML).


```

∀ x ∈ TR: x = COMBOBOX ∧ x.type != "simple"
→ Addnode ("ComboBox", idcombox) where idcombox = NodeAmount(Tt)
∀ x ∈ TR: x = COMBOBOX ∧ x.type != "simple" → AddAttribute (idcombox, "id", idcombox)
∀ x ∈ TR: x = COMBOBOX ∧ x.CtrlId != NULL ∧ x.type != "simple"
→ AddAttribute (idcombox, "name", x.CtrlId)
∀ x ∈ TR: x = COMBOBOX ∧ x.CtrlId = NULL ∧ x.type != "simple"
→ AddAttribute (idcombox, "name", idcombox)
∀ x ∈ TR: x = COMBOBOX ∧ x.visible=true ∧ x.type != "simple"
→ AddAttribute (idcombox, "isVisible", "true")
∀ x ∈ TR: x = COMBOBOX ∧ x.visible=false ∧ x.type != "simple"
→ AddAttribute (idcombox, "isVisible", "false")
∀ x ∈ TR: x = COMBOBOX ∧ x.disabled != true ∧ x.type != "simple"
→ AddAttribute (idcombox, "isEnabled", "true")
∀ x ∈ TR: x = COMBOBOX ∧ x.disabled = true ∧ x.type != "simple"
→ AddAttribute (idcombox, "isEnabled", "false")
∀ x ∈ TR: x = COMBOBOX ∧ parentNode(x).height != NULL ∧ parentNode(x).fontSize != NULL ∧
x.type != "simple" → AddAttribute (idcombox, "maxLineVisible", parentNode(x).height / ((11x
parentNode(x).FontSize/8)-1))
∀ x ∈ TR: x = COMBOBOX ∧ x.type = "dropdown"
→ AddAttribute (idcombox, "isEditable", "true")
∀ x ∈ TR: x = COMBOBOX ∧ x.type = "dropDownList"
→ AddAttribute (idcombox, "isEditable", "false")

```

The second example is about the derivation of `combobox` into a `comboBox` element in the CUI model. If the `type` attribute of the `combobox` is not `simple`, the derivation is applied. Otherwise, a derivation rule of the `comboBox` into a `listBox` is processed. Following the value of the `type` attribute, either the `isEditable` is set to `true` if the `type` equals `dropdown` or set to `false` if it equals `dropdownlist`. If the `disabled` attribute is set to `true` in the source tree, then the `isEnabled` attribute is set to `false` in the CUI model. As for the previous example, the number of visible lines (`maxLineVisible`) is defined following the `fontsize` and the `height` of the `dialog` containing the `combobox`. If the `combobox` has the `type` attribute different from `simple`, then the number of visible lines is calculated by dividing `height` of the `dialogbox` by `x` where `x` is equal to `11/8` multiplied by the `fontsize` (defined in the `dialogbox` element) and by subtracting 1 to this result

```

∀ x ∈ TR: x = RADIOBUTTON ∧ x.pushLike = false ∧ x.textnode != NULL
→ ConstrBox (BoxId, "horizontal") where BoxId = NodeAmount(Tt)
∀ x ∈ TR: x = RADIOBUTTON ∧ x.pushLike = false
→ Addnode ("radioButton", radioid) where radioid = NodeAmount(Tt) ∧ AddArc (BoxId, radioid)
∀ x ∈ TR: x = RADIOBUTTON ∧ x.pushLike = false → AddAttribute (radioid, "id", radioid)
∀ x ∈ TR: x = RADIOBUTTON ∧ x.pushLike = false ∧ x.CtrlId != NULL
→ AddAttribute (radioid, "name", CtrlId)

```

Chapter 8 Reverse engineering of resource files

$\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge x.\text{CtrlId} = \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{radioid}, \text{"name"}, \text{radioid})$ $\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge \text{withoutAmper}(x.\text{text}) \neq x.\text{text}$ $\rightarrow \text{AddAttribute}(\text{radioid}, \text{"defaultMnemonic"}, \text{CharAfterAmper}(x.\text{text}))$ $\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge x.\text{Visible} = \text{true}$ $\rightarrow \text{AddAttribute}(\text{radioid}, \text{"isVisible"}, \text{"true"})$ $\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge x.\text{Visible} = \text{false}$ $\rightarrow \text{AddAttribute}(\text{radioid}, \text{"isVisible"}, \text{"false"})$ $\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge x.\text{disabled} \neq \text{true}$ $\rightarrow \text{AddAttribute}(\text{radioid}, \text{"isEnabled"}, \text{"true"})$ $\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge x.\text{disabled} = \text{true}$ $\rightarrow \text{AddAttribute}(\text{radioid}, \text{"isEnabled"}, \text{"false"})$ $\forall x \in T_R: x = \text{RADIOBUTTON} \wedge x.\text{pushLike} = \text{false} \wedge x.\text{textnode} \neq \text{NULL}$ $\rightarrow \text{Addnode}(\text{"textComponent"}, \text{labelid}) \text{ where } \text{labelid} = \text{NodeAmount}(T) \wedge \text{AddAttribute}(\text{labelid}, \text{"isEnabled"}, \text{"true"}) \wedge \text{AddAttribute}(\text{labelid}, \text{"isVisible"}, \text{"true"}) \wedge \text{AddAttribute}(\text{labelid}, \text{"defaultContent"}, \text{withoutAmper}(x.\text{textnode})) \wedge \text{AddArc}(\text{BoxId}, \text{labelid})$

This last example shows the rules needed to derive `radioButtons`. If the element is a `radioButton` with its `pushlike` attribute set to `false`, then a horizontal `box` and a `radioButton` is created with their `id` attribute set to a computed value. The `box` is created to put the `radiobutton` and its descriptive label together. The `name` attribute in the target tree takes the value of the `CtrlId` from the source tree (`CtrlId` corresponds to the first integer of the line).

The next rule uses a string parsing method, as `CharAfterAmper()` returns the character following an ampersand symbol in a string. If the result of this function is not null, the `defaultMnemonic` attribute in the CUI model takes the value of its return. Finally, following the value of the source's attribute `disabled` and `visible`, the values of the `isEnabled` and `isVisible` attributes from the target tree are defined thanks the four last rules. Input elements are often specified with a descriptive label, which is derived into a separate `textComponent`. The `defaultContent` attribute of the `textComponent` is specified by applying the `withoutAmp()` function to the `textnode` of the `radioButton`. This function is a string parsing method allowing removing ampersands from a given text. Other attributes of the `textComponent` are generated automatically. Finally an arc is added between the parent horizontal `box` and the `textComponent`.

A case study for the reverse engineering of Windows resource file is given in appendix H-4.

8.4 Conclusion

By widening the scope of the approach to non-markup languages, this chapter demonstrated the feasibility of the reverse engineering of Windows resource files according to the method presented in this thesis. The notation for derivation rules is still valid for this kind of UI language, after some light modification of the source code in order to obtain a coherent tree structure. Other concepts such as the reference framework and the UsiXML language also remain valid in this study.

The different subcategories of the reverse engineering problem (see section 4.6) are identical to markup languages. However, some of them are not solved at the moment: elements and attributes are covered in the proposed approach, but not the hierarchy and layout detection, as the solution for these categories would require a very complex algorithm out of scope of this research. Dialog recuperation is not covered following the working hypotheses. The multiple trees and retargeting operations categories have not been completed in this approach.

In conclusion, the research focused on the analysis of a new type of declarative UI languages and proved the applicability of the approach to Windows resources scripts. This process could also be generalized to other types of declarative UI. Similar languages (e.g. Delphi or Visual Basic forms) could thus be analyzed according to the current approach, as the basic concepts elaborated in this thesis would remain the same.

The reverse engineering of Windows resources files allowed us to highlight some limitations of a static analysis. The quality of this type of reverse engineering is dependent on the source file: if the source file is rich in information, the resulting model will be of a high quality, but this technique does not allow obtaining a more expressive output than the input of the process.

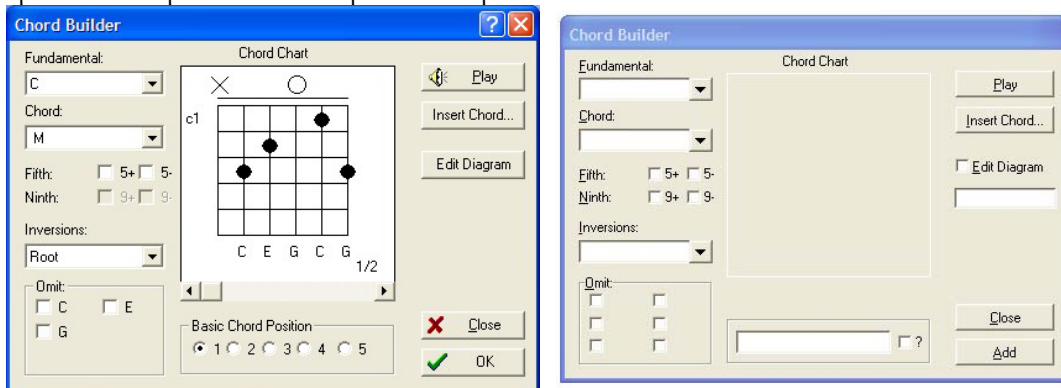


Figure 8-5 Limitation of a static analysis

Chapter 8 Reverse engineering of resource files

As stated in section 8.1, some parts of the UI can be described in the functional core, and thus a static analysis is not able to reach these specifications. Figure 8-5 illustrates graphically this limitation: the original UI as presented to the user is shown on the left hand side and the UI as recovered by a resource decompiler on the right hand side. This second UI is the input of the reverse engineering process. As one can see, some incoherence exists with the original UI in this decompiled resource file: elements are not present in the resource, or specified as another type, labels of buttons are not identical, some elements are specified but are never presented to the user etc... This shows clearly a shortcoming of this approach. This could be overcome thanks to a dynamic analysis - such as in visual TAP (see chapter 3) which analyzes the video signal and thus the UI as displayed to the user- that would complete or correct the UI specification at run-time.

Chapter 9 Validation

This chapter contains the validation of the proposed approach, i.e. concepts, method and tools. This chapter is divided into four parts, the first about the internal validation (theoretical validation), the second for the external validation (empirical validation) and the third dedicated to a comparison with the approaches of the state of the art (chapter 3) and more particularly with ADAPT, a transcoding approach. The last section concludes this chapter. The objective of this chapter is to validate the step of conceptualization and operationalization to identify what material in the process could be generalized [Long96].

The scope of the validation is thus the complete approach, which can be decomposed into three subjects:

- for the conceptual point of view: the CUI and AUI models recovered. The validation has been done for both levels, AUI and CUI, but the CUI level is mostly analyzed and this for two reasons: first it is the most frequently used and useful level for a reverse engineering, and secondly, due to its structure and complexity, the CUI level is more interesting to analyze than the AUI level which is a very coarse description of the UI.
- for the method: the combination of the flexible reverse engineering and forward engineering to achieve a complete reengineering. This point can be subdivided into two subjects, the flexibility of the reverse engineering, by using

different reverse engineering configuration and the complete reengineering by reusing models produced thanks to the approach with different forward engineering tools.

- for the tool implementing the approach: ReversiXML and the UsiXML language. The implementation of ReversiXML was paused on version 1.1 and this version of the tool was based on UsiXML 1.4.6. Therefore some differences exist with the current versions of AUI and CUI models (the most recent published version of UsiXML is 1.6.4).

9.1 Internal validation

This section addresses the internal validation (i.e. theoretical validation of the approach) with respect to four criteria: the coverage, coherence, correction and performance, each criteria being respectively addressed in the next subsections. These criteria have been chosen to evaluate the objectives of generality, predictability, flexibility and controllability for the approach presented in this thesis (see section 9.1.5).

9.1.1 Coverage of source language

This section analyses the coverage - of the HTML 4.0 language by the approach and tool described in this thesis. Coverage can be divided into two subjects, the first about the coverage of tags/elements, the second about the coverage of concepts.

Elements coverage. The language is composed of 95 different tags and most of them are recognized and reverse engineered. But some of them are not analyzed, and the list of uncovered tags is given below:

- **Hr** (displays a horizontal rule) is not recovered as there is no element in UsiXML 1.4.6 to define separators in the UI. This observation led to a modification of the language, and separators are now part of the version 1.6.4 of UsiXML (published in March 2006).
- **Isindex** (displays a textbox) is not recovered as the tag is deprecated (W3C recommend the usage of `input` instead of this tag, however, it is still specified in the HTML 4.0 DTD to insure backward compatibility).
- **Col** and **colgroup** (defines properties of columns) is not reverse engineered. They usually contain information relative to the size of a column, the alignment of its content, etc. They are not reverse engineered because the benefits of this would be marginal compared to the effort to recover this information. Indeed, it would be

Chapter 9 Validation

required to make an abstract representation of the table to find out which cells are targeted by these two nodes, as **colspan** attributes may merge several columns and thus change the number of cells per row. Moreover, these tags are rarely used (it was not found in the 29 pages reverse engineered in this chapter) and the information in these nodes is not of first importance in the UI specification (i.e. it does not change composition or behaviour of the UI).

- **Script** (element containing code that not belongs to the HTML language) is ignored following the working hypothesis (see section 6.2).

- **Thead**, **tbody** and **tfoot** (defines part of a table) are not recovered as is it impossible to differentiate cells in UsiXML. This observation also led to a recommendation for the next version of the language. Moreover, there is no visual distinction in browsers for these three tags and thus the fact to ignore these nodes does not change anything.

- **Link**, **meta**, **!Doctype** are not analyzed as these tags do not represent part of the UI. **Link** defines the semantic relation between a document and another, such as a copyright, glossary or chapter link, and is not recovered as this kind of semantic information can not be recorded in UsiXML CUI or AUI models, **meta** contains information about the Web page such as keyword, author etc and **!doctype** defines the version of the DTD used in the current file.

- **Noframes** and **noscripts** (defines the error message to be displayed in browsers that are not able to process frames or scripts) are not recovered, as this conditional behaviour can not be expressed easily in UsiXML. It could be specified in UsiXML, but conditions are extremely complex and verbose in the language, and are thus not generated as is it would lengthen models significantly for minor benefits.

There can be three main reasons for ignoring an HTML element: either they can not be represented in a UsiXML model (there is not correspondent category in the CUI or AUI model) or it has been declared in the working hypothesis that this type of element will not be analyzed in this research. Another reason is that the tag does not represent a part of the UI. The only tags that do not belong to these categories are the **col** and **colgroups** tags for which the cost of reverse engineering them is extremely high compared to the benefits of the information recovered. 13 elements out of 95 are not analyzed in the approach, thus the coverage is of 86.3 % of the totality of tags. After the subtraction the tags not representing elements of the UI from this amount, having no corresponding representation in UsiXML CUI and AUI models, or not modifying the UI presentation and behaviour, we fall down to 6 elements, and thus the coverage of the language increases to 93.2% of the HTML language. Moreover, the portion of the language effectively reverse engineered represents the

Chapter 9 Validation

vast majority of tags used in HTML code, as these elements are the most important, frequent, and useful components of the language. If we remove the script element from the list, more than 99% of the content of the HTML code analyzed in the various investigations of this chapter is reverse engineered.

Coverage of conceptual design. Coverage of the conceptual design is about the recovery of design options or design decisions that have been implemented in the HTML code during the creation of the Website. The recovery of these aspects is impossible without any human intervention, as an automated process could not resolve ambiguities inherent to this kind of problem, and would be hard to cover in an assisted way. Some researches tried to reverse engineer the conceptual design in an assisted way, such as in the Pima project (see chapter 3). In this approach, layout and heuristics are used to deduce semantic structure of UI, and it is thus possible to recover design decision thanks to this approach. As this type of reverse engineering is not an objective of our approach, it has not been achieved in this thesis.

9.1.2 Coherence

This section is about the coherence of the application of the rules. Coherence is defined by a logical, orderly and consistent application of rules, i.e. that rules produces uniform results. In the proposed approach, a rule applied for one type of element of the source file will always produce the same output in the target model (with a fixed set of reverse engineering options). Conversely, one element belonging to the target model is always the results from applying one and only one derivation rule on one precise type of element of the source file. In some cases, one element of the UsiXML specification can be the result of several different tags (e.g. buttons are generated in the CUI model when a **button** tag or when an **input** node with its **type** attribute set to **button** is detected), but in this case the possible source elements belong always to a well defined domain of derivation rules. As an example, the application of rules for checkboxes is shown below, by applying the rule set G13:

`<input type="checkbox" name="cb1" disabled>`

is derived thanks to the derivation rules of G13

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{checkbox} \rightarrow \text{Addnode}(\text{"checkbox"}, \text{idcheck})$ where $\text{idcheck} = \Sigma \text{node} \in T_t$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{checkbox} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"id"}, \text{idcheck})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{checkbox} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"name"}, \text{idcheck})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{checkbox} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"isVisible"}, \text{"true"})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{checkbox} \wedge (x.disabled = \text{false} \vee x.disabled = \text{NULL})$

Chapter 9 Validation

$$\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \wedge x.\text{checked} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"isEnabled"}, \text{"true"}) \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"defaultState"}, \text{"checked"})$$

into

```
<checkbox name="cb1" id="cb1" isVisible="true" isEnabled="false"
defaultState = "unchecked">
```

An **input** node with its **type** attribute set to **checkbox** will always produce the same output, a **checkbox** element in the UsiXML specification. Moreover, the combination of attributes will also produce the same set of attributes in the UsiXML code. Conversely, a **checkbox** element belonging to the UsiXML specification produced by ReversiXML is always the abstraction of an **input** node with its **type** attribute set to **checkbox**. This reasoning could be generalized similarly by analogy to all others rules sets implemented in ReversiXML, consequently the entire rules set could be considered as coherent since their application is coherent.

9.1.3 Performance

This section contains an analysis of the performance of the tool at the CUI level. For this experiment, the tool has been tested on 20 different Web pages of various sizes and contents. The selected pages are from different types, such as hotel booking systems (lodging.com, fhr), flight arrivals schedules (Brussels airport), conference (CHI, WCRE, LAWEB) and universities homepages (UCL, Valenciennes,...) or information (CNN, Euronews) and newspaper Websites (le soir). This investigation has been achieved by using the on-line version of the tool, located on the UCL Web server. Table 9-1 summarizes the results of the performance analysis. The first column represents the page identification and its corresponding URL can be found in appendix F. The second column contains the data relative to the original HTML page, and is itself decomposed into 2 columns, one for the size (in ko) of pages and the second for the amount of nodes. The amount of HTML nodes corresponds approximately to the number of interaction objects contained in the Web page, and is also an indicator of the length of the analysis (as size in ko is not always the best indicator, the page may contain several comments or scripts).

The second column UsiXML (1) contains data relative to the produced model. This first reverse engineering has been processed without configuration file, i.e. by capturing the maximum of information of the original HTML page. This column is subdivided into three columns, the size (in ko) of the UsiXML CUI model, the number of boxes used to describe the layout of the UI and the time required to

Chapter 9 Validation

perform this analysis (in seconds). The third column UsiXML (2) represents data about a reverse engineering applied by using a configuration file designed for mobile platforms (without images, sound, color and external components). This column is divided into two other columns, the first representing the size (in ko) of the generated model and the second containing the time needed to apply this “lightened” reverse engineering.

N°	Original Web Page		UsiXML (1)			UsiXML(2)	
	Size (ko)	number of nodes	Size (ko)	Number of boxes	Time (s)	Size (ko)	Time (s)
1	7.83	140	25.0	131	1.86	19.7	1.65
2	3.83	71	16.1	56	2.25	12.8	1.31
3	14.3	143	27.4	81	2.21	20.8	1.96
4	30.5	292	55.1	234	7.34	45.9	6.13
5	74.3	729	145.4	776	18.32	107.92	14.30
6	13.1	187	35.8	184	4.72	24.6	3.18
7	8.22	128	25.5	78	2.18	16.9	1.29
8	4.29	89	11.5	59	1.01	8.49	0.83
9	19.7	336	53.0	307	6.23	42.1	5.58
10	79.9	898	177.1	728	22.59	119.8	18.48
11	5.29	82	19.3	61	1.48	15.1	1.45
12	44.4	540	102.9	416	16.49	81.8	15.41
13	3.81	69	7.11	39	0.95	5.4	0.82
14	25.0	165	26.2	133	3.30	12.9	1.49
15	45.3	575	76.9	183	9.72	68.8	9.52
16	33.9	449	75.3	229	9.02	50.7	7.09
17	7.48	102	17.9	85	1.81	7.2	1.14
18	10.7	138	30.0	120	5.79	24.1	3.74
19	19.4	253	48.1	154	4.69	46.1	4.22
20	67.7	516	105.7	414	11.8	75.6	9.59
Av.	25.95	295.11	54.06	223.41	6.69	40.33	5.46

Table 9-1 Performance analysis

The last row of table 9-1 contains the average values for each parameter. The average size of reverse engineered Web pages is 25.95 ko and the average size of the CUI models is 54.06 ko, which is about 208 % of the size of the original HTML files. The average time to download the Web page, perform the reverse engineering, and save

the results on the server is 6.69 seconds. This time does not exceed the system response time recommended by cognitive psychology for complex tasks. This allows applying the reengineering at run-time, but could not be applied in the case of a real time reengineering. It can thus be estimated that one second is needed for 3.88 ko of HTML code. A better measure to evaluate length of pages is the number of nodes representing UI elements in the HTML code (average of 295.11 nodes per page), and in this case, ReversiXML processes 44.1 nodes/second. Note that these speed measures may be affected by several factors: time needed to generate dynamic Web pages, the load of the server on which ReversiXML is installed, connection speed with the target HTML file's server ... and therefore, two measurements will never give the same results. However, variations are relatively small (about 10% of the average time). The average number of boxes is 223.41, which is a very high number, almost one box for an HTML node, but as stated previously in section 6.4.5, heuristics have already been developed to decrease this number but they are not implemented in the tool. By adding these heuristics to ReversiXML, the total processing time would be lengthened but models size would be shortened.

For the second reverse engineering, the size and time needed to apply a lightened reverse engineering has been measured by using a configuration file designed for mobile platforms. The average size of CUI models for this category is 40.33 ko, thus 74.6 % of the size of the "full" CUI model. The average time required to perform this analysis is 5.46 seconds, which is 81.6% of the time of a normal reverse engineering. The reduction effects of these configured reverse engineering is double: firstly, it saves processing time by skipping the reverse engineering of several elements, but it also simplifies the box structure as, by ignoring these element, several boxes are left empty and are thus removed during a second parsing of the produced model, thus reducing file size.

9.1.4 Correction

Correction is the ability of the approach and tool to produce correct results compared to the foreseen output. Errors during reverse engineering can be classified in five categories which are similar to the dimensions of the reverse engineering sub-problems, as defined in section 4.6 (two categories are missing, the hierarchy detection task which has been merged in the elements and layout categories, and the mutli-trees transformations, as none was needed in this section). This empirical study has been conducted by reverse engineering five Web pages from section 9.1.3 by hand, and by comparing these results with models produced by the tool. Two of

Chapter 9 Validation

those pages were also reverse engineered a second time by using configuration files to check the difference in the success rates of a retargeting process compared to a traditional reverse engineering.

Five categories of errors have been defined for this study:

- *Elements*: establishes the proportion of UI elements effectively recovered. This is done “by hand”, by checking the correct derivation of each HTML element.
- *Graphical Transitions*: it represents the percentage of links recovered in the CUI model. Again, for each link found in the source code, it has been checked that it has been correctly copied in the target model, with the corresponding URL.
- *Box Layout*: represent the percentage of boxes correctly translated or added to the CUI model advisedly. A percentage here is certainly not a good measurement, as an incorrect box at the highest level of hierarchy would have much more impact on the UI structure than a box situated almost at the leaf level. The type and level of errors for this parameter is listed in the comments of the analysis for each page.
- *Attributes*: corresponds to the proportion of attributes *correctly derived*. It means that this proportion is based on the number of HTML attributes, not UsiXML attributes as some of these attributes are directly added to the UsiXML code without any counterpart from the HTML code (for example the two mandatory `isEnabled` and `isVisible` attributes are automatically set to true for each element).
- *Element position (glue)*: it represents the proportion of correct alignment’s elements recovered. For each glue attribute, its relevance is checked and the number of errors is reported on the total number of `glueVertical` and `glueHorizontal` attributes that should be specified in the model.

Isys Website

The first page analyzed is the home page of the Isys Website (<http://www.isys.ucl.ac.be>). The entirety of the UI elements is recovered, so as the layout and the graphical transitions (see table 9-2). However, there are some small errors in the style attributes for `textComponents`. The Web page requires a .CSS file which is not taken into account, and therefore some style attributes are not recovered in the models. Moreover, the color attribute should be set to `#000000` for the “bienvenue”, “enseignement” and “recherche” `textComponents` (no color is specified for these elements in the output of ReversiXML). There are also six values set to wrong attributes: all the images (`filet.gif`) below the titles (bienvenue, enseignement, ...) should have the `glueVertical` attribute set to `bottom` instead of the `glueHorizontal`

Chapter 9 Validation

attribute set to **bottom**. The results for this first experiment are very good, as the totality of the UI components, box composition and graphical transitions are recovered. The number of boxes is much higher than in the manual version, but the layout remains correct. The only errors on the page are for some color attributes or position attributes, which are not as important as the first three categories. There are less than 10 % of errors for these two last categories. The .CSS file contains only a new style definition for links, and the loss is thus not very important in this case.

Google

The second page analyzed is the well-known home page of the Google search engine (<http://www.google.be>). As for the previous analysis, the three most important categories are fully recovered. However, some errors still remain in the page: the box containing the entirety of the Google logo should possess a **glueHorizontal** attribute set to **middle**, but it is not recorded in the CUI model (as the logo is composed of four sub-images, four errors are reported in the table 9-2). Otherwise, all the other **glue** attributes are correctly recovered. There were two types of errors at the attribute level: the “nouveau” **textComponent** should possess an **isSuperScript** attribute set to **true**, but the attribute was not recorded in the UsiXML specification. Another error in the CUI model resides in the **radiobuttons** allowing selecting the scope of the Google-search: the first one should have the **defaultState** attribute set to **true**, while this attribute is missing in the CUI specification. Finally, the page possess a style tag (style attributes declared with the .CSS syntax and embedded in the HTML file), and as it is not analyzed, the information contained in this tag is not recovered in the model. However, these style tags only define the font pick order (arial before a sans-serif font) and are thus not very important.

The correctness of the CUI model was also verified for a reverse engineering using a configuration file (column g2 of table 9-2). The configuration of options was the same as in section 9.1.3 (i.e. a reverse engineering without images, color, external components, sound and style attributes). As for the “full” reverse engineering, the three first categories of elements were recovered entirely. Box layout was less complex as several empty boxes were deleted (as they did not contain images) and this operation was applied by preserving a correct structure of the entire page. The elements positions were also recovered without error, as the element positions errors of the full reverse engineering occurred only for images. For attributes, as style attributes were not recorded in the produced model, the error related to the **isSuperScript** attribute is not present in this simplified model. However, the

Chapter 9 Validation

`defaultState` missing attribute for `radiobuttons` was also not recovered in this second process. This is the only error of this configured reverse engineering.

	Isys	Google	(g2)	WCRE	(w2)	Airport	Infospace
Elements	100%	100%	100%	100%	100%	100%	100%
Graphical Transitions	100%	100%	100%	100%	100%	100%	100%
Box Layout	100%	100%	100%	100%	100%	99.75% 1/133	100%
Attributes	98.1% 3/160	97.7% 2/85	98.9% 1/85	99.0% 2/191	99.0% 2/191	100%	99.2% 3/356
Element Positions (glue)	91.9% 6/74	95.5% 4/89	100%	100%	100%	42.8% 24/56	80% 17/85

Table 9-2 Success rates for the correction analysis.

WCRE 2006 form

The third page analyzed in this study is the contact form of the WCRE 2006 conference (<http://www.rcost.unisannio.it/wcre2006/contacts/index.htm>). All the elements were recovered correctly (see table 9-2). All the graphical transitions were recorded in the CUI model. The box layout has also been recovered without any error, so as the elements positions. There were three errors at the attribute level: the size (`numberOfCols` and `numberOfRows`) of the `textarea` (derived as a `textComponent` in UsiXML) has not been recorded in the produced model, so as the border color of the last cell. This is because the color has been described in a style tag (.CSS expression), which is not recovered with the current version of the tool (the error was not counted in table 9-2). Errors on the page are very minor and the recovered model contains almost all the information from the original page.

As for the Google analysis, the WCRE Web form was reverse engineered with a configuration file designed for mobile phones (column w2 in table 9-2). Again, several boxes were removed but the layout remained correct after modifying the layout. The errors are exactly similar than in the previous “full” test.

Brussels airport

The fourth study is the analysis of the Brussels airport Website (<http://www.brusselsairport.be/>). The entirety of the UI widgets contained in the original page is recovered in the UsiXML specification (see table 9-2). But this page is a “special case” as two inline frames are used in the HTML code. The inline frames are linked with an .ASP page which is inaccessible for robots (but these small .ASP pages are accessible by normal browsers), and thus the code can not be downloaded.

Chapter 9 Validation

They contain in totality 28 UI elements which are not recovered in the CUI model, but it is more for technical reasons than an error in the approach or in the tool implementation.

For the box layout, one error occurred for the “hotline phone number” which is reverse engineered under the two logos “English” and “Nederlands” instead of on the same row. This error is relatively small as it only concerns one leaf element. A second layout error (replicated seven times) exists for the labels underlined with an orange line, as the line is recovered in the same horizontal box than the label in the UsiXML specification. This error is not taken into account as these elements are effectively specified side-by-side in the HTML code, but a style coming from an external .CSS file redefines the layout of orange lines by putting these element 15 pixels under the “normal” row vertical position, so the line appears to be under the label. Graphical transitions were recovered in their totality without errors.

This page had few attributes defined in the original source code, as it possess an heavy .CSS file in which every style/color attribute is defined. Therefore, every element in the CUI model possesses color attributes set to black (the default value if the tool can not find any color value in the file) and no style attributes. Moreover, alignment properties are mostly defined in the .CSS file in the present case. However, some alignment modifiers are defined in the HTML source file. Several errors remains for the detection of these glue attributes: in the first column on the left part of screen, 10 interaction objects should possess the `glueVertical` attribute set to `top` whereas nothing is specified in the UsiXML code for these elements. The same error occurs on the right side of UI, where 22 `gluevertical=top` attributes are missing. It comes from the fact that several alignment modifications are done thanks to div tags; their complex embedding is well recovered by the tool, but alignment modification coming from parent table or cell is lost when all the div tags are closed. This provokes a high number of errors for this last category.

Infospace

The fifth analysis is the reverse engineering of the Infospace Website (<http://www.infospace.com/>), a phone number directory page. All the interaction objects have been recovered correctly in the UsiXML model, so as the layout and graphical transitions.

There were some errors at the attribute level. Three `textComponents` were reverse engineered with a wrong color. A second error was that the `groupname` attribute of the three radio buttons was missing. Another type of error, due to style tags, was that the background color of three cells was not recovered. Instead of using the adequate

Chapter 9 Validation

attribute (`bgcolor`) of these elements, designers used a style tag to define these properties and this information was consequently lost. The same happened for the table border: designers put the border attribute to 0 but defined a style tag in which the border was defined at 1. These four last errors were not counted in table 9-2, as all they are due to style expressions. Finally, for the element positions, several alignment attributes were missing: six attributes `valign=top` were not recovered, one attribute `valign=bottom` and ten `valign=middle`. These errors are scattered across the page, and are again due to several overlapping alignment properties.

In conclusion for this section, the correction-analysis gave acceptable results for each case. The most important categories are almost entirely recovered (just one error for box layout category), but some errors remains for the detection of attributes and alignment properties. Almost all the errors in the attributes category occurred for style properties of `textComponents`. This comes from the fact that several ways exist in HTML to specify the style of labels, and often several tags are used to define the style parameters, by using different methods in a same HTML source file. This can create relatively complex cases, and keeping track of all the style changes is not easily done and causes some small errors at this level. The same phenomenon occurs with alignment properties, as several methods exist to assign this parameter (such as the alignment defined in the parent's containers, div tags, alignment of elements...) and some errors are made in the detection of the current alignment modifier.

However, these two types of errors could be corrected easily by putting more development efforts on these two subjects. The current implementation is achieved thanks to arrays, keeping the information of style or alignment properties. A special function allows checking the closing of tags, and in this case, removes the current slot and moves one slot back in the array of properties. When the style or alignment information is needed, arrays are read and the model is written accordingly. But this kind of solution implies inaccuracies, e.g. when several nodes are closed simultaneously, or when special cases occurs (e.g. usually, div tags applies to all its child nodes, except for the content of tables, and as only the alignment property is kept in the array- not the source element- the alignment properties are written erroneously). This array-solution was implemented to optimize the speed of the process. A solution based on the verification of the path of each element would give better results, but would then slower the reverse engineering process.

9.1.5 Conclusion for the internal validation

According to the working hypothesis and the aims of this research, it can be shown that the objectives are met after the internal validation. The tool gives satisfying results in the performance analysis (section 9.1.3), as the processes time are already relatively short and could be shortened by reducing the number of boxes and by optimizing the PHP code. Moreover, the performance analysis has been realized in the “ReversiXML-UI mode”, and results are even shorter under the form of a library as the construction and display of the model under the form of a tree is not executed in this second version. Thanks to the optional application of derivation rules, it is possible to recover a CUI model targeted for a particular (set of) platform(s). This reduces the work load of designers, as models are smaller (for the investigation in section 9.1.3 about 25 % in average and up to 50% in some cases) than a reverse engineering without option by removing elements not needed in the target model. The approach also gives the possibility to the designer to apply a “classic” reverse engineering, and let him do all the modifications herself in an edition tool or during the forward engineering phase.

The tool generates correct results as all the UI elements, their positioning and links with other models and elements are recovered entirely. Only one box layout recovery error occurred in the five studies of section 9.1.4. Some small errors remain at the attribute level, but are not “vital” in the sense that they only modify esthetical aspects (e.g. wrong color) of UI widgets or for the alignment of elements within a container. Important attributes such as **defaultContent**, **height**, **width**, **location** etc...are always well recovered. The alignment errors occurred in some cases very often, but this loss is not too important as glue attributes only help to fine tune the position of elements. If a page is described thanks to an important quantity of containers, the major part of the layout will be defined thanks to boxes and it will only produce small differences in the positions of interaction objects. Rules are applied without ambiguities (see section 9.1.2), i.e. a configuration of options – or group of derivation rules- applied to a defined set of elements will always give the same results. The rules cover almost the entirety of elements and concepts belonging to the HTML language (see section 9.1.1). The parts of the language that are ignored in this approach have been defined in the working hypothesis, or are not expressible in the target language.

The objectives of *flexibility and controllability* are met thanks to the correction and performance analysis, as the reverse engineering process can be parameterized to reverse engineer only those UI elements of interest and rejecting those not concerned, by reducing effectively and correctly the size of models. The designer can also control the process in the reverse engineering or/and in the translation step, by

applying some transformations of one type of interaction object into another and thus produces a model with only interaction objects existing on the target platform (retargeting).

Coverage combined with these two criteria allows also *generality* as the derivation and translation rules are not designed to accommodate a fixed target platform, but could be used to extract retargeted models for a large scope of platforms.

The *predictability* of the approach presented in this thesis is also achieved, by ensuring that almost all the elements can be reverse engineered, that a precise set of derivation rules generates only one possible result and also by the fact that these resulting models are correct (at least for the most important categories).

9.2 External Validation

The external validation (i.e. empirical validation of the approach) consists in evaluating the approach by designers and users of Web sites, i.e. by the target audience of this research. The first should use the tool to redesign a UI for a particular platform and the latter should use the new UI in order to evaluate the results. The populations targeted by this approach are Web/UI designers, persons in charge of the maintenance of Web sites and reverse engineering experts. Experts as the reverse engineering profession does not exist but more and more organizations specialize in the reverse engineering of information system (e.g. Fujitsu, Callatay and Wouters, Raincode...). Actors using a reverse engineering approach can have three different needs, i.e. recover abstract models in order to:

- Have a better understanding of the current system (e.g. for modifying or adding components to the system) by possibly generating documentation.
- Apply the reengineering to facilitate maintenance.
- Reengineer the system for other contexts of use.

This study should be a quantitative study but there is a difficulty to conduct such a type of validation as designers have a (high) cost and it is difficult to find a sufficient number of them to retrieve significant results of the validation.

Another burden is that the evaluation would be biased as the results of the reengineering would imply the use of another forward engineering tool and so this evaluation would validate both approaches. After evaluating the needed resources, this kind of investigation would be desirable but not practicable. Therefore, the quantitative investigation has been replaced by two other types of experiments: firstly a qualitative exploratory study of the reengineering of three Websites achieved by UI course's students and secondly by three cases study of various complexities.

The exploratory study allows us to understand how users will use the tool, and have an external feedback about the use, utility and performance of the tool. The case study shows how a Web page is reverse engineered by ReversiXML at the CUI and AUI levels, by illustrating only significant parts of the process.

9.2.1 Exploratory study of reengineering

This exploratory study consisted in reverse engineering three Web pages with ReversiXML combined with the forward engineering towards two different platforms, pocket pc and mobile phones, thanks to Grafxml (see appendix E on UsiXML compliant tools).

9.2.1.a Method

Three Web pages coming from the UCL repertoires were analyzed:

-<http://www.ucl.ac.be/reperto.html>

-<http://ultra2.sia.ucl.ac.be:8000/GIPE/SilverStream/Pages/pgRechercheCours.html>

-<http://ultra2.sia.ucl.ac.be:8000/GIPE/SilverStream/Pages/pgRechercheOffre.html>

The first page is an index page (see figure 9-1), where the user can choose various types of research in UCL databases. The second and third pages are two different types of researches, one for courses and the other for study programs.

Protocol Description. This qualitative study was realized by 17 master students in computer sciences and management sciences of the third cycle's course LINF 2356 (interaction homme-machine, a UI course) that had little experience with UI. These students were chosen because they have basic knowledge in the domain of reengineering and studied on the UI subject. Management sciences students had little or no experience with model-based approaches and had an intermediate level in computer science in general while computer sciences students had moderate experience in general but were experts in computer sciences. Students had to write a report commenting the approach by expressing freely their critics (no questionnaire was used as it was a qualitative evaluation). The 17 students were composed of 2 women and 15 men, and their age ranged between 22 and 27 years.

Population selection was biased in two manners: firstly, these students do not represent Web designers or reverse engineering experts and secondly, all the students of the course LINF 2356 without distinction were chosen to conduct this investigation, there was no characterization of the population. However, as this


Chapter 9 Validation

investigation is an exploratory study, we hope that difficulties about the use of the tool can be detected despite these facts.

UCL Université catholique de Louvain
Université membre de l'Académie universitaire 'Louvain'

Liste des répertoires disponibles à l'UCL

- [Répertoire du personnel](#)
- [Curriculum vitae des professeurs \(accès restreint\)](#)
- [Répertoire des étudiants](#)
- [Répertoire des entités](#)
- [Publications récentes de membres de l'UCL](#)
- [Inventaire de la recherche](#)
- [Répertoire des cours](#)



- [Libellule - Les bibliothèques de l'UCL](#)

[\[UCL\] \[Pointeurs utiles\]](#)

Dernière mise à jour : 5 octobre 2004 - Responsable : Patrick Tyteca - Contact : [webadep](#)

UCL Université catholique de Louvain
Université partenaire de l'Académie universitaire 'Louvain'

Recherche des cours

Validité	2004	
Sigle du cours		Sigle
Mot du titre		Mot
Mot(s) du cahier de charges/résumé		Mot opér
		Exer
		.
		.
Nom de l'enseignant		Nom prér
Département de charge		Sigle
Département de discipline		Sigle
Sigle du programme		Sigle
<input type="button" value="Rechercher"/> <input type="button" value="Recherch"/>		

[Recherche simple - Recherche par programmes - Programme d'études](#)
[Aide - Renseignements généraux](#)
[\[UCL\] \[Pointeurs utiles\]](#)

Responsable : Jean-Louis Marchand

UCL Université catholique de Louvain
Université partenaire de l'Académie universitaire 'Louvain'

Recherche des programmes

Validité	2004	
Mot du titre à rechercher		Mot ou partie de mot à rechercher dans le titre du programme
Mot de la page à rechercher		Mots ou parties de mots à rechercher dans les informations concernant les programmes. Vous pouvez utiliser les opérateurs AND et OR.
		Exemples :
		• math : chercher tout ce qui contient "math"
		• math AND géographie : chercher tout ce qui contient à la fois math et géographie
		• math OR géographie : chercher tout ce qui contient math ou géographie
Catégorie	TOUTES	
Horaire aménagé pour les adultes et expérience souhaitée	<input type="checkbox"/>	
Horaire aménagé pour faciliter l'accès aux adultes	<input type="checkbox"/>	
Formations accessibles aux étudiants de diverses facultés	<input type="checkbox"/>	
Formations destinées à l'enseignement supérieur non-universitaire	<input type="checkbox"/>	
Mémoire	<input type="checkbox"/>	
Stages	<input type="checkbox"/>	
Durée (en années)	Toute durée	
Inter-universitaire	<input type="checkbox"/>	
<input type="button" value="Rechercher"/> <input type="button" value="Recherche imprimable"/> <input type="button" value="Nouvelle recherche"/>		

[Recherche simple - Recherche par cours - Programme d'études](#)
[Aide - Renseignements généraux](#)
[\[UCL\] \[Pointeurs utiles\]](#)

Responsable : Jean-Louis Marchand

Figure 9-1 The three Web pages of the exploratory study

The results of this investigation had to be presented under the form of a report, in which the students had to describe the different retargeting operations used during this reengineering process and the manual modifications they realized after the reverse engineering (the instructions given to students can be found in appendix K). The report had to contain the illustrated description of each step of the process, and a conclusion containing the major positive and negative aspects of this reengineering. The configuration files used during this process, and the resulting UI saved in

Chapter 9 Validation

Grafixml had to be sent with the report. A short presentation (30 minutes) was done during a course to show the basic functionalities of the two tools.

Reengineering project description. Students had to use first ReversiXML to obtain a model suiting the two platforms by choosing the best configuration of reverse engineering options. The options selection can be characterized by issues on size-reduction (for a 96x65 screen), color and interaction object availability for mobile phones and only a size reduction problem for pocket PC (for a 240x320 screen). After the reverse engineering, students had to use Grafixml to edit the UI, by modifying, deleting or moving elements between windows. Some manual modifications to original components could also be made during this step. As Grafixml was not able to produce final code at that moment (May 2005), they had to show a preview of the final UI in the tool. Thus all the operations needed to reengineer Web pages were achieved until the code generation. Some examples of UI previews for this study are shown on figure 9-2.

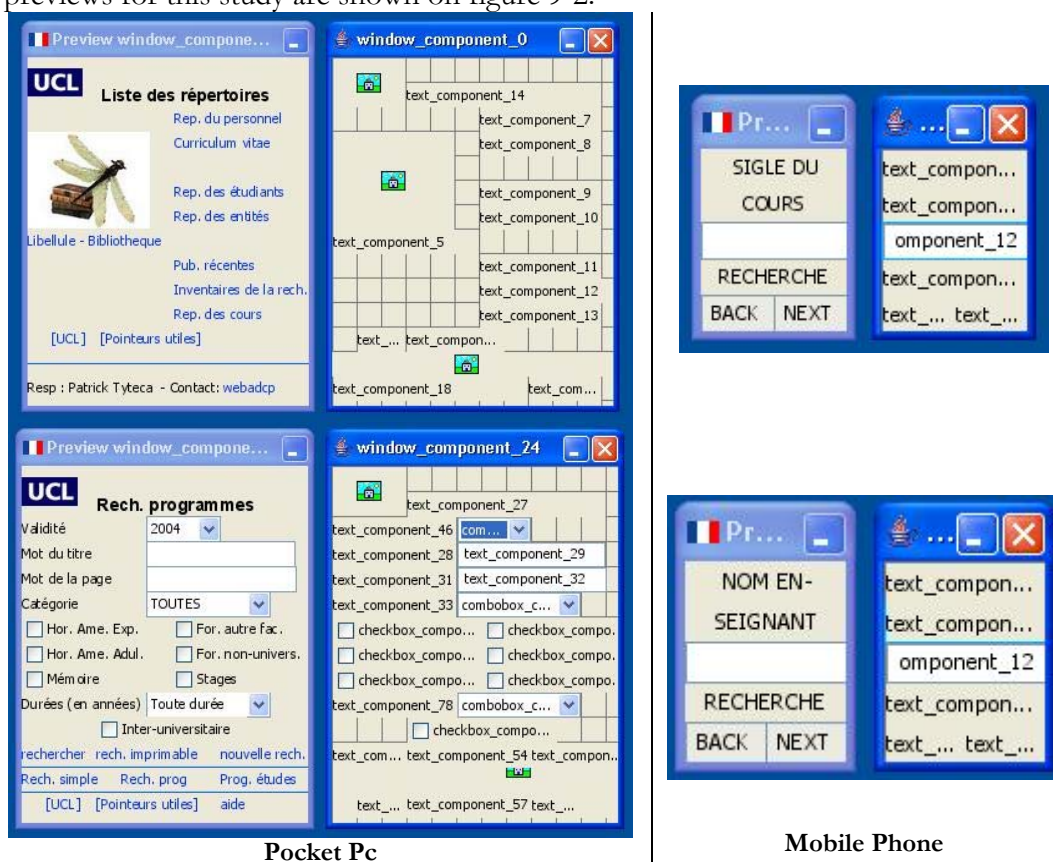


Figure 9-2 Reengineering in the exploratory study

9.2.1.b Results

While using ReversiXML, most students used the following built-in transformations/options:

- Colors removed (for mobile phones).
- Style attributes for text components removed (for mobile phones).
- Remove images (for mobile phones).
- Resize image (for pocket pc).
- Merge (fold) some labels (for both platforms).
- Remove components that are not available (for mobile phones).
- Transform interaction objects into other type of interaction objects taking less screen space (for both platforms).

Results were satisfying, really helping them to reduce the amount of work needed for the next translation step (see critics). Only one technical error was made during the reverse engineering that students had to correct by hand. This error was a wrong detection of the category of element (a **checkbox** was reverse engineered into a **textComponent** due to an implementation error).

To complete the translation, they had to modify the produced specification thanks to the Gرافixml UI editor. The most common and relevant actions are listed below by order of importance:

- Split windows into several small windows and add corresponding navigation components.
- Transfer of semantically linked elements to display them on a same screen.
- Reduce text size (by removing useless words or by replacing words by an abbreviation ...).
- Addition of interaction objects (e.g. add a “rechercher” button on each split window to let users launch research whenever they want).
- Reduce images size by removing white borders and reduce number of colors (these last transformations have been done thanks to an external drawing tool).
- Transformation of some long descriptive text components into the **toolTipText** attribute of the object that these labels describe. Another way to avoid long labels was to put “help”-links giving access to the label.
- Suppression of non-important labels, such as the update date or page author.

Some of these operations could be automated, such as window splitting, or the reduction of the number of colors of images, but it would be impossible to incorporate some of them into a tool (such as the label’s size reduction as it implies

the understanding of the semantic of the sentence). Therefore, a complete reengineering of quality (i.e. producing usable UIs) towards small platforms will always require a human intervention in the translation process to apply some modifications and refinements to the produced model. The topic about translation between two contexts of use is also the subject of other researches [Flor06] and thus only few transformations are developed in this thesis.

9.2.1.c Discussion

As a conclusion for this study, several positive and negative aspects of the tool are pointed out by the students. The most representative aspects are summarized below:

+ Positive

- The tool allows an easier development of a user interface for different context of use. The use of tools such as ReversiXML (and GrafiXML) reduces the amount of work required to reengineer an HTML UI for another platform.
- The tool is easy to use, there are just some difficulties at the beginning. The output is clearly presented and corresponds to the expected result.
- The use of configuration files allows saving time as it avoids removing elements and attributes, adding new components and correcting the model by hand.

Comments: Positive remarks of students were very interesting, as they show that users get the expected results (predictability of the process), and that the reverse engineering produces correct results with different configurations (correctness /flexibility). Moreover, the flexible process (i.e. reverse engineering configurations) has been found very important, as it facilitate greatly the reengineering process by cutting off useless components for a particular platform. Finally, all the students found that they saved time and efforts by using the tool compared to a simple generation of UI by observing the source UI.

- Negative

- The tool may generate errors during the reverse engineering, and it may be difficult to identify these losses without a careful examination of the model.
- Even if the tool is easy to use for some, other observed that no help or no guidelines are available and including it would make the process easier to understand.
- It was difficult to use configuration files, even when their locations were identified on the servers, and it appeared that the file could not be used remotely

for security reasons: users were not able to load configuration files saved on their own computer.

Comments: Negative aspects pointed out by students are more about technical problems than for the reverse engineering itself. Apart from wrong reverse engineering of an element, they do not have expressed important critics for the reverse engineering tool.

The three next sections illustrate the reverse engineering process with ReversiXML on various case studies. Some part of the HTML code and UsiXML is shown to describe how the code is derived into a CUI or AUI model.

9.2.2 Case Study 1: the Sedan-Bouillon Web site

The pages for this case study come from the sedan-bouillon Website, which was developed by IS3. This site has been chosen for three reasons:

- This tourist site is visited, by nature, by many users in different contexts of use.
- Several platforms access this Website (see figure 9-3) and therefore a reengineering of the UI for these different platforms/ users would have a sense.
- Finally, the site has been developed by IS3, an industrial partner of the Cameleon project, and thus illustrates how the method/tools apply to real world applications.

The case study illustrates a simple task: the user wants to order documentation, and has therefore to fill in a form. The user has to navigate through four pages to achieve this task (introduction page, global navigation page, page for the selection of the documentation to order, page containing form about customer information).

The pages can be found at:

- 1) <http://www.sedan-bouillon.com/>
- 2) <http://www.sedan-bouillon.com/catalogue/index.php>
- 3) <http://www.sedan-bouillon.com/catalogue/ctg.php?c=13>
- 4) <http://www.sedan-bouillon.com/catalogue/commande.php>

For each of these pages, some parts of the UsiXML specifications are extracted to illustrate the reverse engineering at the CUI level. To make the specification more readable, the two mandatory attribute, `isVisible` and `isEnabled` have been removed from the UsiXML specification, as these elements are always set to true. Another simplification has been made, the suppression of most of the boxes to reduce the size of code (box are only kept in the first example and are removed in the rest of the text and, thus these excerpts does not reflect exactly the HTML code).

Chapter 9 Validation

Systèmes exploitation			
OS		Hits	Pourcentage
	Windows XP	36446	44.1 %
	Windows 98	20148	24.4 %
	Windows 2000	12869	15.5 %
	Windows NT	5073	6.1 %
	Windows Me	4992	6 %
	Mac OS	1312	1.5 %
	Inconnu	1124	1.3 %
	Windows 95	397	0.4 %
	CP/M	89	0.1 %
	Windows CE	46	0 %
	Linux	32	0 %

Navigateurs				
Navigateurs		Aspirateur	Hits	Pourcentage
	MS Internet Explorer (Versions)	Non	80072	97 %
	Netscape (Versions)	Non	1162	1.4 %
	Inconnu	?	1115	1.3 %
	Nutsrape	Non	89	0.1 %
	Opera	Non	70	0 %
	Konqueror	Non	20	0 %

Figure 9-3 Platforms and navigators accessing the site

(source: <http://www.is3server.com/is3stats/is3stats.pl?config=sedan-bouillon.com&year=2004&month=year>)

9.2.2.a Introduction page

The first page of the Web site is very simple as it contains only 36 CUI elements and 22 boxes. The page contains two links, one 'mailto:is3' and the other one bringing the user to an index page.

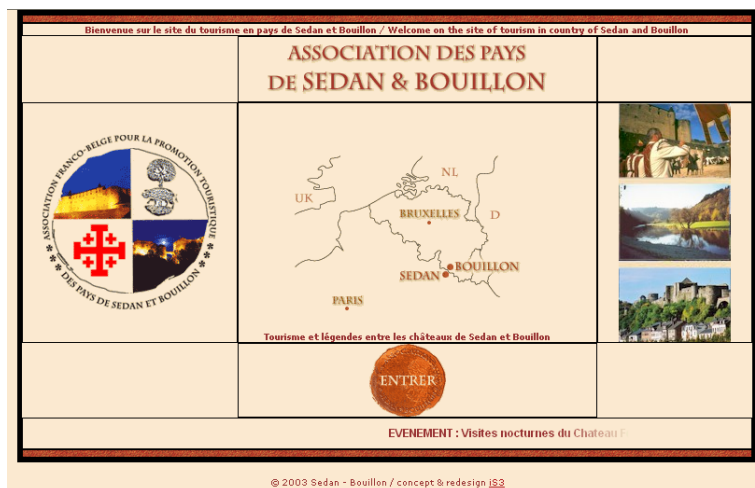


Figure 9-4 Division into boxes

The page is shown on figure 9-4. The division of the page into boxes based on the table-structure of the page has been made visible to show how the page will be

Chapter 9 Validation

divided into boxes in UsiXML. The black lines represent the cells of the table structuring the page in the HTML code. The beginning of the HTML code and its corresponding UsiXML code is shown below:

```
HTML
<HEAD><TITLE>Tourisme en pays de Sedan et Bouillon / tourism in country of Sedan
and Bouillon</TITLE></HEAD>
<BODY vLink=#b2161d aLink=#b2161d link=#b2161d bgColor=#f9e79f leftMargin=0
topMargin=0>
<TABLE cellSpacing=0 cellPadding=0 width=780 align=center background="" border=0>
<TBODY>
  <TR>
    <TD colspan=3>
<IMG height=7 src="1-Tourisme en pays de Sedan et Bouillon - tourism in country of
Sedan and Bouillon files/hp-top.gif" width=780 border=0></TD></TR>
```

```
UsiXML
<window Name="Tourisme en pays de Sedan et Bouillon / tourism in country of Sedan
and Bouillon" id="window_1" bgColor=#f9e79f windowLeftMargin="0"
windowTopMargin="0">
<Box type="vertical" name="boxVer_T_1" id="boxVer_T_1">
<Box type="vertical" name="boxVer_T_2" id="boxVer_T_2" glueHorizontal="middle"
borderWidth="0">
<Box type="horizontal" name="boxHor_R_3" id="boxHor_R_3">
<Box type="horizontal" name="boxHor_D_4" id="boxHor_D_4">
<imageComponent name="Image 5" id="Image 5" ImageHeight="7" ImageWidth="780"
defaultContent="1-tourisme%20en%20pays%20de%20sedan%20et%20bouillon%20-
%20tourism%20in%20country%20of%20sedan%20and%20bouillon_files/hp-top.gif"
imageborder="0" /> </Box> </Box>
```

When ReversiXML detects the body tag in the HTML code, a **window** is created in the CUI model. The **name** of this **window** corresponds to the content of the **title** tag in the HTML code. The other attributes of the **body** are copied into UsiXML attributes, and links color attribute are put in temporary variables which will be used when a link (**textComponent**) is created in the UsiXML specification. Then a first vertical box is created (“**boxVer_T_1**”) that will contain all the other boxes of the page (see layout detection algorithm, chapter 6), but it is not the counterpart of an existing object in the source file. The second vertical box (“**boxVer_T_2**”), corresponds to the **table** in the HTML code. As the table has an attribute **align=center**, an attribute **glueHorizontal=middle** is added in the UsiXML specification. The **glueHorizontal=middle** attribute means that the element is centered compared to its container (which is **boxVer_t_1**). Two other boxes are added because the tool detected a **<tr>** (**boxHor_R_3**) and a **<td>** tag (**boxHor_D_4**).

Finally, the content of the cell is translated in the UsiXML specification as an **imageComponent**. The value of the **src** attribute in HTML is replicated in the **defaultContent** attribute. The other attributes of the image are copied in similar attributes in the UsiXML specification.

The next portion of code represents the enter image. This is an image-link (when the user clicks on the image, the link is triggered and the target page is loaded) which gives access to the rest of the site.

Chapter 9 Validation

HTML code

```
<DIV align=center>
<A href="http://www.sedan-bouillon.com/catalogue/index.php" target=_top>
<IMG height=78 alt="Tourisme en pays de Sedan et Bouillon" src="1-Tourisme en
pays de Sedan et Bouillon - tourism in country of Sedan and Bouillon_files/hp-
entrer.gif" width=82 align=top border=0></A></DIV>
```

UsiXML code

```
<imageComponent glueVertical="top" glueHorizontal="middle" name="ImageLink_26"
id="ImageLink_26" imageHeight="78" imageWidth="82" defaultContent = "1-
tourisme%20en%20pays%20de%20sedan%20et%20bouillon%20-%20tourism%20in
%20country%20of%20sedan%20and%20bouillon_files/hp-entrer.gif" imageBorder="0"
hyperLinkTarget= "http://www.sedan-bouillon.com/catalogue/index.php" />
...
<graphicalTransition id="gt_1" transitiontype="open">
<source sourceId="ImageLink_26" />
<target targetId=" http://www.sedan-bouillon.com/catalogue/index.php " />
</graphicalTransition>
```

The three tags (`div`, `a`, `img`) are mapped into one element in the UsiXML code, an `imageComponent`. The source of the image is used to create the `defaultContent` attribute of the `imageComponent` and the size attributes are copied into the `imageHeight` and `imageWidth` attributes. As the image is embedded in a `div` tag, the centered alignment is reproduced in the UsiXML specification by the `glueHorizontal=middle` attribute. This image is aligned on the top of its cell, and therefore the attribute `glueVertical=top` is added. As this image is an image link, a graphical transition is created to represent the link in the UsiXML specification (`gt_1`). The graphical transition has as source the `id` of the `imageComponent` ("imageLink_26") and as target the `href` attribute of the HTML code.

9.2.2.b Index page

This second page (see figure 9-5) is an index page which gives access to the rest of the site. The particularity of this page is that it contains several flash components. The page is composed of 234 CUI elements, and 147 from them are boxes. The page is mostly composed of links (33) on this page and images (28).

HTML Code

```
<OBJECT codeBase=http://download.macromedia.com/pub/shockwave/cabs/
flash/swflash.cab#version=5,0,0,0 height=118 width=280 classid=clsid:D27CDB6E-
AE6D-11cf-96B8-444553540000>
<PARAM NAME="movie" VALUE="images\header1.swf">
<PARAM NAME="quality" VALUE="high">
<embed src="images\header1.swf" quality=high pluginspage="http://www.
macromedia.com/shockwave/download/index.cgi?P1_Prod_Version=ShockwaveFlash"
type="application/x-shockwave-flash" width="280" height="118"></embed></OBJECT>
```

UsiXML code

```
<finalComponent width="280" height="118" type="application/x-shockwave-flash"
glueHorizontal="middle" name="finalComponent_5" id="finalComponent_5" location
="images/header1.swf" />
```

Chapter 9 Validation

This portion of code corresponds to the top of the page, where the two animated images are displayed (with the is3 logo on the golf green).

The cell is translated into a horizontal box, and its content, a flash component is mapped as a finalComponent. The two tags, object and embed, correspond to the same component and are thus mapped as one finalComponent in the UI specification. The Flash source file for the finalComponent is stored in the location attribute, and the type of plug-in is stored in the type attribute in the UsiXML specification. There is a second Flash component on the page, translated in the UsiXML in the same manner as this first component. This second component contains also a link (a mailto:is3...) which cannot be detected by ReversiXML. Therefore a dynamic solution, as presented in section 6.4.5, could give more complete results than the current method. Another problem with this page is the navigation bar, under these two flash components. This navigation bar is entirely implemented with an external JavaScript, and is not analyzed by ReversiXML. Again, section 6.4.5 provides a potential solution for this kind of problem. In the current version, these two problems imply the missing of 11 links on this page.

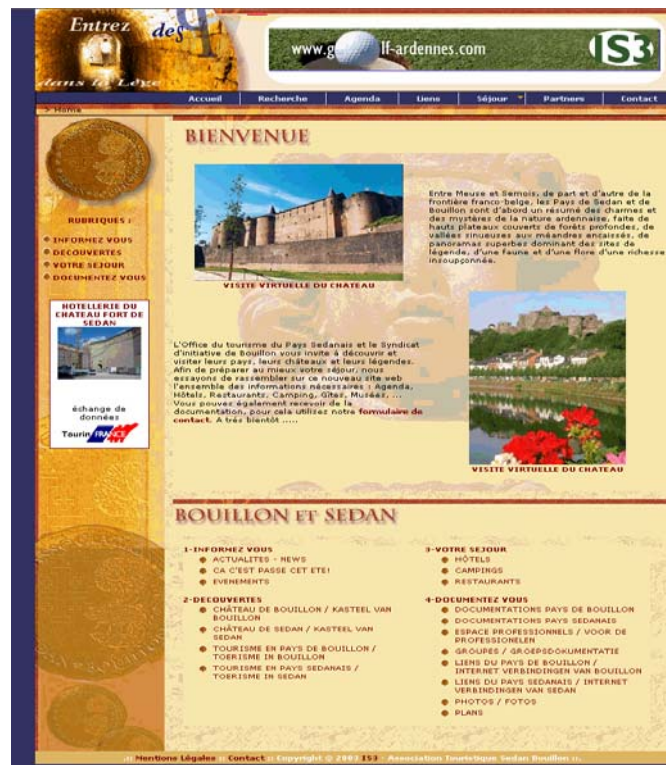


Figure 9-5 Index page of sedan-bouillon Website

Chapter 9 Validation

The second excerpt of this page is related to a textual link at the bottom of the page (shown at the bottom of the following table). The link is preceded by a bullet-image, mapped as an `ImageComponent` (`Image_140`). The link is derived in a `textComponent` with additional attributes. This `textComponent` has four more attributes than for normal static text: `linkVisitedColor`, `activeLinkColor`, `textColor` and `hyperlinkTarget`. The three first values can be found in the `body` node, and the `hyperlinkTarget` attribute corresponds to the `href` attribute from the HTML code.

HTML code

```
<IMG height=12 src="2-Sedan Bouillon Tourisme en Ardennes hotel, restaurant, camping, gite, chateau_files/piece.gif" width=14>
<A class=nick href="http://www.sedan-bouillon.com/catalogue/ssctg.php?c=11&s=45">TOURISME EN PAYS SEDANAIS / TOERISME IN SEDAN</A>
```

UsiXML code

```
<imageComponent name="Image_140" id="Image_140" ImageHeight="12" ImageWidth="14"
defaultContent="2-
sedan%20bouillon%20tourisme%20en%20ardennes%20hotel,%20restaurant,%20camping,
%20gite,%20chateau_files/piece.gif" /> </Box>
<textComponent name="TextLink_142" id="TextLink_142" defaultContent="
TOURISME EN PAYS SEDANAIS / TOERISME IN SEDAN" linkVisitedColor="#660000"
activeLinkColor="#660000" textColor="#660000" hyperlinkTarget="http://www.sedan-
bouillon.com/catalogue/ssctg.php?c=11&s=45"/>
...
<graphicalTransition id="gt_29" transitiontype="open">
<source sourceId="TextLink_142" />
<target targetId="http://www.sedan-bouillon.com/catalogue/ssctg.php?c=11&s=45" />
</graphicalTransition>
```



9.2.2.c Order Documentation

The third page (see figure 9-6) is dedicated to the ordering of documentation. This page gives the choice to the user between several types of documentation (over different cities, books, etc.).

When the user chooses one of the links on the bottom of the page, he has to complete a form, which is the fourth page of the case study (see figure 9-7). This page is composed of 217 CUI elements, where 93 of them are boxes, 17 inputs elements, 40 labels, 12 images and 6 links (graphical transitions).

The first portion of code represents the three radio buttons where the user can choose his/her title (Mr, Miss, Mrs)

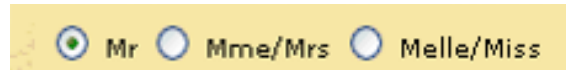
Chapter 9 Validation



Figure 9-6 Choose documentation

HTML code

```
<FONT face="Verdana, Arial, Helvetica, sans-serif" size=1>  
<INPUT type=radio CHECKED value=Mr name=Titre> Mr  
<INPUT type=radio value=Mrs name=Titre> Mme/Mrs  
<INPUT type=radio value=Miss name=Titre> Melle/Miss </FONT></TD>
```



UsiXML code

```
<radioButton defaultState="true" name="radiobutton_0" groupname="Titre"  
internalValue="Mr" id="radiobutton_0"/>  
<textComponent defaultContent="Mr" textFont="Verdana, Arial, Helvetica, sans-  
serif" textColor="#000000" textSize="1" name="label_11" id="label_11"/>  
<radioButton defaultState="false" name="radiobutton_1" id="radiobutton_1"  
groupname="Titre" internalValue="Mrs"/>  
<textComponent defaultContent="Mme/Mrs" textFont="Verdana, Arial, Helvetica,  
sans-serif" textColor="#000000" textSize="1" name="label_12" id="label_12"/>  
<radioButton defaultState="false" name="radiobutton_2" id="radiobutton_2"  
groupname="Titre" internalValue="Miss"/>  
<textComponent defaultContent="Melle/Miss" textFont="Verdana, Arial, Helvetica,  
sans-serif" textColor="#000000" textSize="1" name="label_13" id="label_13"/>
```

The font tag is not translated into an element, but its values are recorded in temporary variables. The first radio buttons from the HTML code (input type=radio) has an attribute CHECKED and is translated into a radioButton with its attribute defaultState set to true. The name attribute of the HTML radio button is copied into the groupname attribute in UsiXML, as this attribute indicates which radio button are from the same set. The value attribute of the HTML radio button is translated in UsiXML as its internalValue.

Chapter 9 Validation

The first label, “Mr”, is translated into a `textComponent(label_11)`, with its `textSize` set to 1 and its `textFont` attribute set to Verdana, Arial, Helvetica, sans-serif, which are the values of the `font` tag. The color of the label is black (`#000000`), which is the default value for labels as no indication is given in the code to change this value.

The same translation rules are applied for the two next radio buttons (`radio_button_1` and `radio_button_2`) and the two next labels (`label_12` and `13`). There is only one difference for the radio buttons: the `defaultState` attribute is set to `false`, as there is no checked attribute for these radio buttons.

The image shows a web browser window displaying a contact form. The page has a yellow background with a large, faint image of a lion's head. At the top, there is a navigation menu with links: Accueil, Recherche, Agenda, Liens, Séjour, Partners, and Contact. The main heading is 'CONTACT' in large, bold letters. Below the heading, there is a sub-heading 'Documentations' and a red instruction: 'Remplissez ce formulaire pour recevoir nos documentations. Please, complete this form and we will rush your request to you!'. There are three columns of document covers: 'Le Château Fort de Sedan', 'Carte Sedan - Bouillon', and 'Guide du Pays Sedanais'. Below these, there are radio buttons for 'Mr', 'Mme/Mrs', and 'Mlle/Miss'. The form fields include: 'Nom / Name *', 'Prénoms / Firstname *', 'Organisation', 'Adresse / Address *', 'Code Postal / ZipCode *', 'ville / City *', 'Country *' (with a dropdown menu), 'Phone', 'Fax', 'E-mail *', and 'Message'. A 'Send' button is at the bottom of the form. The footer contains 'Mentions Légales - Contact - Copyright © 2003 ISB - Association Tourisme Sedan Bouillon'.

Figure 9-7 Order documentation form

The next excerpt of code represents the email and message input fields and the “send” button (see the image in the table). The layout has been simplified to clarify code, by removing all the tables, cells and rows from both codes excerpts.

HTML code

```
<FONT face="Verdana, Arial, Helvetica, sans-serif" color=#000000 size=1>  
<B>E-mail <FONT color=#993333>*</FONT> :</B></FONT></TD>  
<FONT face="Verdana, Arial, Helvetica, sans-serif" size=1>  
<INPUT size=35 name=email> </FONT>
```

Chapter 9 Validation

```
<FONT face="Verdana, Arial, Helvetica, sans-serif" color=#000000
size=1><B>Message :</B></FONT></TD>
<FONT face="Verdana, Arial, Helvetica, sans-serif"size=1>
<TEXTAREA name=message rows=3 cols=50>commande de documentations</TEXTAREA>
</FONT>
<DIV align=left><FONT face="Verdana, Arial, Helvetica, sans-serif"size=1>
<INPUT type=submit value=Send name=Submit> </FONT></DIV>
```



UsiXML code

```
<textComponent name="text_372" id="text_372" defaultContent="E-mail" textSize="1"
textFont="verdana, arial" textStyle="bold" textColor="#000000" />
<textComponent name="text_373" id="text_373" defaultContent="*" textSize="1"
textFont="verdana, arial" textStyle="bold" textColor="#993333" />
<textComponent name="text_374" id="text_374" defaultContent=":" textSize="1"
textFont="verdana, arial" textStyle="bold" textColor="#000000" />
<textComponent name="textbox_376" id="textbox_376" numberOfColumns="35"/>
<textComponent glueHorizontal="middle" name="text_379" id="text_379"
defaultContent="Message :" textSize="1" textFont="verdana, arial, helvetica,
sans-serif" textStyle="bold" textColor="#000000" /> </Box>
<textComponent name="txtArea_380" id="txtArea_380" defaultContent="commande de
documentations" numberOfRows="3" numberOfColumns="50" />
<button name="submitBt_384" id="submitBt_384" defaultContent="send" />
```

Again, a font tag surrounds each textual content, and its attributes will be inherited by every label contained in this font tag. The label “Email*.” is translated into three different labels in the UsiXML code, as the color of the * (#993333) is different from the two other parts of the label (“email” and “.” are displayed in black). As the elements are also contained in a tag, the textStyle is set to bold for the three labels.

A textbox (<input type=text> is translated in the UsiXML language as a textComponent (here text_376). This element possesses an additional attribute: numberOfColumns="35". It corresponds to the size of the textbox.

The next component, the label “message:” can be translated by only one textComponent, as there is no change of color or style in this label. The next widget is a <textarea> tag, where the user can write some additional comments to his form. As for the textbox, the textarea is translated by a textComponent, with one more attribute than for the textbox, numberOfRows="3", to represent the second dimension. As this interaction object contains already a default text (“commande de documentations”), this one is put in the defaultContent attribute of the element.

Finally, the send button is represented in the UsiXML code by the button element. The value attribute in HTML represent the label of the button and is translated in UsiXML in the defaultContent attribute. The align=left attribute has not been reverse engineered, as this value (left) is the default value for glueHorizontal attributes. A

graphical transition having as source this button is also added at the end of the model, with the URL to which the form will be sent when the user clicks the submit button.

9.2.2.d Complete reengineering

With Teresa

Teresa (see appendix E on UsiXML compliant tools) has been used in this case study to apply a forward engineering based on the produced model. In a few words, the Teresa tool is developed at ISTI-CNR and supports the generation of XHTML, VoiceXML and WML starting from a task model, an abstract or concrete UI model expressed in TeresaXML, or a concrete UI specified in UsiXML.

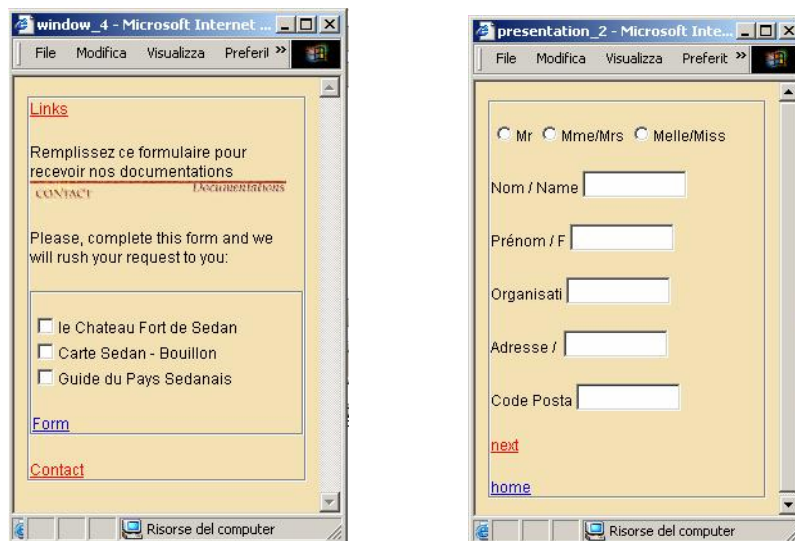


Figure 9-8 Results of the forward engineering with Teresa

For this reengineering, the last page of the case study (section 9.2.2.c) has been used as input in ReversiXML. The page has been reverse engineered without configuration file, as some transformations are performed by Teresa. The UsiXML produced by ReversiXML is then loaded in the Teresa tool, which then applies some modifications to produced CUI model and generates several UI specifications for various platforms. The source Web page can be found on figure 9-7 so as the resulting UI for Pocket PC specified in XHTML (figure 9-8). More information about this reengineering process can be found in [Mori05].

Chapter 9 Validation

With QtXML

The QtXML tool has been developed by Vincent Denis [Vinc05] in his master thesis in management sciences. The tool is able to produce oZ/Qt code based on UsiXML specifications and is available on <http://www.UsiXML.org>.

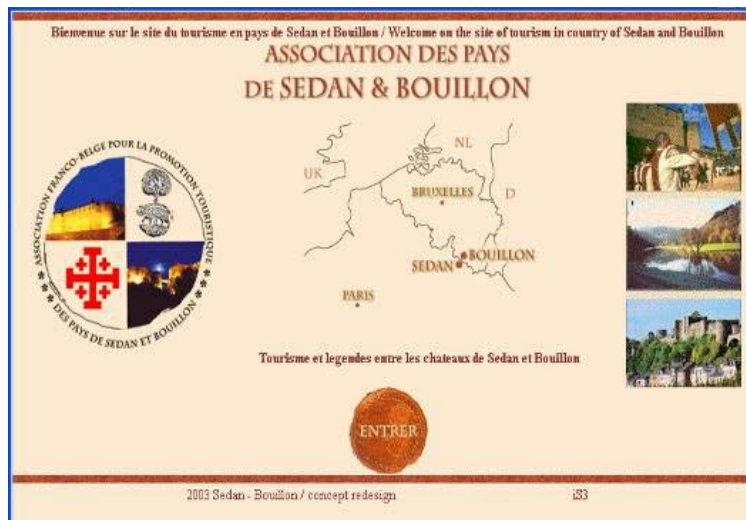


Figure 9-9 First page of case study regenerated with QtXML

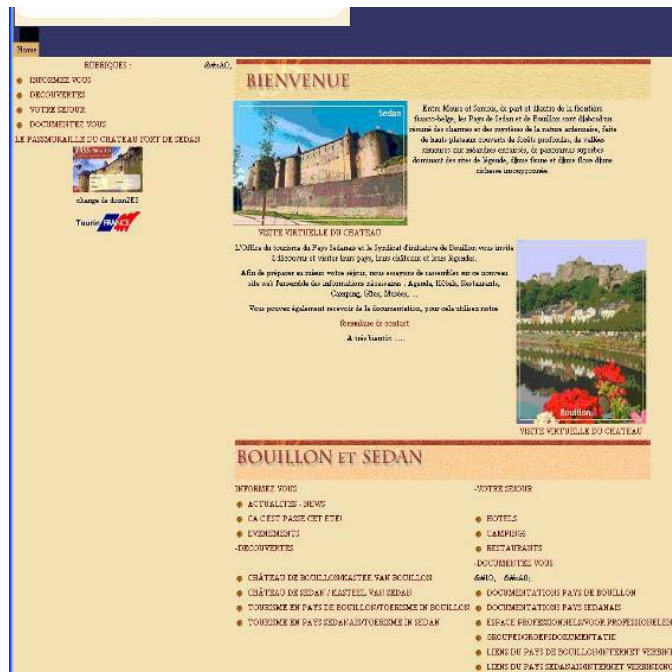


Figure 9-10 Second page of case study regenerated with QtXML

Chapter 9 Validation

Specifications reverse engineered by ReversiXML have been used in the tool to generate new UI code. The results of the reengineering are shown on figure 9-9, 9-10, 9-11 and 9-12. The reengineered pages are very similar to the original ones (figure 9-4, 9-5, 9-6 and 9-7). However small differences exist, such as the absence of the flash components (replaced by a blue line in the top of the UI, difference number 1 on fig. 9-11) as they can not be inserted in the Qtk environment as external components.

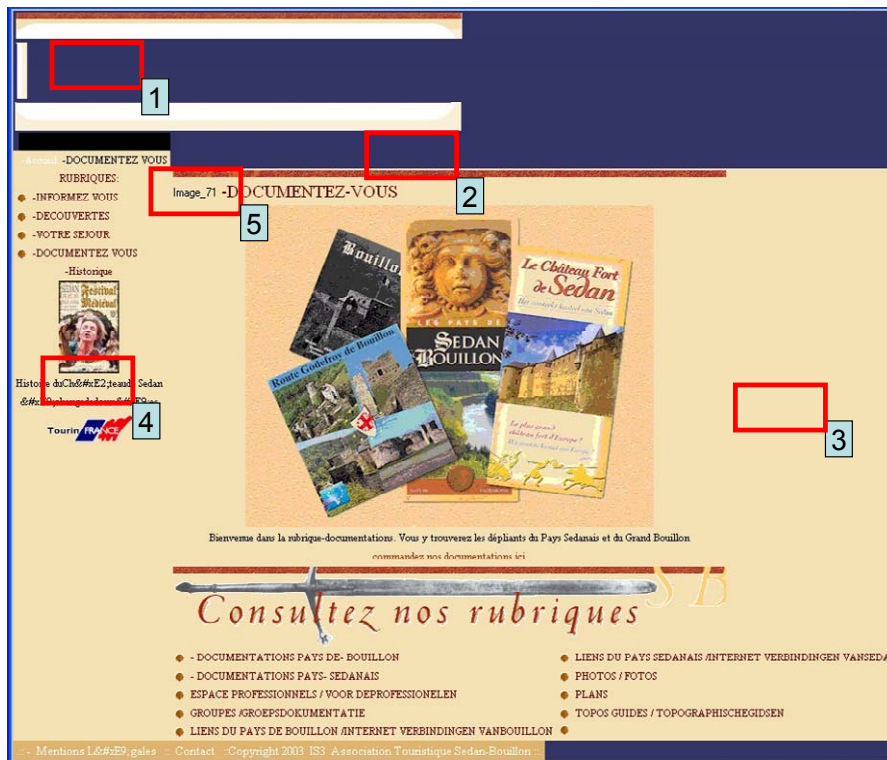


Figure 9-11 Third page of the case study regenerated with QtXML

The navigation menu just below the flash component (difference number 2 on figure 9-11) has also not been reengineered, as it was specified in javascript and thus not recovered by ReversiXML. Another difference is that Qtk does not accept background images (only colors) for `td` and `tr` widgets (representing boxes in Qtk) and therefore the derivations rules related to this attribute have been deselected in the configuration file before using ReversiXML (difference number 3 on figure 9-11). Some special characters are displayed in the Unicode format (difference number 4 on figure 9-11) as this character set is not supported in Qtk.

Chapter 9 Validation

Images can also be used in QtXML, but only those in the .GIF and .BMP format. JPG images have been replaced by their name during the forward engineering (see for example the “image_71” label, difference number 5 on figure 9-11). There are also some small visual differences due to the widget set of Qtk that one can remark on figure 9-12, where text boxes appear in grey, and with the “type here” default content. The dropdown list box of figure 9-7 has been replaced by a simple list box on this last example.

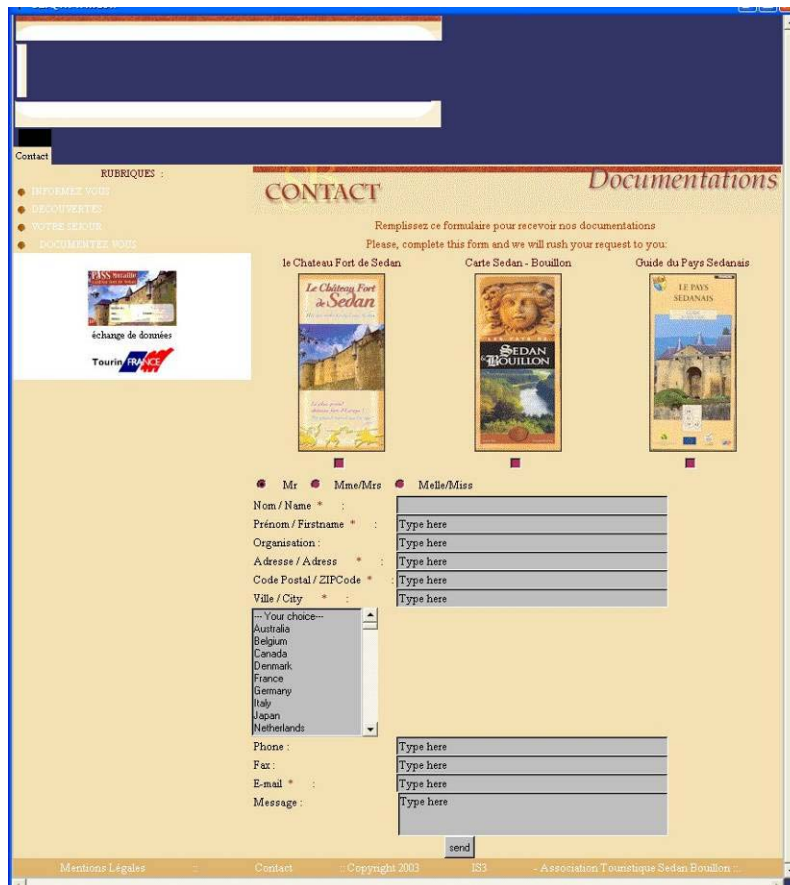


Figure 9-12 Fourth page of the case study regenerated with QtXML

Minor differences exist in the layout of the new UIs. This is more due to a lack of expressiveness of the box mechanism in UsiXML than to the combination of the reverse and forward engineering of ReversiXML and QtXML. The box layout is currently refined in the UsiXML language in order to obtain a more precise definition of the position of element, but the new version of the language including this enhancement is not completely finished for the moment.

9.2.3 Case study 2: UGC movie booking service

The subject of this second analysis is the booking service of UGC cinemas (available at http://www.ugc.be/FR/reservation/ChoixResa.jgi?RAZ_REGION=O), shown on figure 9-13. This page has been chosen for two reasons: it illustrates the reverse engineering of a common interaction object that had not been analyzed in previous case study and because this page (and the entire site) would be a good candidate for a reengineering for mobile platforms, as it is a typical service researched by mobile phone users.

The Web page is composed of 271 CUI elements, but 150 of them are **boxes** and 91 **images** (mostly blank images to fill space in order to have a stable layout). The rest of elements is divided into **textComponents** and images-links (not counted in the 91 images). The produced model is 42,4 ko long, which is about 150% of the size of the original page (27,7 ko). This page allows the user to make a reservation by choosing a cinema in a region of Belgium. He can access the region by clicking either on the Belgian map, or by clicking on the links displayed over the map. The other components composing the page are a navigation bar for UGC Website at the bottom of the UI, and another navigation bar on top of Belgian map where the user can choose another type of bookings (by movie and by timetable).

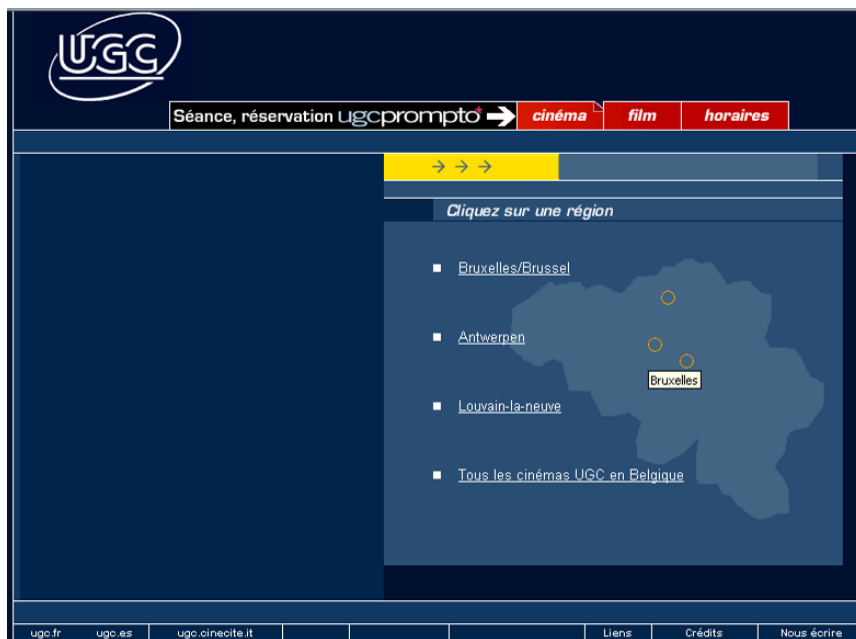


Figure 9-13 City selection page on UGC reservation site

Chapter 9 Validation

The first excerpt of code shows how a part of this latter menu is reverse engineered (the table structure has been removed).

HTML code

```
<div align="right">
</div>
<a href="http://www.ugc.be/FR/cinematheque/Cinematheque.jgi?TYPE=film" onMouseOut
MM_swapImgRestore()" onMouseOver="MM_swapImage('film11','','img/home/film2-new_
over.gif',1)">
</a>
```



UsiXML code

```
<imageComponent glueHorizontal="right" name="Image_43" id="Image_43"
ImageHeight="26" ImageWidth="317" defaultContent="img/home/sceance2.gif" />
<imageComponent name="ImageLink_46" id="ImageLink_46" imageHeight="26"
imageWidth="79" defaultContent="img/home/cinema2-new_out.gif" imageBorder="0"
hyperLinkTarget="http://www.ugc.be /FR/reservation/ChoixResa.jgi?TYPE=film" />
. . .
<graphicalTransition id="gt_3" transitiontype="open">
<source sourceId="ImageLink_46" />
<target targetId="http://www.ugc.be/FR/reservation/ChoixResa.jgi?TYPE=film" />
</graphicalTransition>
```

The first image (sceance2.gif) is embedded into a `div` tag that possesses an `align=right` attribute. Therefore, this first image in UsiXML has a `glueHorizontal=right` attribute, as this property is inherited by all the children of the `div` tag in HTML code (a factorization that does not exist in UsiXML, and thus the `glue` attribute has to be replicated for each of its children during the reverse engineering, in the present case only once).

The next images in the HTML code are associated with a small JavaScript, allowing the addition of a “roll-over effect”, i.e. the image changes when the user moves his mouse over and out of the image area. As JavaScripts are not analyzed, only the first displayed image is recorded in the UsiXML specification. All these images are “image-links”, i.e. a graphical transition is triggered when the user click on it. An example of `graphicalTransition` (for `imageLink_46`, the first “image-link”) is shown on the bottom of the example, where the `source` indicates the interaction object allowing this transition and the `target` contains the URL accessible thanks to the transition. In this case, the loss due to the fact that JavaScript are not recovered is very minor (only a visual effect).

The second excerpt of code illustrates the image-map component. An image map is an area divided into zones equivalent to links. In this case, three zones are available, one for Brussels, the second for Louvain-la-Neuve and third for Antwerp. The equivalent links are also displayed in a textual form on the left of the image.

```
<TD><a href="/FR/reservation/ChoixResa.jgi?REGION=1">
<span class="titreblanc"><u>Bruxelles/Brussel</u></span></a></TD>
```

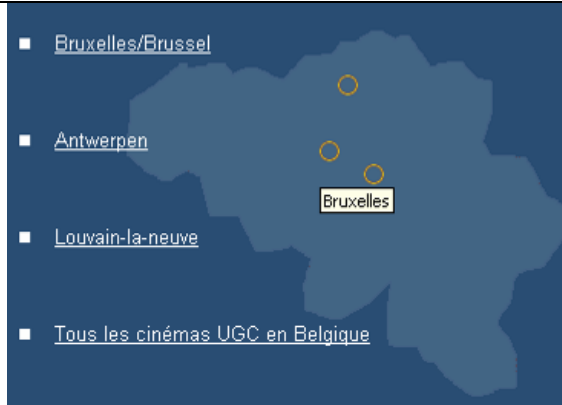
Chapter 9 Validation

```

<TD colspan="2" rowspan="15" valign="top">

<map name="MapRegion">
<area shape="circle" coords="68,101,10" href="/FR/reservation/
ChoixResa.jgi?REGION=1" alt="Bruxelles">
<area shape="circle" coords="78,58,10" href="/FR/reservation/
ChoixResa.jgi?REGION=2" alt="Anvers">
... </map></TD></TR>
<TR><TD colspan="3">
</TD>
<TD></TD>
<TD><a href="/FR/reservation/ChoixResa.jgi?REGION=2">
<span class="titreblanc"><u>Antwerpen</u></span></a></TD></TR>

```



```

<Box type="horizontal" name="boxHor_D_153" id="boxHor_D_153">
<textComponent glueHorizontal="middle" name="TextLink_154" id="TextLink_154"
defaultContent="Bruxelles/Brussel" HyperLinkTarget="/FR/reservation/
ChoixResa.jgi?REGION=1" isUnderline="true" /> </Box>
<Box type="horizontal" name="boxHor_D_155" id="boxHor_D_155">
<imageComponent glueVertical="top" name="ImgMap_156" id="ImgMap_156"
ImageHeight="300" ImageWidth="236" defaultContent="img/blank.gif" imageBorder="0">
<ImageZone Shape="circle" name="ImgZone_157" id="ImgZone_157" coordinates="68,
101,10" hyperLinkTarget="/fr/reservation/choixresa.jgi?region=1" />
<ImageZone Shape="circle" name="ImgZone_158" id="ImgZone_158"
coordinates="78,58,10" hyperLinkTarget="/fr/reservation/choixresa.jgi?region=2" />
...</imageComponent></Box></Box>
<Box type="horizontal" name="boxHor_R_161" id="boxHor_R_161">
<Box type="horizontal" name="boxHor_D_162" id="boxHor_D_162">
<imageComponent name="Image_163" id="Image_163"
defaultContent="img/reservation/blank.gif" /> </Box>
<Box type="horizontal" name="boxHor_D_164" id="boxHor_D_164">
<imageComponent name="Image_165" id="Image_165" ImageHeight="7" ImageWidth="7"
defaultContent="img/reservation/bl.gif" /> </Box>
<Box type="horizontal" name="boxHor_D_166" id="boxHor_D_166">
<textComponent glueHorizontal="middle" name="TextLink_167" id="TextLink_167"
defaultContent="Antwerpen" hyperLinkTarget="/FR/reservation/ChoixResa.jgi?REGION=2"
isUnderline="true" /> </Box></Box>

```

This HTML code is composed of 3 table rows (tr) composed of several cells (td). The first row contains the Brussels text link, which is placed in the <u> tag (underlines the embedded text nodes). Therefore, the textComponent in the UsiXML specification possesses the isUnderline="true" attribute.

Chapter 9 Validation

The second cell has the `valign=top` attribute in its specification: all its content will have a `glueVertical=top` attribute (in this case only one element, the component `ImgMap_156`). This cell contains an image map, which is in fact a blank image (`blank.gif`). This blank image is superposed on the Belgian map image, which is the background image of a previous cell (not present in the excerpt of code). The `usemap` attributes in the HTML code defines the image map to use (in this case `MapRegion` which is situated just after). In the UsiXML specification, this is translated by an `imageComponent`, with four `ImageZone` children (only two are shown in the excerpt of code). Every `ImageZone` corresponds to an `area` tag, representing the image-division. The `shape` and `coordinates` UsiXML attributes are copied from the `shape` and `coords` attribute in the HTML code.

The next row is composed of three cells. The first cell contains a blank image, the second cell contains an image representing the bullet and the third cell contains the link to Antwerp City. This last element is translated by a `textComponent`, and the `textnode` “Antwerpen” is the value of the `defaultContent` attribute.

The same page has been reverse engineered a second time with a configuration file designed for mobile phones. The images, style and color attributes have been removed from the reverse engineering process. The number of elements composing the UI is only 184 (about 100 less than in the “full” version), and the produced model is 14,752 ko long. The difference here is very significant, as the “lightened” model is below 35% of the size of the previous model, and would thus facilitate greatly the reengineering process to a mobile phone. The same excerpt of code related to the image map is shown below in its ‘mobile-version’.

```

                                UsiXML
<Box type="horizontal" name="boxHor_D_103" id="boxHor_D_103">
<textComponent glueHorizontal="middle" name="TextLink_104" id="TextLink_104"
defaultContent="Bruxelles/Brussel" hyperlinkTarget="/FR/reservation/ChoixResa.jgi
?REGION=1" /> </Box>
<Box type="horizontal" name="boxHor_R_111" id="boxHor_R_111">
<Box type="horizontal" name="boxHor_D_114" id="boxHor_D_114">
<textComponent glueHorizontal="middle" name="TextLink_115" id="TextLink_115"
defaultContent="Antwerpen" hyperlinkTarget="/FR/reservation/ChoixResa.jgi
?REGION=2"/> </Box></Box>
```

The size of the excerpt has been drastically reduced. Instead of 15 interaction objects, it can now be represented thanks to five objects only. Only the two `textComponents` representing the links to Brussels and Antwerp’s regions are still in the model, and in their minimal form (as all the style attributes have been removed). Two attributes are specified, the text of the link (`defaultContent`) and the targeted

Web page (`hyperLinkTarget`). Another difference with the previous version is the number of **boxes**. Numerous **boxes** have been deleted as by removing images, some of them are left empty. These empty **boxes** are deleted after the parsing of the HTML file. No special heuristic had to be used during this second reverse engineering, as the navigation was duplicated (the image map's links where also displayed in their textual form). In other cases, a rule such as 'transform image maps into a list of links' could have been used to keep the links in the model.

9.2.4 Case study 3: SNCB homepage

This third case study is the reverse engineering of the SNCB homepage (see figure 9-14) available at <http://www.b-rail.be/main/F/index.php>. The particularity of this case study is that the site has been reverse engineered at the two levels of abstraction, at the CUI and AUI levels. The reverse engineering at the AUI level is not totally finalized, and had to be completed by hand.



Figure 9-14 SNCB Homepage

At the CUI level, the UI is composed of 311 elements, mostly boxes as 126 of them are used to describe the layout of the UI. Put aside boxes, the UI is composed of images (47 normal images and 7 images link) and 27 `textComponents`. The rest of the UI specification is composed of form elements and 17 `graphicalTransitions`. The first excerpt of the UI illustrated here is the menu on top left corner of figure 9-16, allowing the user to select another language (Dutch, German or English). The

Chapter 9 Validation

HTML code presented here is composed of 4 cells (td) containing one simple image (hp_corner.gif) and three image links.

```
HTML code
<td></td>
<a href="/main/N/index.php">
 </a>
```



```
UsiXML CUI
<Box type="horizontal" name="boxHor_D_27" id="boxHor_D_27">
<imageComponent name="Image_28" id="Image_28" ImageHeight="22" ImageWidth="32"
defaultContent="../../assets/images/interface/hp_corner.gif"/> </Box>
<imageComponent name="ImageLink_30" id="ImageLink_30" imageHeight="22"
imageWidth="32" defaultContent="../../assets/images/interface/bu_nl.gif"
imageBorder="0" hyperLinkTarget="/main/N/index.php"/>
. . .
<graphicalTransition id="gt_6" transitiontype="open">
<source sourceId="ImageLink_32"/>
<target targetId="/main/D/index.php"/>
</graphicalTransition>
```

```
UsiXML AUI
<abstractContainer id="AC27" name="AC27">
<abstractIndividualComponent id="AIC28" name="AIC28">
<output id="AIC28" name="AIC28" outputContent="../../assets/images/interface/
hp_corner.gif"/> </abstractIndividualComponent>
</abstractContainer>
<abstractIndividualComponent id="AIC30" name="AIC30">
<output id="AIC31" name="AIC31" outputContent="../../assets/images/interface/
bu_nl.gif" />
<navigation id="AIC32" name="AIC32"/>
</abstractIndividualComponent> . . .
<audiDialogControl id="DC1" name="DC1" type="[]" >
<source id="AIC30"/>
<target id="/main/N/index.php"/>
</audiDialogControl>
```

The UI specification at the CUI level is composed of four horizontal boxes, derived from the table cells. Each of these horizontal **boxes** contains an image. To reduce code length, only the first cell and two first images are kept into the excerpt of specifications. The first image does not possess a **hyperLinkTarget** as it is a simple image. The three other images (only one is shown) possess this attribute and are linked with a **graphicalTransition**. An example of these **graphicalTransitions** is shown for **ImageLink_32** with the URL to which the image gives access.

The same structure of specification can be found at the AUI level, presented after the CUI specification. Each cell is represented by an **abstractContainer** at this level. Each **abstractContainer** is composed of an **abstractIndividualComponent**, which are the abstraction of images. An **abstractIndividualComponent** (AIC) represents a widget of the user interface, and has between one and four facets (**input**, **output**, **navigation** and **control** facet). As the first image is a simple picture, only one facet is used to describe

Chapter 9 Validation

this component, the **output** facet. This **output** element possesses only one attribute (put aside the **id** and **name** attributes), **outputContent**, which plays a similar role as the **defaultContent** attribute of the CUI model. The next image is an image link, and this is represented by the addition of a **navigation** facet to the **AIC**. This navigation is then defined at the end of the model in the **auDialogControl** element. This last element contains three different information, the **id** of the **source** (here the navigation **id** from the first image contained in a link) and the URL or name of the link's **target** so as the **type** of dialog control. **Types** of dialog controls are specified thanks to LOTOS operators. In the present case, the specified symbol is $[>]$, meaning that the current specification is disabled if the transition is triggered.

The second example is a small part of the form at the bottom left part of the UI. This excerpt is composed of a label and two drop down list boxes allowing the user to select a time. The second part of this example contains two radio buttons, permitting the user to define the previous time as an arrival or departure time. Table structure and options of drop down list boxes have been removed to clarify the code.

HTML code

```
Heure <select class='copy' type='text' name='TravelHours'>
<option value="00" >00</option>
<option value="01" >01</option> ... </select>
<select class='copy' type='text' name='TravelMinutes'>
<option value="00" >00</option>
<option value="05" >05</option> ... </select>
```

Heure	13	00
-------	----	----

UsiXML CUI

```
<textComponent name="text_169" id="text_169" defaultContent="Heure" />
<comboBox>
<item name="item_173" id="item_173" defaultContent="00" />
<item name="item_174" id="item_174" defaultContent="01" /> ...
</comboBox>
<comboBox>
<item name="item_197" id="item_197" defaultContent="00" />
<item name="item_198" id="item_198" defaultContent="05" /> ...
</comboBox>
```

UsiXML AUI

```
<abstractIndividualComponent id="AIC211" name="AIC211">
<output id="AIC212" name="AIC212" outputContent="Heure"/>
</abstractIndividualComponent>
<abstractIndividualComponent id="AIC214" name="AIC214">
<output id="AIC215" name="AIC215" outputContent="../../assets/images/interface/
transparent.gif />
</abstractIndividualComponent>
<abstractIndividualComponent id="AIC217" name="AIC217">
<input id="AIC217" name="AIC217" actionType="SelectTravelHours" actionItem=
"attribute value" dataType="integer" inputMaxCard="1" inputMinCard="1">
<selectionValue name="00" />
<selectionValue name="01" />... </input> </abstractIndividualComponent>
<abstractIndividualComponent id="AIC218" name="AIC218">
<input id="AIC219" name="AIC219" actionType="SelectTravelMinutes" actionItem=
"attribute value" dataType="integer" inputMaxCard="1" inputMinCard="1">
<selectionValue name="00" />
```

Chapter 9 Validation

```
<selectionValue name="05" />... </input> </abstractIndividualComponent>
```

HTML code

```
<input type="radio" name="DateModee" value="depart" checked>Départ  
<input type="radio" name="DateModee" value="arrive" >Arrivée
```



UsiXML CUI

```
<radioButton name="radioBt_215" id="radioBt_215" groupName="DateModee"  
defaultState="checked" />  
<textComponent name="text_216" id="text_216" defaultContent="Départ"  
textSize="8" texFont="sans-serif" textColor="#000000" />  
<radioButton name="radioBt_217" id="radioBt_217" groupName="DateModee" />  
<textComponent name="text_218" id="text_218" defaultContent="Arrivée" />
```

UsiXML AUI

```
<abstractIndividualComponent id="AIC231" name="AIC231">  
<input id="AIC232" name="AIC232" actionType="SelectDateModee"  
actionItem="attribute value" dataType="string" inputMaxCard="0" inputMinCard="1">  
<selectionValue name="Départ"/>  
</input> </abstractIndividualComponent>  
<abstractIndividualComponent id="AIC233" name="AIC233">  
<output id="AIC234" name="AIC234" outputContent="Départ" />  
</abstractIndividualComponent>  
<abstractIndividualComponent id="AIC235" name="AIC235">  
<output id="AIC236" name="AIC236" outputContent="Arrivée" />  
</abstractIndividualComponent>  
<abstractIndividualComponent id="AIC237" name="AIC237">  
<input id="AIC238" name="AIC238" actionType="SelectDateModee"  
actionItem="attribute value" dataType="string" inputMaxCard="0" inputMinCard="1">  
<selectionValue name="Arrivée"/>  
</input> </abstractIndividualComponent>
```

The first element is a simple text node which is derived into a `textComponent`. The two drop down list boxes (two `<select>` tags) are abstracted into `comboboxes` in UsiXML. Each choice, represented by the `<option>` tag in HTML, is translated by an `<item>` element appended as child of the `combobox` in the CUI model.

The second part is composed of four components, two `textComponents` and two `radioButtons`. The `name` attribute in the HTML code is translated into the `groupname` attribute of `radioButtons` in UsiXML. As the first radio button in HTML has the `checked` attribute in its attributes list, its abstract representation also possesses the `defaultState` attribute set to `checked`.

At the AUI level, the “heure” label is derived in an AIC with an `output` facet. `Select` nodes are reverse engineered as two AIC equipped with an `input` facet to represent these two drop down list boxes. The `input` facets possess five specific attributes `inputMaxCard="1"` `inputMinCard="1"` `actionType="SelectTravelHours (minutes)"`, `actionItem="attribute value"` and `dataType="integer"`. The `inputMaxCard` and `inputMincard` attributes define the number of items that the user can pick for this input element. Both take the value “1” as it is derived from a `select` tag without the `multiple` attribute. `ActionType` is composed of the type of input, a selection action, and the HTML’s `name` of the components (`TravelHours`). The fourth attribute is automatically generated, and signifies that this input will set a value to an attribute

(which is the case for every `<input>` node except `<input type="button">`). The last attribute is the type of data that this attribute will hold. The `<option>` elements from the HTML source code are derived into `<selectionValue>` elements at the AUI level. These elements have an attribute `name` that contains the text node of the HTML's `option` tag, and are appended as children of the `input` elements.

The radio buttons in the second row are also derived into AIC with an `input` facet. As the HTML code follows a structure of the type label-radio-label-radio, the AUI specification is composed of four AIC equipped with an `output` and `input` facet alternatively. The two inputs have exactly the same attributes: `actionItem= "attribute value"` which is automatically added, the `dataType` attribute is set to a "string" value as the choices let to the user contains non-numeric values, and the `actionType` is set to "selectDatemodee" which is the `name` of the `<input>` tags in the HTML code. For these elements, minimal and maximal cardinalities are 0 and 1. Each `input` is the parent of only one child `selectionValue` element ('depart' and 'arrive' respectively).

The AUI model of this second example has been used as input in IdealXML (see appendix E) to illustrate by a graphical representation the output of this case study (see figure 9-15). The figure represents the eight AIC of the previous specification (one AIC representing a blank image –AIC214– is not shown in the code excerpts), displayed on two different rows which are represented by `abstractContainer`. These two rows are themselves embedded in another `abstractContainer` (containers AC210, AC211 and AC231 have been removed in the code of the last example).

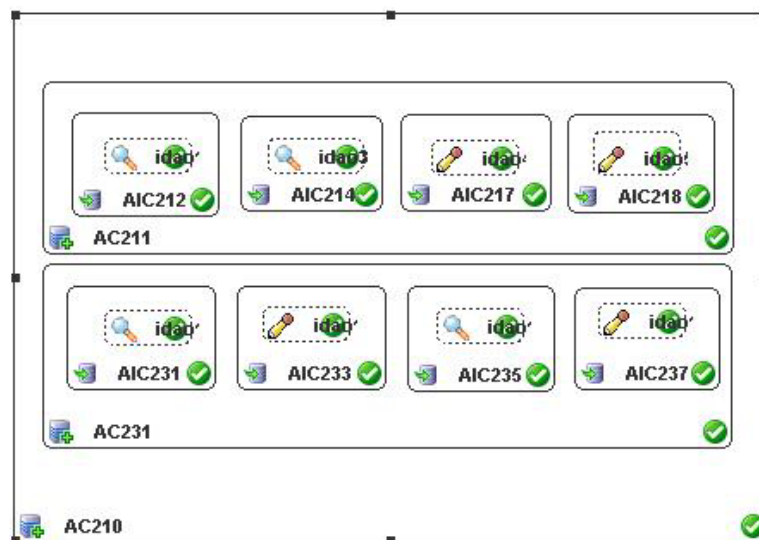


Figure 9-15 AUI model of the case study in IdealXML

9.2.5 Conclusion for external validation

The external validation is divided into two distinct parts, an exploratory study and the presentation of several case studies. These studies replaced an evaluation by UI designers, as the resources required for this kind of investigation are too important (costs, availability of designers...)

The exploratory study results can be characterized by the following properties: the flexible reverse engineering was very helpful for students, as it saved lots of efforts and time in the reengineering process (especially compared to the task of generating the UI for pocket pc and mobile phones from scratch) thanks to the production of correct and predictable results. Predictability is defined here as the possibility to foretell the results of the application of a specific reverse engineering (i.e. with a given configuration of derivation rules). The reverse engineering model could not be used directly in a forward engineering tool as student had to apply some transformations manually, but retargeting made the process easier by avoiding deleting elements and attributes manually and by applying already some transformations. However, several modifications requiring human analysis had to be achieved before the forward engineering phase.

Case studies illustrated the reverse engineering of the most common elements belonging to the HTML language thanks to the analysis of various Websites. It also showed concretely some limits of the tool, such as the loss of a part of navigation by ignoring JavaScripts and Flash components (shortcomings of ReversiXML are discussed in details in section 6.4.5). The results of the first case study were also used to apply a complete reengineering with two different tools, Teresa and QtkXML. The results of the reengineering were acceptable as the produced UI were very similar to the original ones. Teresa produced a UI in XHTML designed for pocket PC and QtkXML generated several UIs in the Qtk format. Some errors/shortcomings still remain in the produced UIs, but the losses are very minor and usual users of the Website would recognize the original UI instantly.

The second case study emphasized the flexibility and controllability of the approach. It was realized by illustrating and comparing the results obtained with a full reverse engineering and a reverse engineering customized for mobile platforms. The case study showed how the box structure, elements and attributes were simplified in the lightened model, reducing the model size to 33% of its original size. Such a simplification would help designers greatly by reducing the cognitive load and amount of work needed to accomplish this step as only elements/attributes available on the target platform are kept in the model.

The last case study focused on the comparison of the AUI and CUI levels, demonstrating the tool capability to recover models at different levels of abstraction (the AUI level is not completely implemented for the moment). The case study demonstrated the feasibility of a complete reverse engineering at this level.

9.3 Comparison with the state of the art

9.3.1 General comparison

This section compares the current approach and the state of the art of the Web reverse engineering (section 3.2) following five properties, the abstraction level, the output languages, the flexibility/controllability of the process, the human intervention and the fact that the process is achieved statically or dynamically.

Abstraction level. First of all, the abstraction levels reached by our approach are the CUI and AUI, and none of the current approaches is able to recover the AUI level. However, WebRevenge, RE of Click's end-user applications and Tamex are able to reverse engineer a model at a higher level of abstraction, the task model, but in Webrevenge and Click's RE, it is only achieved for a redocumentation purpose or for the evaluation of the usability of the Web site. Tamex exploits this level to merge the information contained in several Web pages into one resulting UI. Several approaches stay at the FUI level (Digestor, MTA, ADAPT, the transcoding approach to VoiceXML) as these approaches translate UI code into another UI code without any abstraction. Approaches such as AWT2XIML, Revangie, ReWeb, PIMA, WARE and RE of Click's end-user applications are able to recover a model at the CUI level, which is the most common level reached by reverse engineering approaches in MBA.

Output languages. The results of our approach can be reused in a variety of other tools (see appendix E which is not exhaustive), and more particularly with other forward engineering tools, allowing to produce Qtk (with QtkXML) and XHTML (with Teresa) code, and in a near future Java and XUL (the code generation in these languages is currently implemented in Grafxml). Several approaches are redocumentation tools, such as WARE, WebRevenge, the RE of Click's end-user applications or Revangie, allowing only an indirect reuse of the results. The other approaches have a limited scope, as MTA generates WML, AWT2XIML produces simplified AWT UI, the VoiceXML transcoder outputs VoiceXML, and ReWeb, Digestor, Tamex and ADAPT are only able to generate HTML. PIMA is maybe out of this remark, but this project is not advanced enough to define its scope. As stated

in chapter 3, the reusability of the results of these tools is questionable, as these approaches are implemented for a limited scope of input and output languages.

Flexibility and controllability. Our approach is characterized by a high level of controllability of the reverse engineering process: each reverse engineering derivation rule and retargeting operation can be selected by the designer. The vast majority of reverse engineering approaches abstracts the source UI without alternatives. However flexibility is introduced in some approaches such as MTA, ADAPT and Digestor, but in those cases, the designer can only control few groups of rules, the controllability of the process is not achieved to the same extent than in our approach.

Human intervention. The level of human intervention for ReversiXML can vary between no intervention, in an automatic process with a selection of retargeting rules based on the connected platform, and a low level, as the designer can fine tune each reverse engineering rule before he launches the process. Moreover, as shown in section 9.2.1, the model should be edited afterwards before the forward engineering process in order to obtain UI of better quality, and therefore, we can consider that the level of human intervention required by our approach ranges between none and moderate. This property is similar to the approach developed in Revangie, in which the level vary between no and high as the designer can let the tool automatically produce results or follow and intervene in each step of the process. As in ReversiXML, the quality of the results is often proportional to the level of intervention.

For several approaches (MTA, Digestor, VoiceXML transcoder, the ADAPT and the VoiceXML transcoder, WebRevenge), no human intervention is needed and the results can directly be used on the target platform. In PIMA and the RE of end-user applications, a low level is needed: PIMA requires a feedback mechanism allowing the designer to resolve ambiguities in the inference of semantic information and the RE of end-user applications requires some user inputs to record interaction traces. ReWeb, WARE and Tamex need a moderate human intervention, the user has to specify the inter-pages transformations in ReWeb, use cases have to be defined manually in WARE and the domain extraction step has to be achieved by the designer in Tamex. Finally, AWT2XIML needs a high level of human intervention, as the designer has to code some parts of the target UI manually.

Static/dynamic. The approach developed in this thesis has been designed to be used statically and dynamically, similarly to WARE and PIMA. This has been done to

let the possibility to use the tool automatically at run-time, or manually at design-time. The majority of the other approaches are dynamic approaches, as they apply the reverse engineering (or reengineering/transcoding) on demand at run-time. This is due to the fact that these approaches require none or a low level of human intervention, that the controllability of the process is almost inexistent and that the problem they solve is to make a requested HTML UI accessible to another (class of) platform(s), without having to reverse the Web site entirely. Therefore, the dynamic process is more appropriate for these approaches.

Four approaches can only be used statically (WebRevenge, ReWeb, RE of Click's end-user applications and AWT2XIML), as all these tools except AWT2XIML, produce documentation and the dynamic aspect would not bring benefits to the process. The two other redocumentation tools (Ware and Revangie) need dynamicity as they target dynamic form-based Websites, in which user inputs modify the UI drastically.

9.3.2 Comparison with a transcoding approach

This section compares some properties of our approach and the transcoding approach in ADAPT (section 3.2.8). This transcoding process allows regenerating HTML pages adapted for a PDA or a similar mobile platform. One of the strengths of the approach is that the system allows choosing the navigation pattern for the new system, providing thus some flexibility. Some of the navigation patterns are shown on figure 9-16 and examples for the summary pattern (left) and tree pattern (right) on the CNN Website are illustrated on figure 9-17.

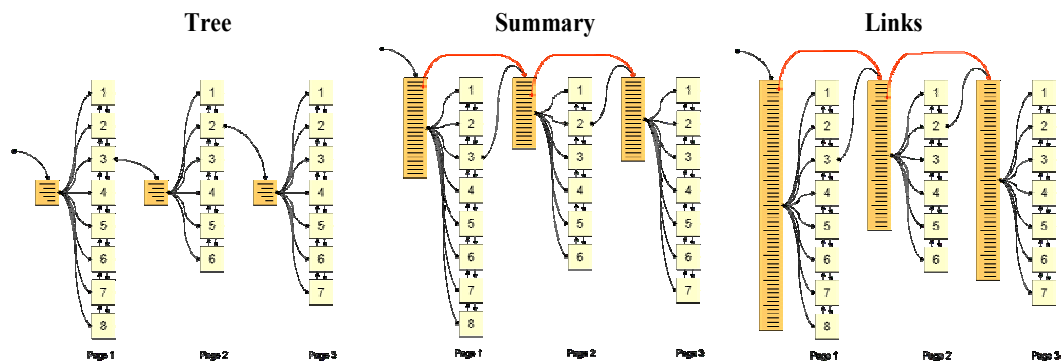


Figure 9-16 ADAPT's navigation patterns

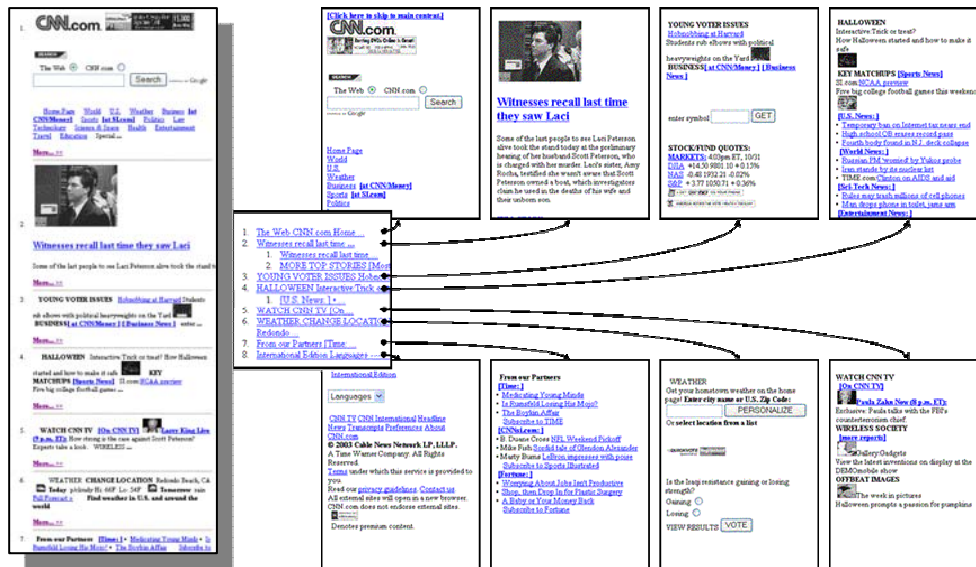


Figure 9-17 Navigation patterns on the CNN Website

Other transformations applied by the tool are mainly composed of text elision, image reduction/transformation, layout modification and redistribution of the UI on several windows according to the navigation pattern.

As the two approaches produce different results - a concrete or abstract UI for ReversiXML and a set of HTML pages with ADAPT -, they are not directly comparable. However, a comparison based on the shortcomings of the approaches, on the performance and the size reduction of the original UI can be achieved.

Limitations. Table 9-3 summarizes the limitations of both approaches. Similarly to ReversiXML, the process of adaptation can not handle scripts, cookies, complex style sheets (whereas ReversiXML do not processes style sheets at all) and absolute positioning. This last limitation could be included in the style sheet support limitations, as absolute positions can only be specified in style definitions. Flash components are simply ignored in ADAPT, but they can be recovered in ReversiXML. ReversiXML can not reverse engineer pages with limited accessibility (i.e. users have to connect the site with a standard browser, the server do not allow Web crawlers/spiders to access the Web site) whereas ADAPT overcome this limitation. Finally, ADAPT supports forms and frames partially, as the tool may generate errors for pages containing those types of components. ReversiXML supports these two types of elements during the reverse engineering without errors.

Chapter 9 Validation

ADAPT	REVERSiXML
Javascripts	Not supported
Required use of cookies	Not supported
Flash components	Supported
Complex style sheets (.CSS)	Style sheets not supported
Absolute positioning	Not supported
Supported	Pages with restricted accessibility
Partial limitation with frames	Supported
Partial limitation with forms	Supported

Table 9-3 Limitations of ADAPT and RevesiXML

As one can see, the limits related to the elements/concepts not covered by the approaches are almost similar.

Performance. The performance analysis is based

- For ADAPT: on the time required by the tool to download the code of the Web page and to adapt this page into a Web page suited for a PDA thanks to XSLT transformations.
- For ReversiXML: on the time needed to download the Web page, save it on the server, load the configuration file (if any), apply the reverse engineering accordingly, save and display the results of the reverse engineering.

The two processes are thus quite different, in the steps they follow and in the outputs produced by the two tools. The time needed to generate the HTML code is less than one second, but ADAPT also adapts the original images (it converts the image to a supported format, reduce the number of colors and scales it down), and by adding the time needed for these operations, the total time required is about 5 seconds (20 seconds if ADAPT do not use a caching mechanism). This time is comparable to the time required by ReversiXML to generate a CUI model suited for mobile platforms (5.46 seconds in average, in the last column of table 9.1). However, this model still needs to be forward engineered, thus we can conclude that the time required to reengineer the UI completely by our approach is at least two times longer (and would be probably more than that). This is not a surprise, as transcoding approaches are designed to support only one output language, and thus the process is optimized for this purpose. The time required to process the original UI is also one of the most important properties of transcoding approaches, whereas in MBA this factor is less important (this feature could still be enhanced in ReversiXML).

Size reduction. As we can not compare directly the size reduction realized by ReversiXML and ADAPT, we will use the following simplification: the size reduction with ADAPT will be measured by the difference between the adapted page and the original page and for ReversiXML, the measure will be the difference between a model produced without retargeting, and one with retargeting. The size reduction in ADAPT is dependent on the type of navigation pattern chosen. Table 9-4 contains the comparison between navigation pattern producing the minimal and maximal size reduction, and the average size reduction with ADAPT. The last column is the size reduction achieved by ReversiXML between the two CUI models (which has already been measured in table 9-1).

	Min. ADAPT	Max. ADAPT	Average ADAPT	Average ReversiXML
Size reduction	35%	90%	70%	25%

Table 9-4 Size reduction with ADAPT and ReversiXML

As shown in this table, the extent of the size reduction varies greatly, as it is only 35% with the single page navigation pattern and 90% with the tree navigation pattern. The average between the six navigation patterns is about 70 % of the original HTML page whereas the size reduction of ReversiXML is around 25%. Thus the size reduction achieved by ReversiXML is a bit more than 1/3 of the ADAPT reduction process, but as stated in the performance analysis, the ADAPT transcoding is only targeted for PDA UI, and is therefore optimized for this objective. In contrast, ReversiXML's flexible reverse engineering was designed to accommodate other types of target platforms, and thus retargeting is obviously less efficient than for a dedicated approach. Moreover, these percentages must be interpreted cautiously as they are not based on the same outputs.

9.3.3 Conclusion of the comparison with the state of the art

This section compared the approach presented in this thesis with the approaches of the state of the art. There are two major differences with the current approaches:

- the *level of flexibility and controllability* of our approach is not reached by any other approach, even if some of them let the designer select between some alternatives in the output of the process (e.g. the navigation patterns of ADAPT).
- The *scope* of our approach, combined with the various UsiXML forward engineering tools, is more extended than the current reverse engineering/reengineering or transcoding approaches.

The deepened comparison of our approach with the ADAPT tool allowed us to state two conclusions, firstly that the limitations of the two approaches are similar, as the concepts/elements not analyzed are almost identical. No solutions were found in ADAPT to process them too, reinforcing the idea that the trade-off between cost and coverage of these elements is disproportionate and that these elements should be the subject of a separate study.

Secondly, that ADAPT is more efficient than ReversiXML if we achieve a comparison of these two approaches based on technical parameters (size reduction and process time). This conclusion is normal as ADAPT has been designed and optimized to transcode HTML to be rendered on PDA screen only, whereas the flexible process of ReversiXML can be modified to target a variety of platforms.

9.4 Conclusion

This chapter validated the approach theoretically in the first section, by evaluating the coherence, performance, correction and coverage of the reverse engineering of HTML UI. This allowed us to show that the objective of controllability, flexibility, generality and predictability are theoretically fulfilled by the approach.

The second section validated the approach empirically, thanks to case studies and an exploratory study, allowing us to demonstrate various techniques of reverse engineering applied on different web sites and that the flexible reverse engineering was useful for the users, as it saved lots of efforts and time in the reengineering process.

Finally, our approach was compared with similar studies/tools aiming the reverse or reengineering of HTML pages. On this subject, it can be stated that our tool has several advantages over other tools: the flexibility of the process is not reached by any other approach, and the use of the retargeting concept allows combining some advantages of MBA and transcoding. Indeed, rather than applying “context-free” derivation rules, rules can be selected to accommodate a particular context of use and thus benefit from a gain of precision for the produced abstract model, while preserving an abstract layer. On the other hand, transcoding tools produce results that can be rendered on the targeted platform without any further modification (i.e. maximal precision in the transformation), but by the fact that no abstraction is achieved, the results and the approach can not be reused for another purpose. As shown in section 9.3.2, performance and efficiency is superior for these tools as these parameters are of first importance for transcoding approaches.

The same generality limitation has been observed with MBA tools as most of these tools have a limited scope. One or two contexts of use are targeted by these

approaches, and the underlying abstract models/languages have been most of time achieved for a single purpose (“private” models). For the purpose of reengineering, our approach has thus some advantages, but approaches such as Reweb, WARE or Click’s RE offer more possibilities for maintenance and documentation. These approaches propose specialized features for these objectives, as they allow to compare various versions of the Website, or to have a global overview of the website structure (database links, navigation plan ...).

A final conclusion can be made about difficulty to generalize the tool support for reverse engineering, by observing the amount of coexistent reverse engineering tools. Indeed, this amount demonstrate the impossibility to produce a generic reverse engineering tool – it does not exist in the scientific or commercial domain – as the variety of modalities, interaction techniques, platform capabilities and objectives of the process itself imply that the reverse engineering problem is very difficult to generalize.

Chapter 10 Conclusion

This chapter concludes this thesis, by summing up the main contributions of this research (section 10.1), discussing its results (section 10.2) and finally by presenting some relevant future work which could bring this problem to new frontiers of solutions (section 10.3).

10.1 Contribution

The main contributions of this thesis can be classified into four categories defined according to the fundamental aspects they handle [Long96]:

1. In the conceptualization:

- The development of a flexible and dynamic method thanks to a model-based approach addressing the problem of declarative UI reverse engineering process for the case at hand, and permitting the reuse of the produced models for the generation of several UIs at a level of generality that is not reached by other approaches.
- The positioning of this approach in a conceptual framework elaborated in the Cameleon research project (section 2.1), which has been continuously be

used throughout this thesis, as well as for defining concepts, notations, and rules, but also for the state of the art.

- The adaptation of the reverse engineering definitions to the UI domain and the positioning of these definitions in the conceptual framework (section 2.2).
- The development of the retargeting concept (section 2.3) and some retargeting rules (appendix G), a new method for the reverse engineering of user interfaces that uses design knowledge particular to platforms to select the most suited reverse engineering derivation rules [Boui02b].
- The constitution of a state of the art in reverse engineering for legacy UI (section 3.1) and for HTML files (sections 3.2), concluded by an analysis of limitations of the current approaches by observation.
- The identification of shortcomings of the output language to represent important information from the analyzed UIs (graphical and vocal) that led to modifications of the UsiXML language.
- The specification of meta-models illustrating the complete structure of XIIML, HTML, WML, VoiceXML, Windows resources files in UML class diagrams, allowing a common representation of the source and target languages (appendix A and B). Note that most of these meta-models did not exist before (at least, we were not aware of any such meta-models) and that this already represents some contribution per se, independently of the UI reverse engineering. But this step was required in order to establish semantic correspondences between the meta-models of the source and the target languages (e.g., HTML and XIIML, HTML and UsiXML, WML and UsiXML, VoiceXML and UsiXML).
- The development of derivation rules based on these meta-models for HTML (section 6.3), WML (section 7.1.3), VoiceXML (section 7.2.3), Windows resource files (8.3) completed by appendix C. Derivation tables for HTML towards XIIML presentation model were also described in section 5.3.

2. For the operationalization:

- Vaquita [Boui01, 02a, 02b, 2c, Vand01], a tool (illustrated in chapter 5) allowing the static flexible reverse engineering of HTML files into an XIIML presentation model.
- ReversiXML [Boui05], an online tool described in chapter 6, allowing the flexible reverse engineering of HTML files into a CUI or AUI model expressed in a second language, UsiXML. The tool is able to detect some

parts of the context of use, allowing the selection of the best configuration file based on this information.

- A prototype (implemented by a student in [Cui05] and presented in chapter 7) of reverse engineering tool able to extract a CUI model from WML files.
- A set of guidelines [Mari05] for the implementation of a plug-in incorporated in GrafiXML to reverse engineer Windows resource files.

3. In the test phase:

- The validation of the proposed method against four criterion (coverage, correction, coherence and performance) allowing validating theoretically the thesis statement (section 9.1).
- An exploratory study realized by 17 students describing a qualitative evaluation of the produced models, the tool, and the advantages of using retargeting in a reengineering process (section 9.2.1). This investigation, combined with case studies, completed the external validation of the thesis statement.
- The illustration of the different reverse engineering techniques elaborated for HTML on three case studies of various complexities (section 9.2.2) so as example of the application of derivation rules for WML, for VoiceXML and for Windows resources files (appendix H).
- A comparison of the approach presented in this thesis with the ADAPT transcoding approach (section 9.3.2).
- An investigation of the full reengineering by combining the results of ReversiXML with TeresaXML and QtkXML (section 9.2.2). Reengineering in a virtualization process is also described in section 5.1.4. This process produces a VRML file based on XIIML specification that was recovered from an HTML file [Boui04].

4. For the last step of generalization:

- A semi-formal notation allowing expressing reverse engineering derivation rules (chapter 4 and appendix C) for various source UIs.
- A generalization of the reverse engineering method thanks to an analysis of the process applied to a large scope of UI and the identification of common sub-problems to be achieved in order to complete UI reverse engineering (section 4.6).

10.2 Discussion

10.2.1 Thesis statement

The thesis statement is re-examined and commented in this section. The thesis statement is that “*the application of the reverse engineering at a higher level of abstraction than the code level supports UI reengineering with flexibility while preserving predictability, more generality and controllability, in the process than with code-to-code (transcoding) approaches or current reverse engineering approaches.*”

State of the art (chapter 3) allowed us to identify shortcomings of current approaches. The approach, the model, and the supporting tools presented in this thesis are different from existing work in the sense that it does not consider the pair fixed (source platform, target platform). Many models and tools exist that translate one UI to another one or multiple UIs for fixed pairs. When there is a need to consider multiple target computing platforms, the ad hoc approach no longer remains a viable approach. Therefore, manipulating the UI at a higher level of description than merely the code level is to be expected.

Moreover, platform to platform tools usually support limited conversion and application of rules and heuristics in a very rigid way. In contrast, we presented an approach that reverse engineers web pages (or other types of UIs) to a level where it can be regenerated for itself or for other computing platforms.

Advantages resulting from composing UI reverse engineering, logical translation, and forward engineering are:

- *Generality*: starting from a UI specification expressed in various languages, a set of heuristics and derivation rules are selectable, allowing designers to explore alternative design options that would be otherwise impossible or hard to cover at the code level. Similarly, once an abstract UI is obtained, it can be submitted to any set of transformations in the logical translation thanks to TransformiXML (appendix E) to accommodate to the target platform. This process is no longer specific to any source-target pair. It could be argued that the translation achieved by TransformiXML could replace retargeting operations, but the tool is only able to process translation based on UsiXML specification, whereas ReversiXML can use additional information coming from the source language (and not covered by UsiXML) to apply retargeting.
- *Flexibility*: the reverse engineering process can be parameterized to reverse engineer only those UI elements of interest and rejecting those not concerned.

Chapter 10 Conclusion

It allows reducing the model's size and complexity, thus facilitating the translation process for the designer.

- *Controllability*: the developer can control the process both in the reverse engineering and in the translation or during the composition of these two processes, the retargeting. This allows to fine tune the process precisely thanks to a transparent system, and this at every step of the reengineering, rather than having a “black-box” reengineering in which the designer can not change anything and obtain fixed results.
- *Predictability*: thanks to coherent derivation rules producing correct models, the designer can foretell the results of the application of a specific reverse engineering process (i.e. with a given set of derivation rules). In forward engineering, it is possible to improve this predictability property by adding a preview of the UI that will be generated thanks to the models. It is difficult to realize the same with reverse engineering. An illustration of the possible design option could be added to the current approach, but it would be intrinsically hard to achieve a similar predictability functionality in a reverse engineering process.

10.2.2 The reverse engineering sub-problems

Derivation rules have been grouped by objective in section 4.6 in order to identify common reverse engineering sub-problems to be achieved to recover a complete CUI model in UsiXML. Table 10-1 describes the scope of the reverse engineering in terms of these sub-problems for the different source languages/tools of this research.

	Elements	Attributes	Dialog	Layout	Hierarchy	MultiTree	Retargeting
Vaquita	✓	✓	✗	✗	✓	✗	+
ReversiXML	✓	✓	+	✓	✓	✓	+
WML	✓	✓	+	✓	✓	✓	✗
VoiceXML	✓	✓	+	✓	✓	✓	✗
RC files	✓	✓	✗	✗	✗	✗	✗

✓ = totally covered

✗ = not covered

± = partially covered

Table 10-1 Coverage of the reverse engineering sub-problems for the different source UI

These sub-problems are the detection of UI elements, the recuperation of attributes, the recovery of the general layout or the temporal sequence of the UI, the capture of dialog, the hierarchy identification, the binding with other models (multiple trees

Chapter 10 Conclusion

transformations) and retargeting operations. This last category is special in the sense that it is not needed to recover an abstract specification from a declarative UI, but was added as it allows to obtain an abstract model for another context of use and several derivation rules are defined in this research to achieve this aim. Three cases are possible for each dimension: either the dimension has been entirely covered, partially covered or not covered at all.

The first approach, concretized by the implementation *Vaquita*, is a simplified reverse engineering, as it allows a partial recovery of the presentational aspects of the UI. Dialog, layout and multiple trees (files) sub-problems are not achieved, but some retargeting operations are already applied in this study.

The second approach, operationalized with *ReversiXML*, covers all the reverse engineering sub-problems, but two of them are incompletely covered: dialog, as imperative languages embedded in HTML allowing to put dynamicity in Web pages are not analyzed, and retargeting operations, as this kind of operations has been defined for several cases but could be extended significantly.

Reverse engineering of *WML* and *VoiceXML* share similar properties. Both approaches cover the entire reverse engineering problem, but dialog is only partially recuperated for the same reason than for the HTML. Moreover, control flow in VoiceXML can be modified thanks to tags defined in the language (no scripts are needed for standard operations), and all this information can only be covered partially in the CUI model. This is due to the fact that conditions are hard – if not impossible- to express in UsiXML, as one of the methodological choices realized when designing the language was to cover only declarative parts of the UI. Retargeting issues were not covered for these two analyses.

Finally, the study of the *resource script (RC)* reverse engineering was, as for *Vaquita*, simplified as only elements and attributes categories were treated. Dialog and multiple trees transformations are out of scope, as they are not recoverable thanks to a static analysis of the resources files. Layout and hierarchy detection sub-problems need a very complex solution to be achieved, as absolute coordinates are used to describe element positions and several burdens complicate the problem (see section 8.1). The retargeting operations were also not covered in this approach.

Thanks to this table, the various sub-problems of the reverse engineering can be ranked following their intricacy, by analyzing the least covered subjects. The most complex reverse engineering sub-problem is the recovery of the dialog components of a UI specification. This is due to two difficulties: firstly dialog is often specified in

Chapter 10 Conclusion

another language or in the functional core of the application, thus not reachable or not covered by the working hypothesis of this research, and secondly it is hard to specify the behavior of UI at an abstract level. Indeed, the limit between dialog belonging to the core of an application (i.e. implementing functionalities of the information system) and dialog specific to the UI is difficult to define, and abstracting UI behavior for several types of source languages is a complex task without falling in a too descriptive specification of the dialog (i.e. reinventing a script language).

Another reverse engineering sub-problem that was not fully achieved in most of the approaches is the retargeting sub-problem. Some operations of this type have been defined and implemented as a proof-of-concept, but this type of research is also conducted in [Flor06] and could be integrated in the current approach.

Finally, the recovery of the layout and hierarchy of UI components can be considered as relatively difficult sub-problems as it requires complex algorithm to be achieved.

10.2.3 Results of reengineering

Several complete reengineering were applied thanks to results of the approach designed in this thesis. Complete reengineerings are mentioned in table 10-2 which contains the tools and the target language of these reengineerings.

Vaquita + Envir 3D	VRML
ReversiXML + Teresa	XHTML for Pocket PC
ReversiXML + QtXML	QTK
ReversiXML + Grafxml	UI preview
WML RE + Grafxml	UI preview

Table 10-2 Complete Reengineering

The first reengineering was achieved by using Envir3D to generate VRML UIs (chapter 5). Several modifications had to be applied to the model produced by Vaquita to accommodate the 3D in XIML, i.e. by adding 3-dimensional positions to elements in the specification. This was done manually, but this translation step could be easily automated. ReversiXML was used in conjunction with three other tools, QtXML to produce Qtk UI code, Teresa to create XHTML UI designed for mobile platforms (chapter 9). The results produced by ReversiXML could be reused directly in QtXML while the results were modified in Teresa in order to redistribute widgets

on several windows. ReversiXML's output was also used with Grafixml (section 9.2), but only to have previews of the UI corresponding to the model, as the tool is not able to generate final UI code. The same was realized with the prototype of WML reverse engineering tool (see section 7.4).

These five different cases allowed us to demonstrate the feasibility of a UI reengineering based on the flexible reverse engineering presented in this dissertation.

10.2.4 Limits of the approach

The approach presented in this thesis suffers from several limitations and the most important of them are listed here.

The retargeting concept offers some benefits already discussed in chapter 2 but introduces also limitations at the conceptual level, i.e. a reduction of generality and maintainability of the models produced compared to a traditional approach and MDA. The separation of concerns, i.e. the fact that abstraction and translation are distinct and cleanly separated processes, is also lost, so as the independence principle. Therefore, the possible reuse of the produced models for other purposes than the first(s) targeted UI is reduced by using this technique.

The notation is limited for declarative UI only, that can be represented in a tree structure and using a straightforward control flow. For example, the Windows resource files are not structured as trees, and have to be modified so that they can be represented in such a type of structure. But other cases could exist were such transformations are impossible, and thus the notation would not hold for those cases.

The dialog aspects of UI are not recovered in this thesis, as stated in section 10.2.2. This is an important limitation of the current approach, as some significant parts of the UI are sometimes only available through a dynamic behavior of the UI. Therefore, the proposed approach is more adequate for the reverse engineering of static UI.

The automation of the reverse engineering process produces UI that can be used in forward engineering to produce a new UI, but maybe not a usable UI. Therefore, models generated by our approach should be edited afterwards to enhance the quality of the final UI. External validation showed us that several modifications had to be applied afterwards to obtain a UI usable on mobile phones or pocket pc. This observation holds also for most of the approaches of the state of the art, as a large amount of the tools require a human intervention to refine the results of the automated process.

The graphical and vocal modalities dependence of the sub-problems identified in section 4.6, as these categories are particularly relevant for these modalities. Other modalities, such as 3D UI or tactile UI may be decomposed into other types of sub-problems. It does not mean that the identified categories would all be invalid, but they would be probably augmented or modified according on the targeted type of modality.

The static reverse engineering of UI implies a loss of information, as some part of the model could only be recovered by a dynamic analysis. For example, the conclusion of chapter 8 shows some limitations of static analysis. In this example, some part of the UI are specified in the functional core (such as button's labels, images, ...) and can not be reached by a static analysis. To overcome this limitation, a dynamic analysis should be used, or technique similar to TAP, to recover the missing information at run-time.

10.2.5 Compliance with MDA

MDA (model-driven architecture) is a major trend in software engineering. It regroupes several important organizations (such as SAP, Nokia, Motorola, some universities ...) and aims at the standardization of methodologies and specification based on models. MDA supports the development of complex, large, interactive software systems providing a standardized architecture. MDA do not focus particularly on UI, but on all the various components of information systems.

Four principles underlie the OMG's view of MDA [Moll04]:

1. Models are expressed in a well formed unified notation and form the cornerstone to understanding software systems for enterprise scale information systems. The semantics of the models are based on meta-models.
2. The building of software systems can be organized around a set of models by applying a series of transformations between models, organized into an architectural framework of layers and transformation.
3. A formal underpinning for describing models in a set of meta-models facilitates meaningful integration and transformation among models, and is the basis for automation through software.
4. Acceptance and adoption of this model-driven approach requires industry standards to provide openness to consumers, and foster competition among vendors.

Even if our approach and the reference framework does not conform exactly to the MDA recommendations, which suggest the use of standards such as Corba or UML to describe models, the compliance of our approach with MDA can be shown.

Chapter 10 Conclusion

Firstly, for the quoted principles, the compliance is achieved by the following way [Vand05]:

1. UsiXML models are well-formed models based on XML schema. The semantics of the models are based on meta-models expressed in terms of UML diagrams, from which the XML schema is derived.
2. Model-to-model transformations are specified in UsiXML. Model-to-code transformations are ensured thanks to a set of forward engineering tools (see appendix E). Code-to-model transformations are achieved by derivation rules (see chapter 4) that are based on the mapping between the meta-model of the source language and the meta-model of UsiXML.
3. All transformations are explicitly defined, based on a series of predefined semantic relationship and a set of the primitive ones (abstraction, reification and translation).
4. The last principle is on the way, as the potential wide adoption of the above techniques will validate the principle.

Levels of abstraction of the reference framework are also compliant with the four levels of abstraction, or development steps, in MDA (figure 10-1): Task and domain level is similar to the computing independent model (CIM) - which is sometimes called domain model - as they are stated independently of any implementation of any interactive systems.

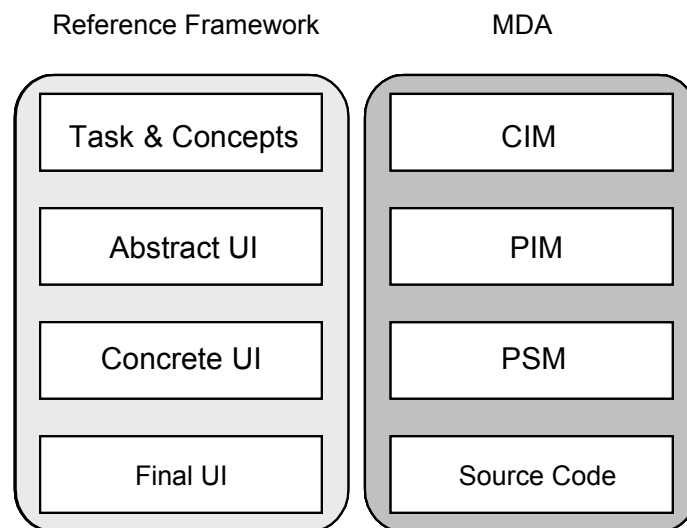


Figure 10-1 Comparison of the reference framework and MDA

Such models could be specified for virtually any type of UI. The platform independent model (PIM) is interpreted as the abstract UI model (AUI) in the

reference framework, in the sense that it is independent of any interaction modality: at this level, we do not know yet whether the UI will be graphical, modal, virtual or multimodal. The platform specific model (PSM) is interpreted as the concrete UI model, as it independent of any vocabulary of markup and programming language. At this level, the modality has been selected, but we do not know which physical computing platform will run the UI. This is why it should be believed that the CUI is not platform specific. Only some aspects of the target platform are selected, the platform being modeled itself in the platform model.

Levels of abstraction are not exactly the same in both approaches, as MDA is intended to model any component of information systems whereas the reference framework purpose is to model the development steps of UI. However, our approach shares enough conceptual properties to be considered as compliant with MDA.

10.2.6 Integration with other works

As stated in introduction, the reverse engineering process is a part of a larger process, reengineering. Therefore, this work can be integrated with several other researches (most of them are conducted at BCHI/UCL). Figure 10-2 illustrates the different other studies or approaches covering the various steps of the reference framework.

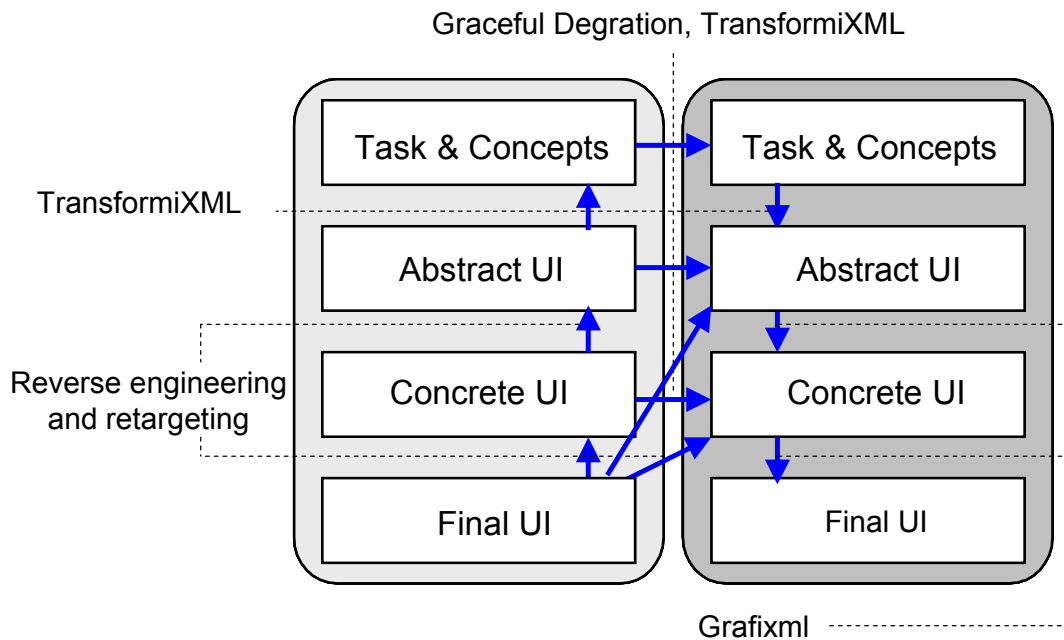


Figure 10-2 Integration with other researches

The reverse engineering and retargeting processes as presented in this thesis corresponds to the arrows starting from the final UI to the concrete UI and abstract UI for the same context of use or for another context of use. Its counterpart, the forward engineering process, is studied in the Graftxml project (see appendix E) and is represented by the reification arrows from the abstract UI or concrete UI to the final UI. Translations from one context to another are symbolized by horizontal arrows. These steps are supported by two researches, TransformiXML (appendix E) and the graceful degradation of UI, an approach developed by Murielle Florins [Flor06]. TransformiXML is about translation and model transformations in general, whereas the graceful degradation approach describes translation rules aiming a more constraining context of use than the source one. TransformiXML can also be used to cover the abstraction from an abstract UI to the tasks and concepts level, and the inverse relation, from the tasks and concepts level to the abstract UI.

Finally, all these approaches can be linked with the research of Q.Limbourg [Limb04c], about the multi-path development of UI, which describes a graph-transformation approach between the different levels/models from the Cameleon reference framework.

10.3 Perspectives

Several future works have been identified and are classified in terms of their desirability and feasibility as short term and long term perspectives:

At short term:

- Extend the coverage of the reverse engineering sub-problems for Windows UI, by developing a method allowing recovering the layout and hierarchy of widgets composing the UI.
- Implement two new tools to reverse engineer VoiceXML and Windows resources files, as all the material has been developed until the tool implementation step for these languages.
- Expand the existing implementation of ReversiXML. The implementation of the tool was stopped with the complete version of UsiXML 1.4.6. The tool should be adapted to the version 1.6.4 of the language (this new version has been released in March 2006)
- Moreover, the architecture of the tool was frozen in the version 1.1 and could be developed until the version 1.2 (see appendix I).
- Integration of new reverse engineering techniques in the current methods, as for example, the use of statistics about the composition of the UI to select

Chapter 10 Conclusion

more accurately the reverse engineering rules and the observation of typical uses of the UI to complete some missing parts of the model (see section 6.4.5).

- Validate the tool and the approach externally thanks to a quantitative study, by asking to designers to use the tool to reengineer web sites. This study should also contain an evaluation of the reengineered sites by Web users. It is fundamental to apply this validation with designers, but as stated previously, their price is very expensive and it is difficult to find a significant number of designers to conduct this study. Moreover, reverse engineering occurs seldom, thus finding designers confronted with this kind of problem makes this type of validation even harder to achieve.

At long term:

- The scope of language analyzed in this research is already acceptable, but could be enlarged to take into account UI coming from other environments such as Macintosh resource files, XUL, Cobol ... This generalization should happen at two levels. For other similar declarative languages (XUL, VB form, etc.), several aspects of the current approach could be reused, as most of changes would occur at the operationalization level. The notation and the reverse engineering sub-problems as defined in section 4.6 should remain the same, while the derivation rules would be adapted to these new types of UI. For imperative languages, such as Cobol, the reverse engineering sub-problems should remain valid, but would be augmented by new categories of rules, such as state detection or control flow recovery. Another issue would also be the link of UI specification with the old functional core. The basis of the notation and derivation rules could be reused, but should also be enhanced with new features specific to this new type of problem.
- Study of the reverse engineering of imperative languages incorporated in markup-languages, in order to be able to complete dialog and maybe other models of UsiXML specifications, by exploring possibilities of abstract representation and pattern matching techniques.
- Integration of more retargeting rules. This part of this research need to be enhanced, by analyzing potential target of forward engineering and adding retargeting operations to facilitate reengineering towards these new platforms. This task can be relatively long as possible targets are numerous, and their number is constantly growing.

Chapter 10 Conclusion

- Complete the reverse engineering of the context model - coupled with an enhancement of the WhoAmI module from ReversiXML- to increase the quality of a reengineering adapting the UI to the context of use (i.e. not only the platform, but also based on the user and environment parameters). The current method would be enhanced by new retargeting operations to take into consideration these new contexts of use.
- Analyze the application of the method developed in this thesis to achieve a reverse engineering towards other approaches/abstract languages, such as UIML. UisiXML and UIML have similar capacities of expression. Therefore, the reverse engineering approach would not differ greatly, as the concepts and reverse engineering sub-problems should remain valid. The notation could also be reused in this objective.
- Explore the reverse engineering of UI at a higher level than the tasks and concepts level defined in the reference framework, such as requirements reverse engineering. This kind of study applied on the UI should not provide the same quality of results than a requirement reverse engineering based on the functional core, or documentation analysis as in [Reve00, Rays99], as the UI source of information is less representative for the recovery of the motivating requirements for the existing functionalities and components of information systems. However this study could be an interesting subject of research, possibly combined with a reverse engineering applied on other parts of information systems (functional core, databases, documentation...).

Long term perspectives for reverse engineering are numerous as it will always be a part of the engineering process of computer-based systems since the evolution of the context of use, and more particularly of the platform parameter, is intrinsically linked with information system (see chapter 1). Therefore, reverse engineering will always be needed. Reverse engineering is thus a very important technique as it facilitates the evolution of information systems, by avoiding starting implementation of the new system from scratch and so reducing time, costs and resources devoted to the implementation of new information systems. Thanks to this process, existing UIs or parts of them can be reproduced on information system using new technologies. But the quality of the results will never be guaranteed totally and permanently, as reverse engineering will always depend on the transformations of context of use since technology is evolving constantly.

Chapter 10 Conclusion

Adaptation to a new technology is not the only case in which reverse engineering can be a valuable process, as the technique can be used for maintenance and redesign of existing UIs at a higher level than the code. It can also be achieved for the reuse of UI as basis for creation, as some parts of specifications can be the starting point for the design of new UI. Reverse engineering can aim at the redocumentation of an information system poorly or not documented. Finally, a methodological advantage is that it also allows recovering abstract specifications from UI that were not designed according to a model-based approach to incorporate them in the same pipe-line and so allowing the standardization of UI specification and development process in an organization.

As we said in the introduction (Section 1.1), information systems will always evolve and so does the reverse engineering problem: as long as new technologies and new contexts of use will appear, there will be instantly new obsolete systems that will be required to evolve. In this global process, the UI is here considered as one component of the entire information system subject to evolution, but not the only one. And as long as there will be some obsolete system, the reverse engineering problem will remain equally to be solved. Outdated interaction techniques have been modeled so that reengineering could occur by abstraction and reification, but perhaps the current interaction techniques will become the outdated interaction techniques tomorrow. The current UI will become archaic in the future, but the upcoming interfaces will be even more complex to model, to manage, and to reverse engineer as they will support many different computing platforms, many different interaction techniques, etc. In this way, it is likely that the main problem of user interface reverse engineering will not become obsolete itself

References

A

- [Addi98] Addison W., History of the html language, 1998, available at <http://www.w3.org/People/Raggett/book4/ch02.html>
- [Astr05] Astrova, I., Stantic, B.: An HTML-Form-Driven Approach To Reverse Engineering Of Relational Databases To Ontologies. Proceedings of the 23rd IASTED International Multi-Conference on Applied Informatics, Innsbruck, Austria (February 2005)

B

- [Bana00] Banavar G., Beck J., Gluzberg E., Munson J., Sussman J., Zukowki D., *Challenges: An Application Model for Pervasive Computing*, In Proceedings of the Sixth Annual ACM/IEEE International Conference on Mobile Computing and Networking (Mobicom 2000), Boston, USA, 2000.
- [Band05] Bandelloni R., Mori G., Paternò F., *Dynamic Generation of Migratory Interfaces*, in proceedings of: "Mobile HCI 2005 (MHCI05)", Salzburg, Austria., ACM Press ISBN:1-59593-089-2, pp. 83-90
- [Barc00] Barclay P.J. and Kennedy J., "Teallach's Presentation Model", Proceedings of the 5th ACM Int. Working Conf. on Advanced Visual Interfaces AVI'2002 (Palermo, 23-26 May 2000), ACM Press, New York, 2000, pp. 151-154.
- [Beau00] Beaudoin-Lafon M., "Instrumental Interaction: an Interaction for Designing Post-WIMP User Interfaces" in proceedings of ACM conference on Human aspects in computing systems, CHI 2000, 1-6 April, Amsterdam, ACM Press, New York, 2000, p446-453.
- [Bick97] Bickmore T.W. and Schilit B.N., *Digestor: Device-Independent Access to the World-Wide-Web*, in Proceedings of 6th World-Wide-Web Conf. WWW'6 (Santa Clara, 7-11 April 1997), Elsevier, Amsterdam, 1997.
- [Bhar05] Bhardwaj Y., *Reverse Engineering End-User Developed Web Application into a Model-based Framework*, Master of Science Thesis, Virginia Polytechnic Institute and State University, May 2005, p.106
- [Bloo] History of the html language, available at <http://www.blooberry.com/indexdot/history/html.htm>
- [Bohe81] Boehm B.W., *Software engineering economics*, Prentice Hall, 1981

References

- [Boeh88]
Boehm B.W., "*A Spiral Model of Software Development and Enhancement.*" IEEE Communications 21, 5 (May 1988), pp61-72.
- [Boui01]
Bouillon L., *Rétro-ingénierie de Pages Web*, Licence en sciences de gestion, Mémoire, Université Catholique de Louvain, September 2001.
- [Boui02a]
Bouillon L., Vanderdonckt J., Souchon N., *Recovering Alternatives Presentation Models of a Web Page with Vaquita*, in Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUP2002 (Valenciennes, 15-17 May 2002), Kluwer Academics Pub., Dordrecht, 2002
- [Boui02b]
Bouillon L., Vanderdonckt J., *Retargeting Web Pages to other Computing Platforms*, Proceedings of IEEE 9th Working Conference on Reverse Engineering WCRE'2002 (Richmond, 29 October-1 November 2002), IEEE Computer Society Press, Los Alamitos, 2002, pp. 339-348.
- [Boui02c]
Bouillon L., Vanderdonckt J., Eisenstein J., *Model-Based Approaches to Reengineering Web Pages*, in Proceedings of 1st International Workshop on Task Model and Diagrams for user interface design Tamodia'2002, (Bucharest, 18-19 July 2002), Academy of Economic Studies of Bucharest, Bucharest, 2002, pp. 86-95
- [Boui03]
Bouillon L., Vanderdonckt J., *User Interface Reverse Engineering*, Proceedings of 2nd Int. Conf. on Universal Access in Human-Computer Interaction UAHCI'2003 (Crete, 22-27 June 2003), Vol. 4, Stephanidis, C. (Eds.), Lawrence Erlbaum Associates, Mahwah, 2003, pp. 1509-1513.
- [Boui04]
Bouillon L., Vanderdonckt J., Chieu Chow K., *Flexible Re-engineering of Web Sites*, in proceedings of the international conference on Intelligent User Interfaces (IUI'04), (Madeira, January 13-16 2004), ACM Press, New York, USA, 2004, pp 132-139
- [Boui05]
Bouillon L., Limbourg Q., Vanderdonckt J., Michotte B., *Reverse Engineering of Web Pages based on Derivations and Transformations*, in Proceedings of LAWEB 2005 (Buenos Aires, 31 Oct.-2 Nov., 2005), IEEE Computer Society Press, Los Alamitos, 2005, pp. 3-13
- [Brit01]
Britton K.H., Case R., Citron A., Floyd R., Li Y., Seekamp C., Topol B., Tracey K., *Transcodin :Extending e-business to new environments*, IBM research journal N°40, 2001, pp153-179

C

- [Call77]
Callier F.M., *Cours de théorie des graphes*, syllabus FUNDP, année 1977-1978
- [Calv01]
Calvary G., Coutaz J., Thevenin D., "*A Unifying Reference Framework for the Development of Plastic UIs*", Proceedings of 8th IFIP Int. Conf. on Engineering for Human-Computer Interaction EHCI'2001 (Toronto, 11-13 May 2001), R. Little and L. Nigay (eds.), Lecture Notes in Comp. Science, Vol. 2254, Springer-Verlag, Berlin, 2001, pp. 173-192.

References

- [Calv02] Calvary G., Coutaz J., Thevenin D., Limbourg Q., Souchon N., Bouillon L., Vanderdonckt J., *Plasticity of User Interfaces: A Revised Reference Framework*, in Proceedings of 1st International Workshop on Task Model and Diagrams for user interface design Tamodia'2002 (Bucharest, 18-19 July 2002), Academy of Economic Studies of Bucharest, Bucharest, 2002.
- [Calv03] Calvary G., Coutaz J., Thevenin D., Limbourg Q., Souchon N., Bouillon L., Vanderdonckt J., *A Unifying Reference Framework for multi-target user interfaces*, Interacting with Computers, Volume 15, Issue 3, Elsevier Science, June 2003, pp. 289-308
- [Cana04] "Global mobile device market shows tremendous growth", Canalys, 4 Aug 2004, available at <http://www.windowsfordevices.com/articles/AT4335151181.html>
- [Canf04] Canfora G., Di Santo G., Zimeo E., *Towards Seamless Migration of Java AWT-Based Applications to Personal Wireless Devices*, in proceedings of the 11th Working Conference on Reverse Engineering (WCRE2004), Delft (the Netherlands), November 8th-12th, IEEE Computer Society Press, Los Alamitos, 2004
- [Canf05] Canfora G., Di Santo G., Zimeo E., "Developing Java-AWT Thin-Client Applications for limited devices," IEEE Internet Computing, September/October 2005, Vol. 9, No. 5, pp. 55-63
- [Capl02] Caplat G, Sourrouille, J.-L., Workshop in Software Model Engineering (WISME 2002), October 1st 2002, Dresden, Germany, 2002.
- [Chik90] Chikofsky E. J., Cross J.H., "Reverse Engineering and Design Recovery: A Taxonomy", IEEE software, volume 7, no.1, January 1990
- [Chow02] Chow K. C., *La conception et l'évaluation d'environnements de salle de contrôle : développement d'un générateur d'environnements de salle de contrôle*, Licence en sciences de gestion, Mémoire-projet, Université catholique de Louvain, Louvain-la-Neuve, Septembre 2002.
- [Chow03] Chow K.C., *GREENHouse: Case tool for user interface construction based on graph transformations*, master of computer science thesis, University of Louvain, Louvain-la-Neuve, June 2003
- [Cord02] Cordy J.R., Dean T.R., Malton A.J., Schneider K.A., "Source Transformation in Software Engineering using the TXL Transformation System", Special Issue on Source Code Analysis and Manipulation, Journal of Information and Software Technology 44,13 (October 2002), pp. 827-837.
- [Cres04] Crescenzi V., Mecca G., Merialdo P., and Missier P., "An Automatic Data Grabber for Large Web Sites", Proceedings of the 30th Int. Conf. on Very Large Data Bases VLDB'2004 (Toronto, August 31-September 3, 2004), Morgan Kaufmann, 2004, pp. 1321-1324.
- [Csab97] Csaba L., *Experience with User Interface Reengineering – Transferring DOS panels to Windows*, 1st Euromicro Working Conference on Software Maintenance and Reengineering CSMR 97, (Berlin, 17-19 March), IEEE Computer Society Press, Los Alamitos, 1997
- [Ctt99] ConcurTaskTree home page, available at <http://giove.cnuce.cnr.it/concurtasktrees.html>, 1999.

References

[Cui05] Cui X., *Transforming Phone-based Interface for Web Access: from WML to Usixml*, master thesis in business administration, Université catholique de Louvain, 2005, 143 p.

[Czar03] Czarnecki K., Helsen S., *Classification of model transformation approaches*, *Workshop on Generative Techniques in the Context of Model-Driven Architecture*, OOPSLA 2003 Conference, (Anaheim, USA, October 26-30 2003), 2003

D

[Deba95] De Baud J.-M and Rugaber S., *A Software Re-engineering Method Using Domain Models*, *Proceedings of Int. Conf. on Software Maintenance* (October 1995), pp. 204-213

[Dilu01] Di Lucca G.A., Di Penta M., Antoniol G., Casazza G., *An approach for Reverse Engineering of Web-Based Applications*, 8th Working Conference on Reverse Engineering WCRE2001, (Stuttgart, 5-7 October 2001), IEEE Press, Los Alamitos, 2001

[Dilu02] Di Lucca G.A., Fasolino A.R., Pace F., Tramontana P., De Carlini U., *WARE: a tool for the Reverse Engineering of Web Applications*, in *proceedings of European Conference on Software Maintenance and Reengineering CSMR2002*, (Budapest, 11-13 March 2002), 2002

[Dom98] *Document Object Model (DOM) Level 1 Specification (W3C Recommendation)*, W3C, 1st October 1998, available at <http://www.w3.org/TR/REC-DOM-Level-1/>

[Drah05] Draheim D., Lutteroth C., Weber G., *A Source Code Independent Reverse Engineering Tool for Dynamic Web Sites*, in *proceedings of the Ninth European Conference on Software Maintenance and Reengineering (CSMR05)*, 21-23 March 2005, Manchester, IEEE computer society press, Los Alamitos, 2005 (available at <http://www.revangie.formcharts.org>)

E

[Eise00] Eisenstein J., Vanderdonck J., Puerta A., *Adapting to Mobile Contexts with User-Interface Modeling*, *Proceedings of 3rd IEEE Workshop on Mobile Computing Systems and Applications WMCSA'2000* (Monterey, 7-8 December 2000), IEEE Press, Los Alamitos, 2000, pp. 83-92.

[Eise01] Eisenstein J., Vanderdonck J., Puerta A., *Model-Based User-Interface Development Techniques for Mobile Computing*, *Proceedings of 5th ACM Int. Conf. on Intelligent User Interfaces IUI'2001* (Santa Fe, 14-17 January 2001), Lester, J. (Ed.), ACM Press, New York, 2001, pp. 69-76

[Embl98] Embley D.W., Campbell D.M., Jiang Y.S., Ng, R.D Y.-K., Smith, Liddle S.W., and Quass D.W., *"A conceptual-modeling approach to extracting data from the web"*, *Proceedings of the 17th Int. Conf. on Conceptual Modeling ER'98* (Singapore, November 16-19, 1998), *Lecture Notes in Computer Science*, Vol. 1507, Springer-Verlag, Berlin, 1998, pp. 78-91.

References

- [Esti03] Estiévenar F., François A., Henrard J., Hainaut J.L., *A tool-supported method to extract data and schema from web site*, Proceedings of the fifth IEEE International Workshop on Web Site Evolution (WSE'03), Amsterdam, 22-23 September, Los Alamitos, 2003
- [EtFo03] "Windows gadgets to outsell Windows-PCs in 5 years", Etf Forecast, April 2003, <http://www.windowsfordevices.com/news/NS5482336283.html>
- F**
- [Flor06] Florins M., "Graceful degradation of UI", Ph.D. Thesis, IAG, UCL, 2006, to appear
- [Fold98] *Free on-line dictionary of computing definition*, 9 sep 1998, available at <http://foldoc.doc.ic.ac.uk/foldoc/foldoc.cgi?query=legacy+system>
- [Fole91] Foley J.D., Kim W.C., Kovacevic S., Murray K : *UIDE – An intelligent User Interface Design Environment*. In Sullivan J.W., Tyler S.W.(eds.): Intelligent User Interfaces. New York, ACM Press 1991, pp. 339-384
- G**
- [Gall91] Gallagher K.B., Lyle J.R., *Using Program Slicing in Software Maintenance*, IEEE Transactions on Software Engineering, August 1991 (Vol. 17, No. 8), Los Alamitos, 1991, pp. 751-761.
- [Gaer03] Gaeremynck Y., Bergman L. D., Lau T., "MORE for less: model recovery from visual interfaces for multi-device application design", Proceedings of the international conference on Intelligent user interfaces (Miami, January 2003), ACM Press, New York, USA, 2003, pp 69-76
- [Geor01] Georgia Tech's reverse engineering group, Georgia tech, 30 april 2001, available at <http://www.cc.gatech.edu/reverse/>
- [Gott04] Gottlob G., Koch Ch., Baumgartner R., Herzog M., and Flesca S., "The Lixto Data Extraction Project - Back and Forth between Theory and Practice", Proceedings of the 23rd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems PODS'2004 (Paris, June 14-16, 2004), ACM Press, New York, 2004, pp. 1-12
- [Grim94] Grimaldi R.P., *Discrete and combinatorial mathematics – An applied introduction, 3rd edition*, Addison-Wesley publishing company, ISBN 0-201-54983-2, USA, 1994
- [Gurm89] Gurminder S., Green M., *A High-Level User Interface Management System User Interface Management Systems*, Proceedings of ACM CHI'89 Conference on Human Factors in Computing Systems, 1989, p.133-138.

References

H

[Hain95]

Hainaut J.-L., Englebert V., Henrard J., Hick J.-M., Roland D., *DB-MAIN : a Database Reverse Engineering CASE tool*, in Proceedings of the 6th Workshop on Next Generation CASE tools, Jyvaskyla, Finland, July 1995, G. Grosz (Ed.), 1995

[Hilb03]

Hilbert D., Trevor J. and Schilit B., *Supporting ubiquitous information on very small devices is harder than you think*, in proceedings of HCI International 2003, June 22, Lawrence Erlbaum Associates 2003

K

[Kaas01]

Kaasinen E., Kolari J., Laakko T., "Mobile-Transparent Access to Web Services", Proceedings of 8th IFIP TC.13 Conf. on Human-Computer Interaction Interact'2001 (Tokyo, 9-13 July 2001), Elsevier Science Pub., Amsterdam, 2001.

[Kaas00]

Kaasinen E., Aaltonen M., Kolari J., Melakoski S., Laakko T., *Two Approaches to Bringing Internet Services to WAP Devices*, in proceedings of WWW9, Amsterdam, The Netherlands, North-Holland Publishing Co., 2000, pp. 231-246.

[Kong99]

Kong L., Stroulia E., Matichuk B., *Legacy Interface Migration: A Task-Centered Approach*, in H.-J. Bullinger and J. Ziegler (eds.), Proceedings of 8th Int. Conf. on Human-Computer Interaction HCI International'99 (Munich, 22-27 August 1999), Lawrence Erlbaum Associates, Mahwah/London, 1999, pp. 1167-1171, accessible at <http://www.cs.ualberta.ca/~stroulia/Papers/hci99.ps>

L

[Limb04a]

Limbourg Q., Vanderdonck J., Michotte B., Bouillon L., Lopez-Jaquero V., *UsiXML: a Language Supporting Multi-Path Development of User Interfaces*, Proceedings of 9th IFIP Working Conference on Engineering for Human-Computer Interaction jointly with 11th Int. Workshop on Design, Specification, and Verification of Interactive Systems EHCI-DSVIS'2004 (Hamburg, July 11-13), 2004.

[Limb04b]

Limbourg Q., Vanderdonck J., Michotte B., Bouillon L., Florins M., Trevisan D., *UsiXML: A User Interface Description Language for Context-Sensitive User Interfaces*, in Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli, May 25, 2004), K. Luyten, M. Abrams, Q. Limbourg, J. Vanderdonck (Eds.), 2004, pp. 55-62.

[Limb04c]

Limbourg Q., *Multi-Path Development of User Interfaces*, Ph.D. Thesis, Université catholique de Louvain, Institut d'Administration et de Gestion, Louvain-la-Neuve, Belgium, 2004.

References

- [Lech96] Lecharlier B., *Abstract Interpretation and Application to Interactive System Verification*, Proceedings of 3rd Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS96 (Namur, 5-7 June 1996), F. Bodart & J. Vanderdonck (Eds.), Springer-Verlag, Vienna, 1996, pp. 46-72.
- [Lece98] Lecerof A., Paterno F., *Automatic Support for Usability Evaluation*, in IEEE Transactions on Software Engineering, October 1998 Vol.24, No.10, Piscataway, NJ, USA, 1998, pp.863-888.
- [Limb04] Limbourg Q., Montero F., Vanderdonck J., Lopez V., *Addressing the Mapping Problem in User Interface Design*, Proceedings of 3rd International Workshop on TAsk MOdels and DIAGrams for user interface design, (Prague, November 15-16), 2004.
- [Long96] Long J., *Specifying relations between research and the design of human-computer interaction*, International Journal of Computer Studies, Vol. 44, 1996, pp. 875-920.
- [Lope01] Lopez J. F. and Szekely P. , *Web page adaptation for universal access*. In: C. Stephanidis (ed.) Universal Access in HCI: Towards and Information Society for All.(Proceedings of 1st International Conference on Universal Access in Human-Computer Interaction, New Orleans, August 8-10) Mahwah, NJ: Lawrence Erlbaum Associates, 2001 pp. 690-694.
- [Lope04] Lopez J., "*Presentation adaptation by implicit model extraction, automatic web page adaptation*", Phd. Thesis, University of southern California, 195 pages, May 2004.
- [Lope05] Lopes D., Hammoudi S., Bézivin J., Jouault F., *Generating Transformation Definition from Mapping Specification : Application to Web Services Platform*, in proceedings of CAISE05, (Porto, 13-17 june), 2005.
- [Luyt04] Luyten K., Abrams M., Limbourg Q., Vanderdonck J., Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" UIXML'04 (Gallipoli, 25 May 2004), 2004.
- [Luo93] Luo P., Szekely P., Neches R., *Management of interface desing in HUMANOID*, in Proceedings of InterCHI93, ACM Press, New York, April 93, pp 107-114.

M

- [Malt93] Malton A. J., "*The Denotational Semantics of a Functional Tree-Manipulation Language*", Computer Languages 19,3 (July 1993), pp. 157-168.
- [Mari05] J. Marion (LINF2), *Reverse engineering of Graphical User Interfaces based on Resource Files*, bachelor thesis in computer sciences, INGI, UCL, 2005.
- [Mcbr99] McBrien P.J. and Poulouvasilis A., *A Uniform Approach to Inter-Model Transformations*, In Advanced Information Systems Engineering, 11th International Conference CAiSE'99, Springer Verlag LNCS 1626, pp. 333-348.

References

- [Mcbr02] McBrien P.J. and Poulouvassilis A., *Schema Evolution in Heterogeneous Database Architectures, A Schema Transformation Approach*, In Proceedings of CAiSE02, Springer Verlag LNCS, Volume 2348, pp. 484-499, 2002, ISSN: 0302-9743, ISBN 3-540-43738-X.
- [Merl95] Merlo E., Elwahidi A., *Generating user interfaces from specifications produced by a Reverse Engineering process*, Proceedings of the International Workshop on Computer-Aided Software Engineering CASE-95, (Toronto, July), 1995.
- [Mell04] Mellor S.J., Scott K., Uhl A., Weise D., *MDA Distilled : Principles of Model-Driven Architecture*. Addison Wesley, New York, 2004
- [Merl95b] Merlo E., Gagne P.-Y., Girard J.-F., Kontogiannis K., Hendren L., Panangaden P., DeMori R., *Reengineering User Interfaces*, IEEE Software, Vol. 12 No. 1, 1995, pp. 64 – 73.
- [Merl93] Merlo E., Girard J.F., Kontogiannis K., Panangaden P., De Mori R., *Reverse Engineering of User Interfaces*, in R.C. Waters, E.J. Chikofsky (eds.), Proceedings of 1st Working Conference on Reverse Engineer-ing WCRE'93 (Baltimore, 21-23 May 1993), IEEE Computer Society Press, Los Alamitos, 1993, pp. 171-179.
- [Mont04] Montero F., Víctor López Jaquero V., Vanderdonckt, J., Gonzalez P., Lozano M.D., Limbourg Q., *Solving the Mapping Problem in User Interface Design by Seamless Integration in IdealXML*, Proceedings of 12th Int. Workshop on Design, Specification, and Verification of Interactive Systems DSV-IS'2005 (July 13–15, 2005, Newcastle upon Tyne), M. Harrison (ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 2005
- [Moor97] Moore M.M., Rugaber S., *Using Knowledge Representation to Understand Interactive Systems*, Proceedings of the Fifth International Workshop on Program Comprehension IWPC'97 (Dearborn, 28-30 May 1997), IEEE Computer Society Press, Los Alamitos, 1997.
- [Moor93] Moore M.M., Rugaber S., *Issues in User Interface Migration*, in Proceedings of the Third Software Engineering Research Forum, Orlando, 10 novembre 1993.
- [Moor94] Moore M.M., Rugaber S., Seaver P., *Knowledge Based User Interface Migration*, in Proceedings of the 1994 International Conference on Software Maintenance (Victoria, British Columbia, September 1994).
- [Moor96a] Moore M.M., *Representation Issues for Reengineering Interactive Systems*, ACM Computing Surveys, Vol. 28, No. 4, article # 199, December 1996, accessible at <http://www.acm.org/pubs/articles/journals/surveys/1996-28-4es/a199-moore/a199-moore.html>
- [Moor96b] Moore M.M., *Rule-Based Detection for Reverse Engineering User Interfaces*, in Proceedings of 3rd Working Conference on Reverse Engineering (Monterey, 8-10 November 1996), L. Wills, I. Baxter, E. Chikofsky (eds.), IEEE Computer Society Press, Los Alamitos, 1996, pp. 42-48.
- [Mori03] Mori G., Paternò F., Santoro C., *Tool support for designing nomadic applications*, Proceedings of the 2003 international conference on Intelligent user interfaces, (Miami, January 2003), ACM Press, New York, USA, 2003, pp. 141-148.

References

- [Mori05] G.Mori, F.Paternò, *Automatic semantic platform-dependent redesign*, Proceedings Smart Objects and Ambient Intelligence 2005, Grenoble, October 2005, pp.177-182
- [Msxml] Microsoft Download Center, Microsoft, Available at <http://www.microsoft.com/downloads/results.aspx?productID=&freetext=msxml&DisplayLang=en>
- [Myer92] Myer B.A., Rosson M.B., *Survey on User Interface Programming Tools and Techniques*, Proceedings of ACM CHI 92 Conference on Human Factors in Computing Systems, 1992, pp 195-202

N

- [Nich02] Nichols J., Myers B.A., Higgins M., Hughes J., Harris T.K., Rosenfeld R., Pignol M., “*Generating Remote Control Interfaces for Complex Appliances*”, Proceedings of the 15th annual ACM symposium on User interface software and technology, (Paris, 27-30 October 2002), ACM press, New York, USA, 2002

[Nich04]

Nichols, J., Myers, B. A., & Litwack, K. (2004). Improving Automatic Interface Generation with Smart Templates. Proceedings of the 8th International Conference on Intelligent User Interfaces IUP2004 (13-16 January, Funchal, Portugal).

O

- [OCL03] UML 2.0 OCL specification, available at <http://www.omg.org/docs/ptc/03-10-14.pdf>, 2003.

[Olle88]

Olle T.W., Hagelstein K., Macdonald I.G., Rolland C., Sol H.G., Van Assche F., Verjin-Stuart A., *Information Systems Methodologies*, Addison-Wesley, 1988.

[Olse00]

Olsen D.R., Jefferies S., Nielsen T., Moyes W., Fredrickson P., *Cross Modal Interaction using XWEB*, Proceedings of the 13th annual ACM symposium on User interface software and technology UIST 2000 (San Diego, USA, 2000), ACM Press, New York, USA, 2000, pp 191-200

[OMG02]

Object Management Group. *Request for Proposal: MOF 2.0 Query / Views / Transformations RFP*, 2002.

P

[Paga02]

Paganelli L., Paterno F., *Automatic Reconstruction of the Underlying Interaction Design of Web Applications*, In Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering (SEKE02), (Ishia, Italy, July 15-19), 2002.

[Pate00]

Paterno F., *Model-Based Design and Evaluation of Interactive Applications*, Springer Verlag, London, 2000.

References

- [Pate02] Paterno F. and Santoro C., *One Model, Many Interfaces*, in Proceedings of CADUI 2002, the 4th International Conference on Computer-Aided Design of User Interfaces (Valenciennes, France, May 2002), Kluwer Academics Pub., Dordrecht, 2002, pp 143-154
- [Pere03] Pérez-Quiñones M.A., Capra R.G., Shao Z., « *The Ears have it: A Task by Information Structure Taxonomy for Voice Access to Web Pages* » in Proceedings of Interact 2003, (Zürich, September 2003), 2003.
- [Pixe] *diagram & history of programming languages*, available at <http://merd.sourceforge.net/pixel/language-study/diagram.html>
- [Puer96] Puerta A.R., “*The Mecano Project: Comprehensive and Integrated Support for Model-Based Interface Development*”, Proceedings of 2nd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96 (Namur, 5-7 June 1996), Presses Universitaires de Namur, 1996, pp. 19-35.
- [Puer97] Puerta A.R., Maulsby D., *MOBI-D: A Model-Based Development Environment for User Centered Design*, in proceedings of CHI'97, ACM Press, Atlanta, March 1997, pp4-5.
- [Puer99] Puerta A.R. and Eisenstein J., “*Towards a General Computational Framework for Model-Based Interface Development Systems*”, Proceedings of ACM Conference on Intelligent User Interfaces IUI'99, ACM Press, New York, 1999, pp. 171-178
- ## Q
- [Qvt03] *QVT Partners revised submission to QVT*, 18/8/2003, QVT, accessible at <http://qvtp.org/downloads/>
- ## R
- [Rays99] Rayson P., Garside R., Sawyer P. (1999). *Recovering Legacy Requirements*. In Proceedings of REFSQ'99. Fifth International Workshop on Requirements Engineering: Foundations of Software Quality, June 14-15 1999, Heidelberg, Germany. Published by University of Namur, pp. 49-54.
- [Resh02] Resource Hacker 2002, available at <http://www.angusj.com/resourcehacker/>
- [Rest06] Restorator resource editor 2006, available at <http://www.bome.com/Restorator/>
- [Reve00] REVERE project, information available at <http://www.comp.lancs.ac.uk/computing/research/cseg/projects/revere/>, 2000.
- [Ricc01] Ricca F., Tonella P., Baxter I.D., “*Restructuring Web applications via Transformation Rules*”, Proceedings of IEEE Workshop on Source Code Analysis and Manipulation SCAM'2001

References

- (Florence, 5-9 November 2001), IEEE Computer Soc. Press, Los Alamitos, 2001, pp. 150-160.
- [Ricc00] Ricca F., Tonella P., *Web site analysis: Structure and evolution*. In Proceedings of the International Conference on Software Maintenance, (San Jose, USA, 2000) , 2000, pp. 76-86
- [Rode05] Rode J., Bhardwaj Y., Pérez-Quiñones M.A., Rosson M.BHowartg., J., *As Easy as "Click": End-User Web Engineering*. International Conference on Web Engineering 2005. Sydney, Australia, July 27-29
- [Rose99] Rosen K.H., *Discrete mathematics and its applications*, 4th edition, McGraw-Hill International Editions, Mathematics and statistics series, ISBN 0-07-289905-0, Singapore, 1999
- [Roui04] Rouillard J., *VoiceXML, le langage d'accès à Internet par telephone*. Vuibert, Paris, 2004
- S**
- [Sali92] Salisin J., *The Design Record: Keystone of Software engineering*, Keynote Speech of the 3rd Reverse Engineering forum, 1992
- [Scot01] Scott N.G., Gingras I., *The Total Access System*, In Proceedings of ACM Conference on Human Aspects in Computing Systems CHI'2001 (Seattle, 31 March-5 April 2001), Extended Abstracts, ACM Press, New York, 2001, pp. 13-14
- [Shao03] Shao Z., Capra R.G., Pérez-Quiñones M.A., "*Transcoding HTML to VoiceXML Using Annotations*", Proceedings of ICTAI 2003, 2003.
- [Sem03] *Software Engineering Management*, Anthony Aaby, 6/11/2003, available at <http://cs.wvc.edu/~aabyan/435/Maintenance.html>
- [Situ00] Situ Q. and Stroulia E., *Task-structure Based Mediation: The Travel-Planning Assistant Example*. In Proceedings of the Thirteenth Canadian Conference on Artificial Intelligence (AI'2000). (Montreal 14-17 May 2000), 2000.
- [Stro00a] Stroulia E., El-Ramly M., Sorenson P., Penner R., *Legacy Systems Migration in CeLEST*. Short Research demonstration, In the Proceedings of the 22nd International Conference on Software Engineering, (Limerick, Ireland, 4-11 June 2000), 2000; Accessible at <http://www.cs.ualberta.ca/~stroulia/Papers/rd-final.pdf>
- [Stro00b] Stroulia E., Thomson J., Situ Q., *Constructing XML-speaking wrappers for WEB Applications: Towards an Interoperating WEB*, In the Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'2000). (23-25 Brisbane, November, 2000), IEEE Computer Society, Los Alamitos, 2000.
- [Stro02] Stroulia E., Kapoor R.V., *Reverse engineering interaction plans for legacy interface migration*, Chapter 26, Proceedings of 4th Int. Conf. on Computer-Aided Design of User Interfaces CADUI'2002 (Valenciennes, 15-17 May 2002), Kluwer Academics Pub., Dordrecht, 2002, pp. 295-310.

References

- [Suss01] Sussman J., Banavar G., Bergman L., *Generalization: A Key Concept in the Creation of Platform Independent User Interfaces*. In UbiTools 2001, Workshop on Application Models and Programming Tools for Ubiquitous Computing, Atlanta, September 2001, 2001.
- [Szek92] Szekely P., Luo P., Neches R., *Facilitating the Exploration of Interface Design Alternatives: The Humanoid Model of Interface Design*, in Bauersfeld P., Bennet K., Lynch G.(eds.): Proceedings of CHI 92. New York, ACM Press, 1992, pp. 507-514.
- [Szek96] Szekely P., Sukaviriya P., Castells P., Muthukumarasamy J., Salcher E., *Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach*, in Engineering for Human Computer Interaction, Chapman & Hall, London, UK, 1996, pp.120-150.
- ### T
- [Thev01] Thevenin, D.: *Adaptation en Interaction Homme-Machine : le cas de la Plasticité*. PhD thesis, Grenoble, 2001, 234 pages. Accessible at <http://iihm.imag.fr/publs/2001/>
- [Tidy03] *Clean Up your Web Pages With HTML Tidy*, D. Ragett, Accessible at <http://www.w3.org/People/Raggett/tidy>
- ### U
- [Uiml04] UIML website, available at <http://www.uiml.org/index.php>, 2004.
- ### V
- [Vand93] Vanderdonckt J., Bodart F., "Encapsulating Knowledge for Intelligent Interaction Objects Selection", Proceedings of ACM Conf. on Human Aspects in Computing Systems In-terCHI'93 (Amsterdam, 24-29 April 1993), ACM Press, New York, 1993, pp. 424-429.
- [Vand94] Vanderdonckt J., *Automatic Generation of a User Interface for Highly Interactive Business-Oriented Applications*, in Plaisant C.(ed.): Companion Proceedings of CHI'94, New York, ACM Press, 1994, pp. 41 & 123-124.
- [Vand95] Vanderdonckt J., *Knowledge-Based Systems for Automated User Interface Generation: The Trident Experience*, Technical report RP-95-010. FUNDP, Institut d'informatique, 1995. Available at [http://www. Info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-010](http://www.Info.fundp.ac.be/cgi-bin/pub-spec-paper?RP-95-010)
- [Vand05] Vanderdonckt J., *A MDA-Compliant Environment for Developing User Interfaces of Information Systems*, in Proceedings of Caise 05 (Conference on Advanced Information System Engineering), 13-17 June, Porto, Portugal, Springer Verlag,2005, pp 16-31.

References

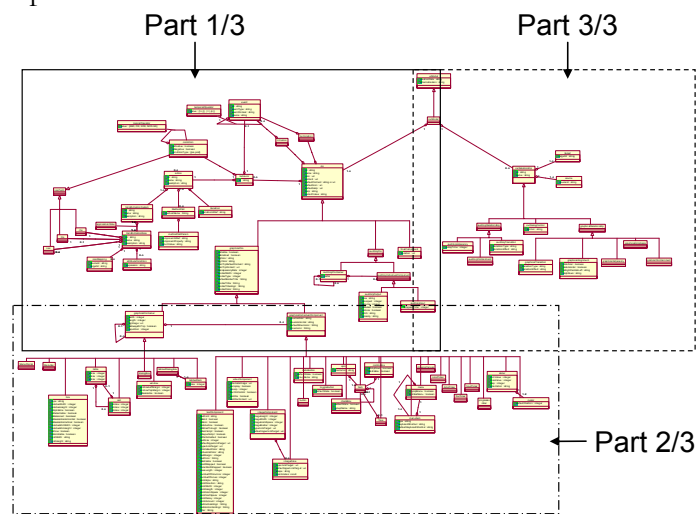
- [Vans93b] Van Sickle L., Liu Z.Y., Ballantyne M., *Recovering User Interface Specifications for Porting Transaction Processing Applications*, EDS Research, Austin Laboratory, 1601 Rio Grande, Suite 500, Austin TX 78701, 1993.
- [Vand01] Vanderdonckt J., Bouillon L., Souchon N., *Flexible Reverse Engineering of Web Pages With Vaquista*, in Proceedings of the 8th Working Conference on Reverse Engineering, (Stuttgart, September 2001), IEEE Computer Society, Los Alamitos, 2001.
- [Vinc05] Vincent D., *Un pas vers le poste de travail unique : QTKiXML, un interpréteur d'interface utilisateur à partir de sa description*, Master thesis, IAG, UCL, 2005.
- [Voic01] <http://studio.tellme.com/vxml2/ref/elements/>
- [Voic02] <http://docs.voxeo.com/voicexml/2.0/>
- [Voic03] <http://www.yoyodesign.org/doc/w3c/voicexml20/>
- [Voic04] <http://www.voicexml.org/tutorials/intro4.html>
- [Voic05] <http://www.wirelessdevnet.com/channels/voice/training/voicexmloverview.html>
- W**
- [Will90] Wills L. M., *Automated Program Recognition: A Feasibility Demonstration*, Artificial Intelligence, Volume 45 , Issue 1-2 (September 1990), Elsevier Science Publishers Ltd. Essex, UK, 1990, pp.113-171.
- [Wils93] Wilson S., Johnson P., Kelly C., Cunningham J., Markopoulos P., *Beyond Hacking : A model-Based Approach to User Interface Design*. Proceedings of HCI 93, Cambridge University Press, 1993, pp. 40-48.
- X**
- [Xhtm] Xhtml page from w3c, W3C HTML working group, available at <http://www.w3.org/TR/xhtml1/>
- [Xian06] Xiang P., Shi Y., *Recovering Semantic Relations from Web Pages Based on Visual Clues*, in proceedings of the international conference on Intelligent User Interfaces (IUI'06), (Sydney, 29 January-1st February), ACM Press, New York, USA, 2006, pp. 342-344.
- [Ximl02] *The XIML Consortium*, Redwhale, 2002, <http://www.ximl.org>
- [Xslt99] XSLT W3C recommendations, available at <http://www.w3.org/TR/xslt>, 1999.

Appendix A

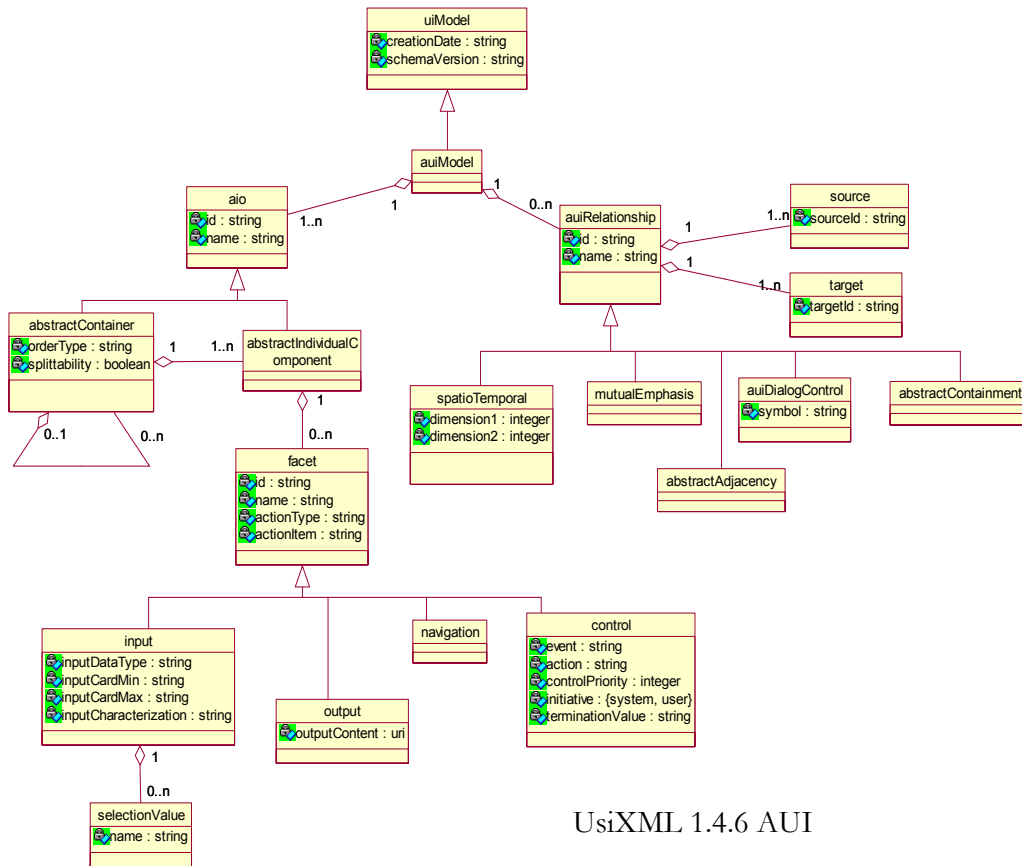
UsiXML 1.4.6 Class Diagram

This first appendix contains the meta-model for the UsiXML 1.4.6 concrete UI model and abstract UI model. The CUI meta-model is divided into 3 parts. The first part contains graph transformations and components for the specification of vocal UI. The second part is dedicated to graphical UI components. The last part contains elements used to describe relations in the model.

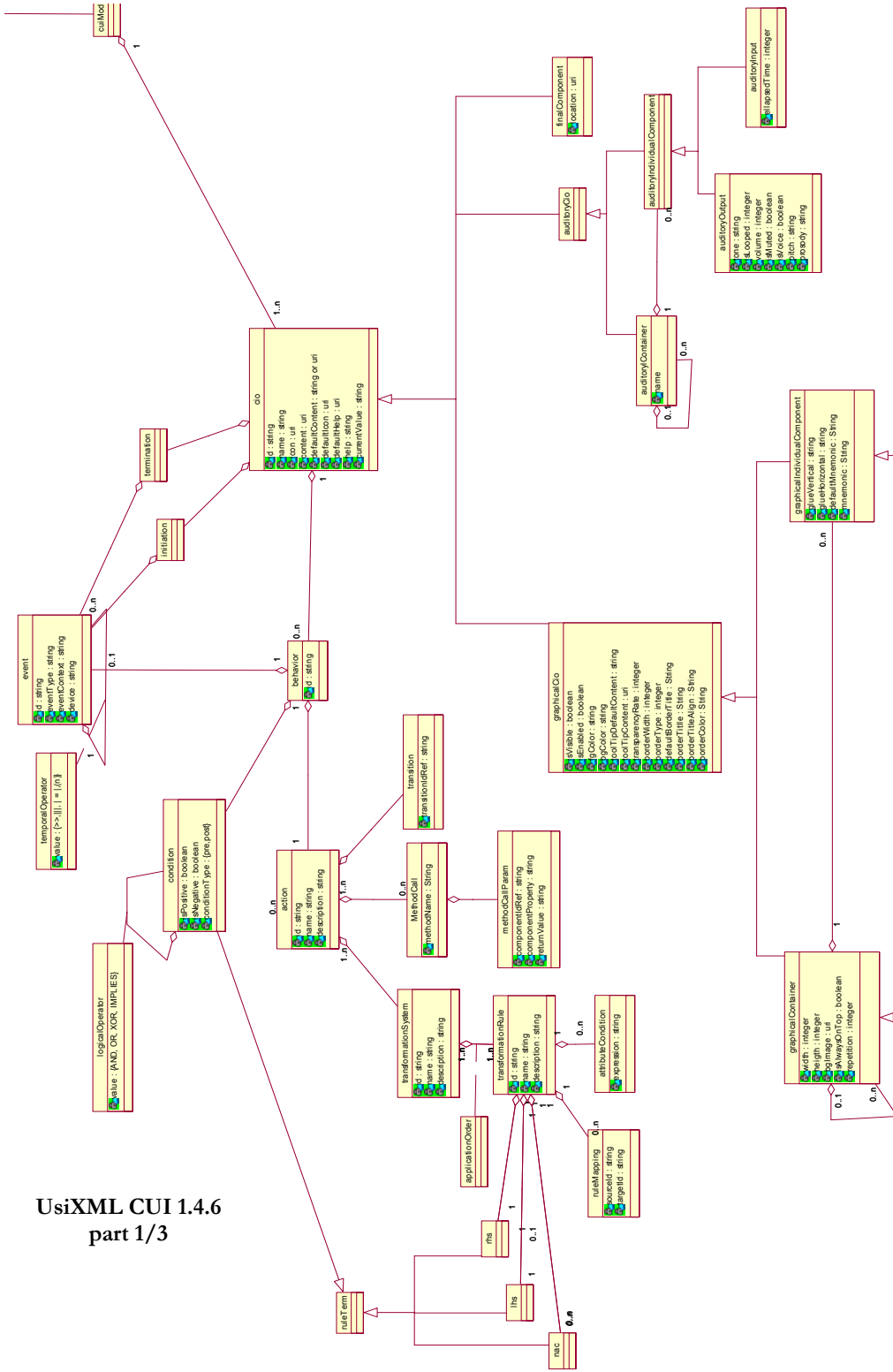
Meta-models from appendix A and B are described thanks to inheritance relations (normal arrow) and aggregation relations (arrows ended with a rombus). Aggregation relations are used to describe that an element may contain another type of element, e.g. element `<a>` may contain an element `` (but not the contrary) in HTML 4.0. Generalization relations are used to facilitate the reading of meta-models by grouping elements with similar attributes and similar aggregation relations. Elements starting with a plus symbol represent these generalizations (in appendix B). None of the classes of these class diagrams possesses methods, as the elements composing these languages are static declarative components of the UI that do not allow the invocation of methods.



Appendix A

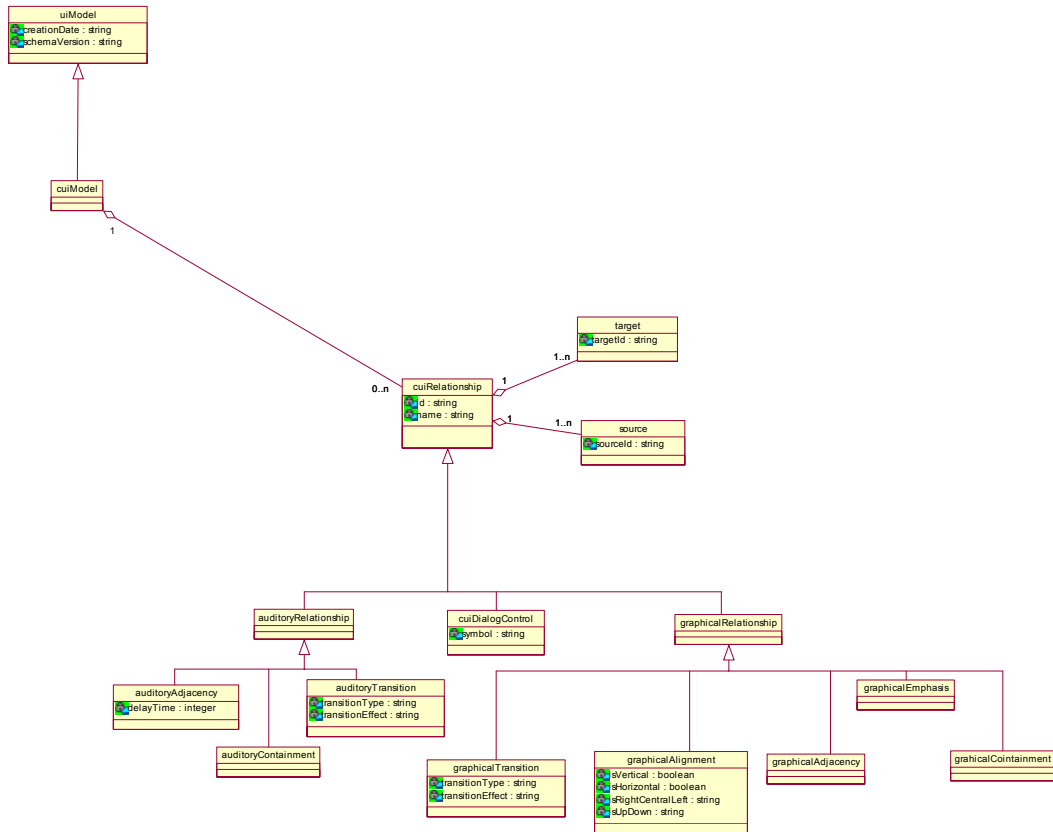


UsiXML 1.4.6 AUI



UsiXML CUI 1.4.6
part 1/3

Appendix A



UsiXML CUI 1.4.6 part 3/3

Appendix B

UML diagrams of languages

This appendix contains all the UML diagrams of the source languages. These diagrams have been constructed thanks to languages DTD for HTML, VoiceXML and WML and thanks to documentation for Windows resource files. Some comments for each of these diagrams are given below.

XIML: The XIML language is mainly composed of four types of components: models, elements, attributes and relations between the elements. We can distinguish two types of models, the **interface** model and the model **components**. The first is the root of any XIML document and contains the various sub-models (model components) available in XIML. The model components (task, domain, user, presentation, dialogue, platform, preferences and the general model) contain information specific to a dimension of the interface. Each model is composed of elements. Each model or element can possess features (composed of attributes or relations) or definitions (attribute or relation definitions).

HTML: There are five categories of elements for the HTML meta-model: elements specific to the head section (**meta**, **script**...), containers, (such as **forms**, **tables**...) that define hierarchy of elements, formatting tags (such as **b**, **i**, **p**...), lists (**dl**, **ul**...) and atomic tags that cannot contain other tags (**img**, **object**, **button**...).

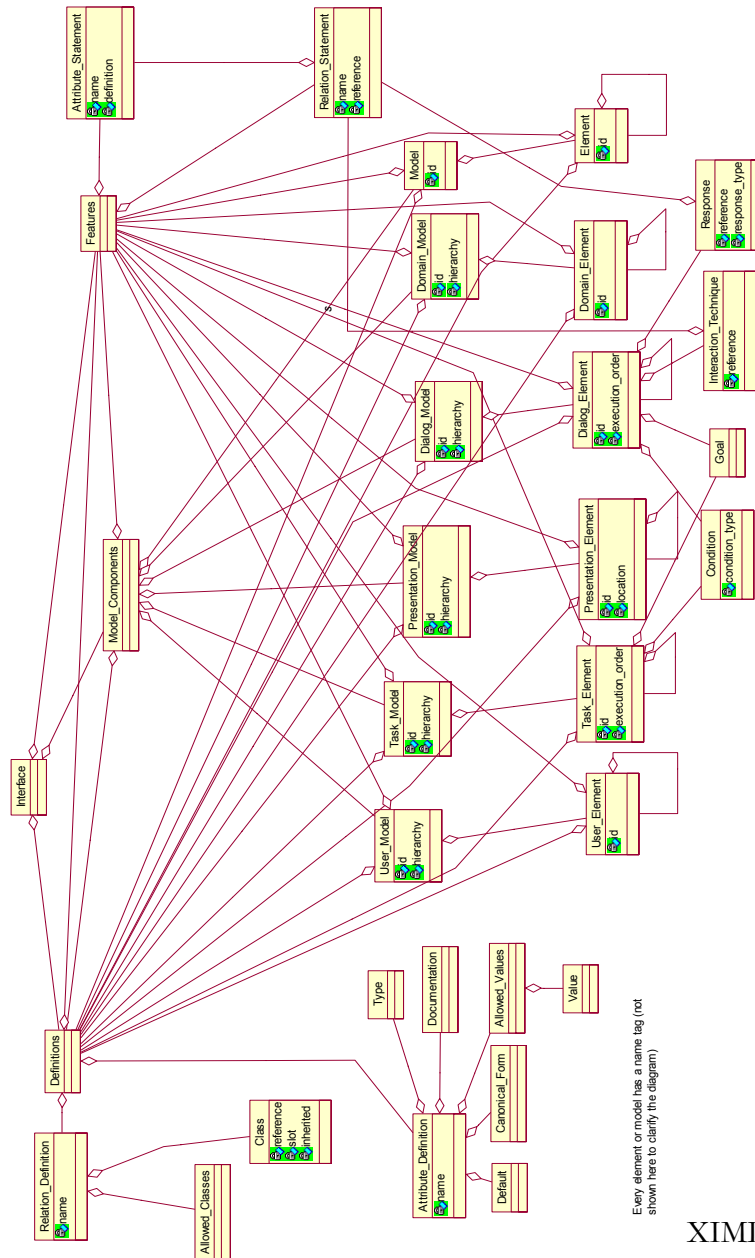
WML: The root element of the model is a **wml** element, which can contain a **meta**, **template** or **card** node. The UI is contained in **card** elements and can be composed of navigation elements (**+navelements**), **timer**, paragraphs (**p**) or fields (**+fields**). Fields are decomposed in several input elements (**select**, **input**...) and flow elements (**+flow**) that represent formatting tags for the text of the UI, such as **b** or **i**(for bold or italic text), **tables**, links(**a**) and images (**img**).

Appendix B

VoiceXML: The meta-model of the VoiceXML 2.0 language is given twice in this section, one in a lightened version, as some simplifications have been made and all the possible containment relations are not shown in this version of the model in order to make it readable, and another representing the full version. The root of this meta-model is a `vxml` element, which can contain meta information (**meta** and **metadata**), **link**, **property**, events handlers (**+event handler**), containers and input (**+container**) or executable content (**+executable content**). The VoiceXML UI is embedded in node belonging to the containers class. This superclass (preceded by a `+` symbol) groups logical containers (**block**, **initial**) and form-input elements (**field**, **record**, ...) as they share common attributes and properties. The **event handler** superclass contains several predefined event (**help**, **noinput**...) so as user-defined catchers (**catch**). Finally, **executable content** is the class for the rest of UI components. It contains tags allowing modifying control flow (**if**, **then**, **else**, **return**...) so as output nodes (**prompt**, **audio**). Executable contents and event handlers are characterized by the fact that these elements can not embed another element of the same class, contrary to containers.

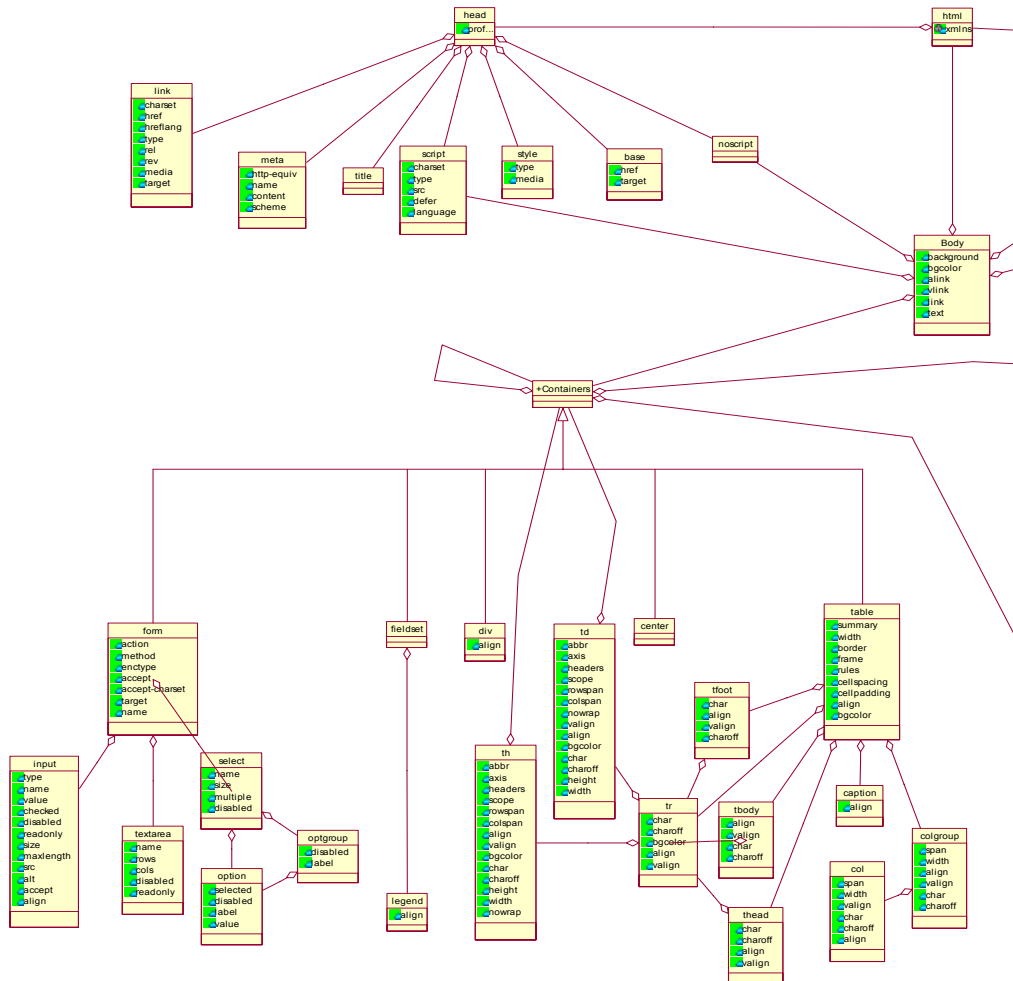
Windows resource files: This meta-model is in fact a model based on a logical representation of resource files, as names and attributes of resource files can be relatively cryptic (e.g. `WS_EX_DLGMODAL_FRAME` has been replaced by `dialogModalFrame` in the window entity). The first model about menus contains only four types of elements: a **menubar** containing the menu specification, **popupmenus**, **menu items** and **separators**. This small meta model can be found in chapter 8. The subject of the second model is the window/dialog box meta-model representation. This model contains almost all the “classic” interaction that we can find in windows application, such as button, combo boxes, check boxes etc...

Appendix B



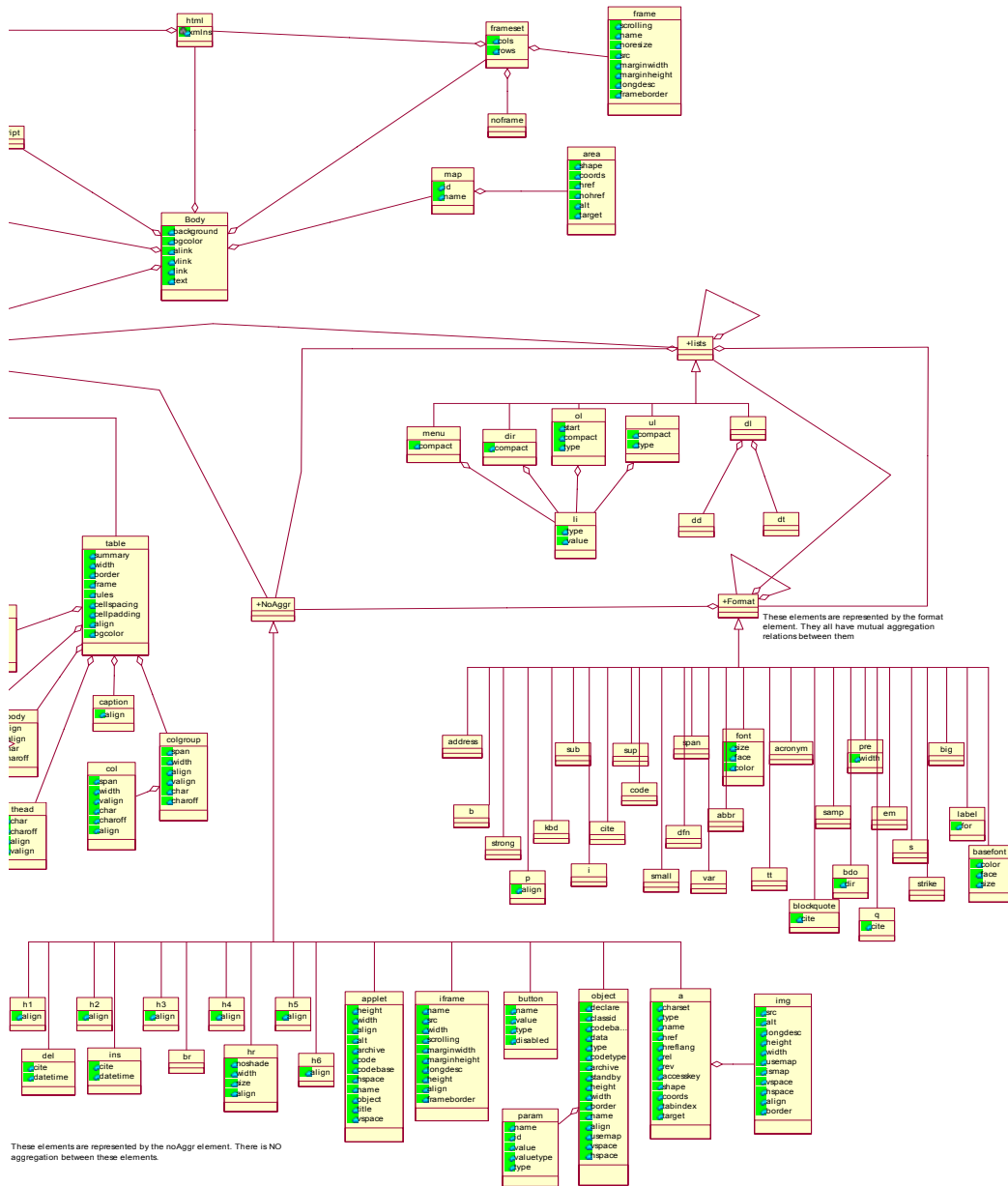
XIML

Appendix B

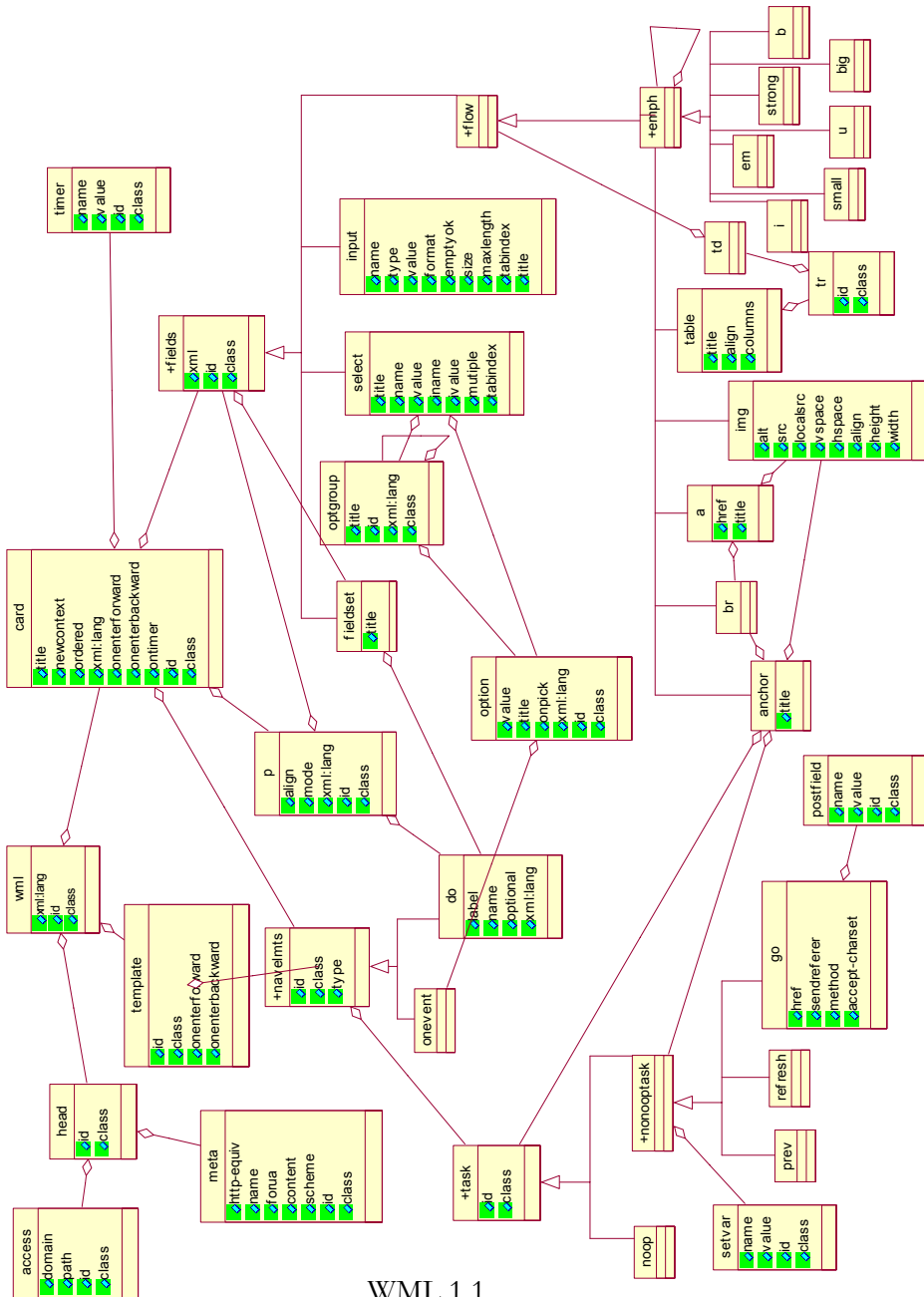


HTML 4.0 part 1/2

Appendix B



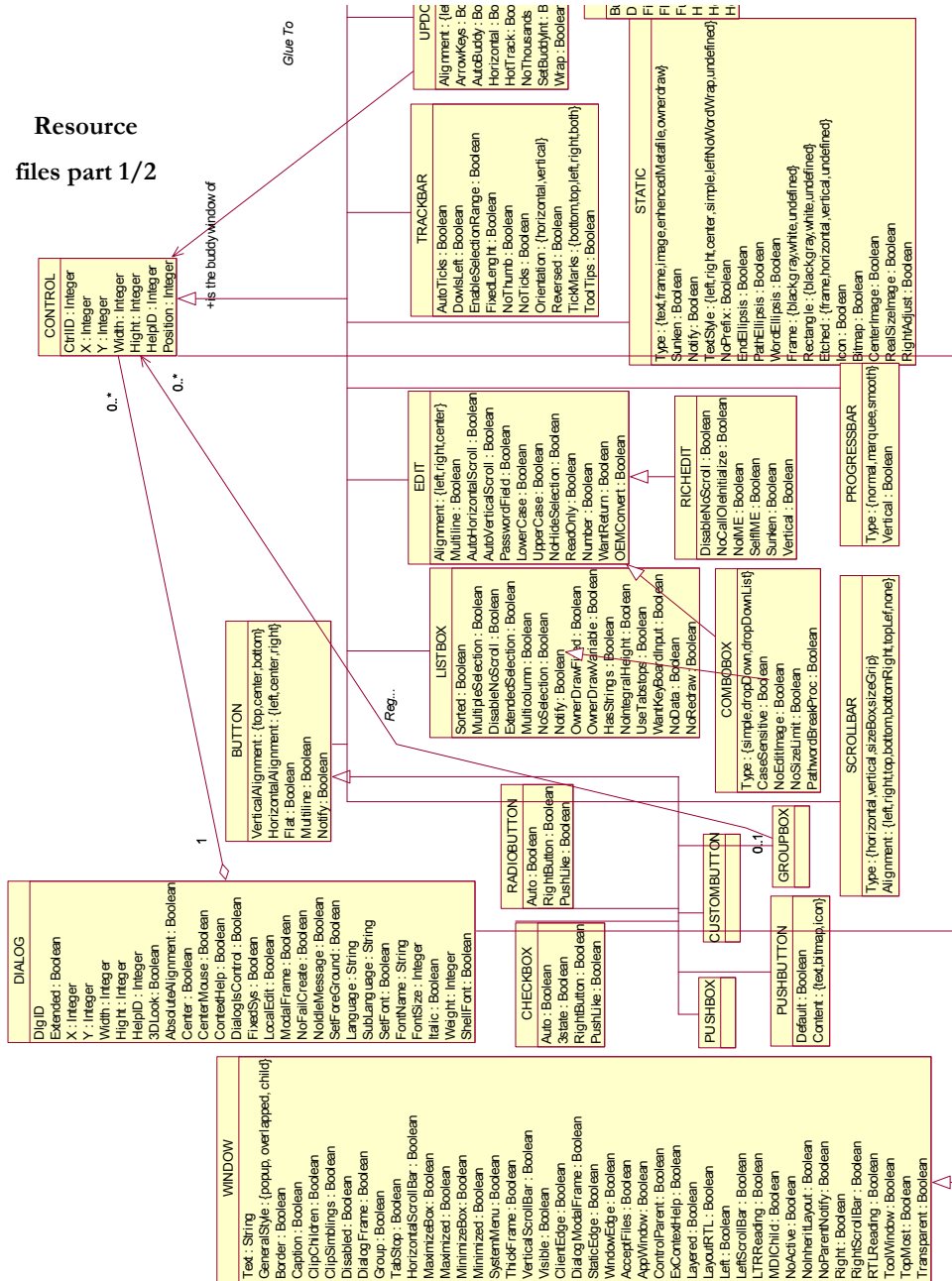
HTML 4.0 part 2/2



WML 1.1

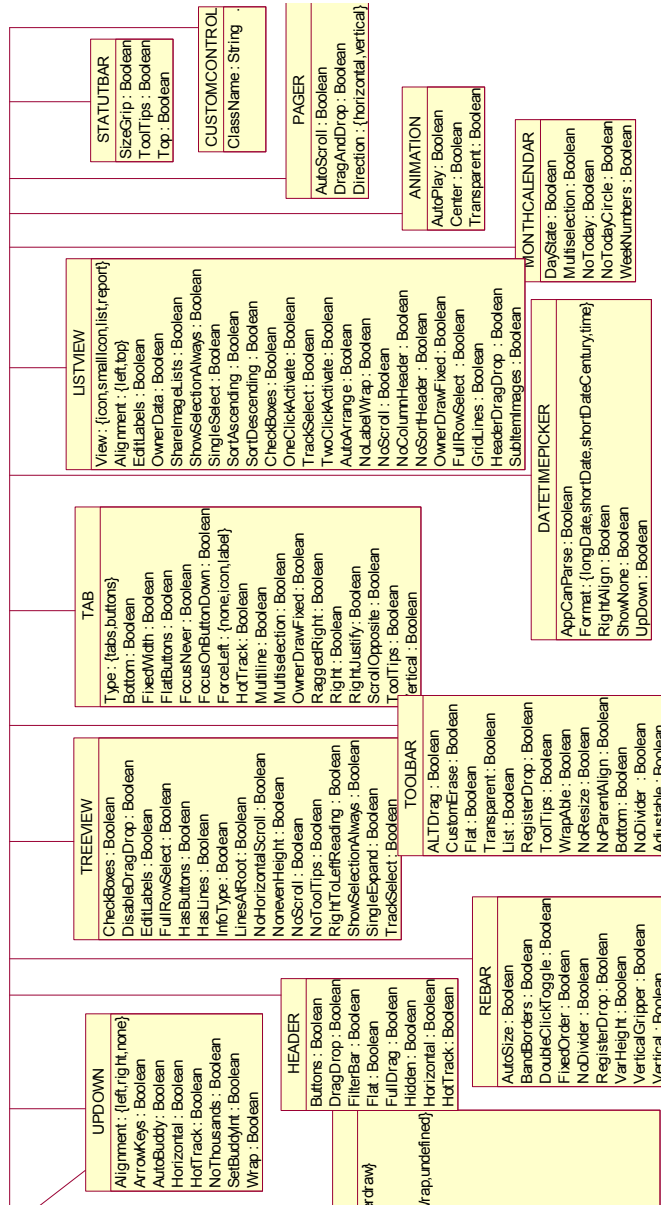
Appendix B

Resource files part 1/2



Appendix B

Give To



Resource files part 2/2

Appendix C

Reverse engineering derivation rules

This appendix is organized into three parts: the first contains inter-tree derivation rules, the second represents multi-tree rules and the third is for intra-tree rules. Some of the mappings are commented after the group of rules, and the reader can also find explanation about some selected derivation rules in chapter 6, 7 or 8.

Inter-Tree derivation rules

This part contains 48 groups of rules. For each group, the first mapping in bold corresponds to the detection of a node in the source tree that causes the creation of a node in the target tree.

G1 – body (window)

$\forall x \in T_H : x = \mathbf{body} \rightarrow \mathbf{Addnode}(\mathbf{"window"}, \mathbf{idwin}) \text{ where } \mathbf{idwin} = \mathbf{NodeAmount}(Tt)$
$\forall x \in T_H : x = \mathbf{body} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"id"}, \mathbf{idwin})$
$\forall x \in T_H : x = \mathbf{body} \wedge \mathbf{title.textnode} \neq \mathbf{NULL} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"name"}, \mathbf{title.textnode})$
$\forall x \in T_H : x = \mathbf{body} \wedge \mathbf{title.textnode} = \mathbf{NULL} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"name"}, \mathbf{idwin})$
$\forall x \in T_H : x = \mathbf{body} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"isEnabled"}, \mathbf{"true"})$
$\forall x \in T_H : x = \mathbf{body} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"isVisible"}, \mathbf{"true"})$
$\forall x \in T_H : x = \mathbf{body} \wedge \mathbf{x.bgcolor} \neq \mathbf{NULL} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"bgColor"}, \mathbf{x.bgcolor})$
$\forall x \in T_H : x = \mathbf{body} \wedge \mathbf{x.topmargin} \neq \mathbf{NULL}$ $\quad \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"windowTopMargin"}, \mathbf{x.topmargin})$
$\forall x \in T_H : x = \mathbf{body} \wedge \mathbf{x.leftmargin} \neq \mathbf{NULL}$ $\quad \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"windowLeftMargin"}, \mathbf{x.leftmargin})$
$\forall x \in T_H : x = \mathbf{body} \wedge \mathbf{x.background} \neq \mathbf{NULL} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"bgImage"}, \mathbf{x.background})$
$\forall x \in T_H : x = \mathbf{body} \rightarrow \mathbf{AddAttribute}(\mathbf{idwin}, \mathbf{"filename"}, \mathbf{filename})$
$\forall x \in T_H : x = \mathbf{body}$

Appendix C

$\rightarrow \text{ConstrBox}(\text{"box"}, \text{"vertical"}, \text{idbox})$ where $\text{idbox} = \sum \text{node} \in T_t \wedge \text{AddArc}(\text{idwin}, \text{idbox})$ $\forall x \in T_H : x = \text{body}$ $\rightarrow \text{ConstrBox}(\text{"box"}, \text{"horizontal"}, \text{idhbox})$ where $\text{idhbox} = \sum \text{node} \in T_t \wedge \text{AddArc}(\text{idbox}, \text{idhbox})$
--

G2 - w - card (window)

<p>► $\forall x \in T_W : x = \text{card} \wedge x.\text{id} = \text{NULL} \rightarrow \text{Addnode}(\text{"window"}, \text{idwin})$ where $\text{idwin} = \text{NodeAmount}(T_t)$</p> <p>► $\forall x, y \in T_W : x = \text{card} \wedge y \neq \text{template} \wedge (x.\text{ordered} = \text{true} \vee x.\text{ordered} = \text{null}) \wedge \text{LeftSibling}(x) = \text{"card"} \rightarrow \text{Addnode}(\text{"TextComponent"}, \text{idtext}, \text{LeftSibling}(x)) \wedge \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{"Next"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, \text{idwin}) \wedge \text{AddArc}(\text{itao}(\text{LeftSibling}(x)).\text{id}, \text{idtext}) \wedge \text{AddGraphTr}(\text{idtext}, \text{idwin})$ where $\text{idtext} = \text{NodeAmount}(T_t)$ Inserts a textComponent (a link that targets this card) to the previous card</p> <p>► $\forall x, y \in T_W : x = \text{card} \wedge y \neq \text{template} \wedge (x.\text{ordered} = \text{true} \vee x.\text{ordered} = \text{null}) \wedge \text{LeftSibling}(x) = \text{"card"} \rightarrow \text{Addnode}(\text{"TextComponent"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{"Back"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, \text{itao}(\text{LeftSibling}(x)).\text{id}) \wedge \text{AddArc}(\text{idwin}, \text{idtext}) \wedge \text{AddGraphTr}(\text{idtext}, \text{itao}(\text{LeftSibling}(x)).\text{id})$ where $\text{idtext} = \text{NodeAmount}(T_t)$ Inserts a textComponent (a link that targets the previous card) to this card</p> <p>► $\forall x, w, y, z \in T_W : x = \text{card} \wedge z = \text{template} \wedge w \in \text{childNodes}(z) \wedge w = \text{do} \wedge w.\text{type} = \text{accept} \wedge y = \text{go} \wedge y \in \text{childNodes}(w) \wedge y.\text{href} \neq \text{NULL} \wedge w.\text{label} \neq \text{NULL} \wedge !(\exists v = \text{card} : v.\text{id} = y.\text{href}) \rightarrow \text{AddNode}(\text{"TextComponent"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{"x.label"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, y.\text{href}) \wedge \text{AddGraphTr}(\text{idtext}, y.\text{href})$ where $\text{idtext} = \text{NodeAmount}(T_t)$</p> <p>► $\forall x, w, y, z \in T_W : x = \text{card} \wedge z = \text{template} \wedge w \in \text{childNodes}(z) \wedge w = \text{do} \wedge w.\text{type} = \text{accept} \wedge y = \text{go} \wedge y \in \text{childNodes}(w) \wedge y.\text{href} \neq \text{NULL} \wedge w.\text{label} \neq \text{NULL} \wedge (\exists v = \text{card} : v.\text{id} = y.\text{href}) \rightarrow \text{AddNode}(\text{"TextComponent"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{"x.label"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, v.\text{id}) \wedge \text{AddGraphTr}(\text{idtext}, v.\text{id})$ where $\text{idtext} = \text{NodeAmount}(T_t)$</p> <p>$\forall x \in T_W : x = \text{card} \wedge x.\text{id} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"id"}, x.\text{id})$ $\forall x \in T_W : x = \text{card} \wedge x.\text{title} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"name"}, x.\text{title})$ $\rightarrow \text{AddAttribute}(\text{idwin}, \text{"borderTitle"}, x.\text{title})$ $\forall x \in T_W : x = \text{card} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"isEnabled"}, \text{"true"})$ $\forall x \in T_W : x = \text{card} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"isVisible"}, \text{"true"})$ $\forall x \in T_W : x = \text{card} \rightarrow \text{ConstrBox}(\text{"box"}, \text{"vertical"}, \text{idbox})$ where $\text{idbox} = \sum \text{node} \in T_t$</p>
--

The `card` element is translated by a `window` in the target tree. Another difference with the `body` tag in HTML is that links are implicitly put on every `window` to access the next or previous `card` of the deck. Therefore, a `textComponent` is also added in the UsiXML output for each `window`.

A `template` can also be applied to every `card` of a deck. In our case, only a `template` that redefines these automatic links (`textComponents`) is taken into account, because other cases are beyond the expressiveness of the UsiXML language (related to the history of visited cards for example). In summary, when a `card` is detected, a `window` and two `textComponents` (depending on the template) are

Appendix C

added in the target tree. One of the `textComponent` is added to the previous `card` (to link it with the current `card`) and another to the current `card`, to link it with the previous `card`.

G3 – table (table-box)

$\forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0 \rightarrow \text{Addnode}(\text{"table"}, \text{idtable}) \text{ where } \text{idtable} = \text{NodeAmount}(Tt)$ $\forall x \in T_H : x = \text{table} \wedge (x.\text{border} = 0 \vee x.\text{border} = \text{NULL})$ $\quad \rightarrow \text{Addnode}(\text{"box"}, \text{idtable}) \text{ where } \text{idbox} = \text{NodeAmount}(Tt)$ $\quad \rightarrow \text{AddAttribute}(\text{idtable}, \text{type}, \text{"vertical"})$ $\forall x \in T_W : x = \text{table} \rightarrow \text{Addnode}(\text{"table"}, \text{idtable}) \text{ where } \text{idtable} = \text{NodeAmount}(Tt)$ $\forall x \in T_{H/W} : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"isEnabled"}, \text{"true"})$ $\forall x \in T_{H/W} : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"isVisible"}, \text{"true"})$ $\forall x \in T_H : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"name"}, \text{idtable})$ $\forall x \in T_W : x = \text{table} \wedge x.\text{title} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"name"}, x.\text{title})$ $\forall x \in T_W : x = \text{table} \wedge x.\text{title} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"name"}, \text{idtable})$ $\forall x \in T_{H/W} : x = \text{table} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"id"}, \text{idtable})$ $\forall x \in T_{H/W} : x = \text{table} \wedge x.\text{border} > 0$ $\quad \rightarrow \text{AddAttribute}(\text{idtable}, \text{"xSize"}, (\max_{tr}(\sum_{td} \text{sibling}(\text{td}) \mid \text{td} \in T_{H/W} \wedge \text{NearestInPath}(\text{table}, \text{td}) = x))$ $\forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0$ $\quad \rightarrow \text{AddAttribute}(\text{idtable}, \text{"ySize"}, (\sum_{tr} \text{sibling}(\text{tr}) \mid \text{tr} \in T_{H/W} \wedge \text{NearestInPath}(\text{table}, \text{tr}) = x))$ $\forall x \in T_W : x = \text{table} \wedge x.\text{columns} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{"ySize"}, \text{idtable})$ $\forall x \in T_H : x = \text{table} \wedge x.\text{border} > 0 \rightarrow \text{AddAttribute}(\text{idtable}, \text{"borderwidth"}, x.\text{border})$ $\forall x \in T_H : x = \text{table} \wedge x.\text{width} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"width"}, x.\text{width})$ $\forall x \in T_H : x = \text{table} \wedge x.\text{height} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"height"}, x.\text{height})$ $\forall x \in T_H : x = \text{table} \wedge x.\text{bgimage} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"bgimage"}, x.\text{background})$ $\forall x \in T_H : x = \text{table} \wedge x.\text{bgcolor} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtable}, \text{"bgcolor"}, x.\text{bgcolor})$ $\forall x \in T_{H/W} : x = \text{table} \rightarrow \text{CheckAligment}(x, \text{idtable})$
--

G4 – tr (box)

$\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddNode}(\text{"box"}, \text{idbox}) \text{ where } \text{idbox} = \text{NodeAmount}(Tt)$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddAttribute}(\text{idbox}, \text{"type"}, \text{"horizontal"})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddAttribute}(\text{idbox}, \text{"isEnabled"}, \text{"true"})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddAttribute}(\text{idbox}, \text{"isVisible"}, \text{"true"})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddAttribute}(\text{idbox}, \text{"name"}, \text{idbox})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddAttribute}(\text{idbox}, \text{"id"}, \text{idbox})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{width} \neq \text{NULL}$ $\quad \rightarrow \text{AddAttribute}(\text{idbox}, \text{"width"}, x.\text{width})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{height} \neq \text{NULL}$ $\quad \rightarrow \text{AddAttribute}(\text{idbox}, \text{"height"}, x.\text{height})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{bgimage} = \text{NULL}$ $\quad \rightarrow \text{AddAttribute}(\text{idbox}, \text{"bgimage"}, x.\text{background})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{bgcolor} \neq \text{NULL}$ $\quad \rightarrow \text{AddAttribute}(\text{idbox}, \text{"bgcolor"}, x.\text{bgcolor})$ $\forall x \in T_H : x = \text{tr} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0$
--

Appendix C

$$\rightarrow \text{AddArc}(i.\text{id}, \text{idbox}) \text{ where } i = \text{itao}(\text{NearestInPath}(\text{"table"}, x))$$

$$\forall x \in T_H : x = \text{tr} \rightarrow \text{CheckAlignement}(x, \text{idbox})$$

G5 - td (box/cell)

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0$$

$$\rightarrow \text{AddNode}(\text{"box"}, \text{idbox}) \text{ where } \text{idbox} = \text{NodeAmount}(T_t)$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0 \rightarrow \text{AddNode}(\text{"cell"}, \text{idcell})$$

$$\text{ where } \text{idcell} = \text{NodeAmount}(T_t)$$

$$\forall x \in T_W : x = \text{td} \rightarrow \text{AddNode}(\text{"cell"}, \text{idcell}) \text{ where } \text{idcell} = \text{NodeAmount}(T_t) = \text{NodeAmount}(T_t)$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \rightarrow \text{AddAttribute}(\text{idcell}, \text{"type"}, \text{"horizontal"})$$

$$\forall x \in T_{H/W} : x = \text{td} \rightarrow \text{AddAttribute}(\text{idcell}, \text{"isEnabled"}, \text{"true"})$$

$$\forall x \in T_{H/W} : x = \text{td} \rightarrow \text{AddAttribute}(\text{idcell}, \text{"isVisible"}, \text{"true"})$$

$$\forall x \in T_{H/W} : x = \text{td} \rightarrow \text{AddAttribute}(\text{idcell}, \text{"name"}, \text{idcell})$$

$$\forall x \in T_{H/W} : x = \text{td} \rightarrow \text{AddAttribute}(\text{idcell}, \text{"id"}, \text{idcell})$$

$$\forall x \in T_{H/W} : x = \text{td} \rightarrow \text{CheckAlignement}(x, \text{idcell})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{width} \neq \text{NULL}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"width"}, x.\text{width})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{height} \neq \text{NULL}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"height"}, x.\text{height})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0 \wedge x.\text{bgimage} = \text{NULL}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"bgimage"}, x.\text{background})$$

$$\forall x \in T_H : x = \text{td} \wedge x.\text{bgcolor} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idcell}, \text{"bgcolor"}, x.\text{bgcolor})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} = 0$$

$$\rightarrow \text{AddArc}(i.\text{idbox}, x.\text{idcell}) \text{ where } i = \text{itao}(\text{NearestInPath}(\text{"tr"}, x))$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"xIndex"}, b) \text{ where } b = \sum_{\text{id}} \text{SiblingBefore}(x)$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"yIndex"}, c) : y = \text{NearestInPath}(\text{tr}, x) \wedge c = \sum_{\text{tr}} \text{SiblingBefore}(y)$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0 \wedge x.\text{width} \neq \text{NULL}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"width"}, x.\text{width})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0 \wedge x.\text{height} \neq \text{NULL}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"height"}, x.\text{height})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0 \wedge x.\text{bgimage} = \text{NULL}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"bgimage"}, x.\text{background})$$

$$\forall x \in T_H : x = \text{td} \wedge \text{NearestInPath}(\text{table}, x).\text{border} > 0 \rightarrow \text{AddArc}(i.\text{id}, \text{idbox}) \text{ where}$$

$$i = \text{itao}(\text{NearestInPath}(\text{"table"}, x))$$

$$\forall x \in T_W : x = \text{td} \rightarrow \text{AddAttribute}(\text{idcell}, \text{"xIndex"}, b) \text{ where } b = \sum_{\text{id}} \text{SiblingBefore}(x)$$

$$\forall x \in T_W : x = \text{td}$$

$$\rightarrow \text{AddAttribute}(\text{idcell}, \text{"yIndex"}, c) : y = \text{NearestInPath}(\text{tr}, x) \wedge c = \sum_{\text{tr}} \text{SiblingBefore}(y)$$

$$\forall x \in T_W : x = \text{td} \rightarrow \text{AddArc}(i.\text{id}, \text{idbox}) \text{ where } i = \text{itao}(\text{NearestInPath}(\text{"table"}, x))$$

This group of rules defines the derivation rules for table cells. There are three bold rules (i.e. creating a node in the target tree), as the cell can be a part of a **table** without border (derived then into a **box**), a part of a **table** with borders (derived in a cell) or part of a WML **table** (also derived in a cell). In the first of these three cases, an attribute **type=horizontal** is automatically added as the content of the box

Appendix C

will be displayed horizontally by default in HTML. The five next attributes are common for both languages and each of the 3 cases, `isEnabled`, `isVisible`, `name`, `id` and `alignment` attributes, as the rule can be applied on $T_{H/W}$ trees and has the only condition that the node name is `td`. The four following attributes are for HTML source trees only, but for both cases (i.e. with `border` greater or equal to 0) and add the `width`, `height`, `bgcolor` and `bgImage` attribute in the UsiXML specification if they are present in the source tree. The next attribute defines the values for the `xIndex` and `yIndex` in the CUI model. This rule is only applied if the parent `table` has a `border` greater than 0. These two attributes are computed by counting the number of siblings `td` (and `tr` nodes respectively) before the current node. The same occurs for the WML node, but without `border`-condition as `tables` are never derived into `boxes` in this language.

Finally, the hierarchy is constructed (by adding an arc), either by linking the element to the abstraction of the nearest `table` - for the WML language or in the case of a parent `table` with `border` greater than 0- or to the abstraction of the nearest row (`tr`) in the case of `boxes`.

G6 – fieldset (box)

```

 $\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{Addnode}(\text{"box"}, \text{idbox}) \text{ where } \text{idbox} = \text{NodeAmount}(Tt)
\rightarrow \text{AddAttribute}(\text{idbox}, \text{"type"}, \text{"horizontal"})$ 
 $\forall x \in T_H : x = \text{legend} \wedge \text{NearestInPath}(\text{fieldset}, x) \wedge x.\text{textnode} \neq \text{NULL}
\rightarrow \text{AddAttribute}(i.\text{id}, \text{"borderTitle"}, x.\text{textnode}) : i = (\text{itao}(\text{NearestInPath}(\text{fieldset}, x)))$ 
 $\forall x \in T_H : x = \text{legend} \wedge \text{NearestInPath}(\text{fieldset}, x) \wedge x.\text{textnode} \neq \text{NULL} \wedge x.\text{align} \neq \text{NULL}
\rightarrow \text{AddAttribute}(i.\text{id}, \text{"borderTitleAlign"}, x.\text{align}) : i = (\text{itao}(\text{NearestInPath}(\text{fieldset}, x)))$ 
 $\forall x \in T_H : x = \text{fieldset} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"name"}, \text{idbox})$ 
 $\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"isEnabled"}, \text{"true"})$ 
 $\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"isVisible"}, \text{"true"})$ 
 $\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"id"}, \text{idbox})$ 
 $\forall x \in T_{H/W} : x = \text{fieldset} \rightarrow \text{CheckAlignement}(x, \text{idbox})$ 
 $\forall x \in T_W : x = \text{fieldset} \wedge x.\text{title} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"name"}, x.\text{title})$ 
 $\forall x \in T_W : x = \text{fieldset} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"id"}, \text{idbox})$ 
 $\forall x \in T_W : x = \text{fieldset} \wedge x.\text{title} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbox}, \text{"borderTitle"}, x.\text{title})$ 

```

In this group, rules for WML and HTML source tree are very similar. If a `fieldset` node is detected, a `box` is created in the UsiXML specification with the `type=horizontal` attribute. As for the previous example, five attributes are common to both languages and are automatically added.

In an HTML tree, if a `legend` node is present in the descendant of the `fieldset` node, its `textnode` and `align` attribute are used to set the value of the `borderTitle` and `borderTitleAlign` attributes of the `box` in the target tree.

For a WML tree, the `borderTitle` information can be found in the `title` attribute of the `fieldset` node.

Appendix C

G7 – textbox (textComponent)

$\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL})$
→Addnode ("textComponent", idtext) where idtext=NodeAmount(Tt)
 $\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL})$
→AddAttribute (idtext, "id", idtext) \wedge AddAttribute (idtext, "name", idtext)
 \wedge AddAttribute (idtext, "isVisible", "true")
 $\forall x \in T_{H/W} : x = \text{input} \wedge x.type = \text{"password"} \rightarrow$ AddAttribute (idtext, "isPassword", "true")
 $\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.size \neq \text{NULL}$
→AddAttribute (idtext, "size", x.size)
 $\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.maxLength \neq \text{NULL}$
→ AddAttribute (idtext, "maxLength", x.maxLength)
 $\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.value \neq \text{NULL}$
→AddAttribute (idtext, "defaultContent", x.value)
 $\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.name \neq \text{NULL}$
→AddAttribute (idtext, "varName", x.name)
 $\forall x \in T_H : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.disabled \neq \text{NULL}$
→AddAttribute (idtext, "isEnabled", "false")
 $\forall x \in T_H : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.disabled = \text{NULL}$
→AddAttribute (idtext, "isEnabled", "true")
 $\forall x \in T_W : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL})$
→ AddAttribute (idtext, "isEnabled", "true")
 $\forall x \in T_H : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.accesskey \neq \text{NULL}$
→AddAttribute (idtext, "mnemonic", x.accesskey)
 $\forall x \in T_W : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL}) \wedge x.tabindex \neq \text{NULL}$
→AddAttribute (idtext, "mnemonic", "tab"+ x.tabindex)
 $\forall x \in T_{H/W} : x = \text{input} \wedge (x.type = \text{"text"} \vee x.type = \text{"password"} \vee x.type = \text{NULL})$
→ CheckAlignement(x,idtext)

G8 – select (combobox)

$\forall x \in T_H : x = \text{select} \rightarrow$ Addnode ("comboBox", idcombo) where idcombo = $\sum \text{node} \in T_t$
 $\forall x, y \in T_W : x = \text{select} \wedge (x.multiple = \text{"false"} \vee x.multiple = \text{NULL}) \wedge (\text{CountInChildNodes}(x, \text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y, x) = \text{true} \wedge y.onpick = \text{NULL}$
→ Addnode ("comboBox", idcombo) where idcombo =NodeAmount(Tt)
 $\forall x \in T_H : x = \text{select} \wedge x.name \neq \text{NULL} \rightarrow$ AddAttribute (idcombo, "varName", x.name)
 $\forall x \in T_H : x = \text{select} \wedge x.disabled \neq \text{NULL} \rightarrow$ AddAttribute (idcombo, "isEnabled", "false")
 $\forall x \in T_H : x = \text{select} \wedge x.disabled = \text{NULL} \rightarrow$ AddAttribute (idcombo, "isEnabled", "true")
 $\forall x \in T_H : x = \text{select} \rightarrow$ AddAttribute (idcombo, "id", idcombo)
 $\forall x \in T_H : x = \text{select} \rightarrow$ AddAttribute (idcombo, "name", idcombo)
 $\forall x \in T_H : x = \text{select} \rightarrow$ AddAttribute (idcombo, "isVisible", "true")
 $\forall x \in T_H : x = \text{select} \rightarrow$ AddAttribute (idcombo, "isEditable", "false")
 $\forall x \in T_H : x = \text{select} \wedge (x.size = 1 \vee x.size = \text{NULL})$
→AddAttribute (idcombo, "isDropDown", "false")
 $\forall x \in T_H : x = \text{select} \wedge x.size > 1 \rightarrow$ AddAttribute (idcombo, "isDropDown", "true")
 $\forall x \in T_H : x = \text{select} \wedge x.accesskey \neq \text{NULL}$
→AddAttribute (idcombo, "mnemonic", x.accesskey)
 $\forall x \in T_H : x = \text{select} \rightarrow$ CheckAlignement(x,idcombo)

Appendix C

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{name} \neq \text{NULL}) \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"varName"}, x.\text{name})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{iname} \neq \text{NULL} \wedge x.\text{name} = \text{NULL}) \wedge (x.\text{multiple} = \text{"false"} \vee$
 $x.\text{multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge$
 $y.\text{onpick} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idcombo}, \text{"varName"}, x.\text{iname})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{value} \neq \text{NULL}) \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"varValue"}, x.\text{value})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isEnabled"}, \text{"true"})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"id"}, \text{idcombo})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"name"}, \text{idcombo})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isVisible"}, \text{"true"})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isEditable"}, \text{"false"})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"isDropDown"}, \text{"true"})$

$\forall x,y \in T_W : x = \text{select} \wedge x.\text{tabindex} \neq \text{NULL} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcombo}, \text{"mnemonic"}, x.\text{accesskey})$

$\forall x,y \in T_W : x = \text{select} \wedge (x.\text{multiple} = \text{"false"} \vee x.\text{multiple} = \text{NULL}) \wedge$
 $(\text{CountInChildNodes}(x,\text{option}) > 6) \wedge y = \text{option} \wedge \text{isInPath}(y,x) = \text{true} \wedge y.\text{onpick} = \text{NULL}$
 $\rightarrow \text{CheckAligment}(x,\text{idcombo})$

G9 – option (item)

$\forall x \in T_H : x = \text{option} \rightarrow \text{Addnode}(\text{"item"}, \text{iditem}) \text{ where } \text{iditem} = \text{NodeAmount}(Tt) \wedge$
 $\text{AddArc}(\text{getId}(y), \text{iditem}) : y = \text{itao}(\text{nearestInPath}(\text{"select"}))$

$\forall x,y,z \in T_W : x = \text{option} \wedge z = \text{NearestInPath}(x,\text{"select"}) : (z.\text{multiple} = \text{"false"} \vee$
 $z.\text{multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(z,\text{option}) > 6) \wedge (y \in \text{sibling}(x) : y = \text{option} \wedge$
 $\text{isInPath}(y,z) = \text{true} \wedge y.\text{onpick} = \text{NULL}) \rightarrow \text{Addnode}(\text{"item"}, \text{iditem}) \text{ where } \text{iditem} =$
 $= \text{NodeAmount}(Tt) \wedge \text{AddArc}(\text{getId}(y), \text{iditem}) : y = \text{itao}(\text{nearestInPath}(\text{"select"}))$

$\forall x,y,z \in T_W : x = \text{option} \wedge z = \text{NearestInPath}(x,\text{"select"}) : (z.\text{multiple} = \text{"false"} \vee$
 $z.\text{multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(z,\text{option}) \leq 6) \wedge (y \in \text{sibling}(x) : y = \text{option} \wedge$
 $\text{isInPath}(y,z) = \text{true} \wedge y.\text{onpick} = \text{NULL}) \rightarrow \text{Addnode}(\text{"radioButton"}, \text{iditem}) \text{ where } \text{iditem}$

Appendix C

$\text{=NodeAmount(Tt)} \wedge \text{Addnode}(\text{"textComponent"}, \text{idtext})$ where $\text{idtext} = \text{NodeAmount(Tt)} \wedge \text{AddAttribute}(\text{idtext}, \text{"id"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"name"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isVisible"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isEnabled"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{x.textnode})$

$\forall \text{x,y,z} \in \text{T}_W : \text{x} = \text{option} \wedge \text{z} = \text{NearestInPath}(\text{x}, \text{"select"}) : \text{z.multiple} = \text{"true"} \wedge (\text{y} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge \text{y.onpick} = \text{NULL}) \rightarrow \text{Addnode}(\text{"checkbox"}, \text{iditem})$ where $\text{iditem} = \text{NodeAmount(Tt)} \wedge \text{Addnode}(\text{"textComponent"}, \text{idtext})$ where $\text{idtext} = \text{NodeAmount(Tt)} \wedge \text{AddAttribute}(\text{idtext}, \text{"id"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"name"}, \text{idtext}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isVisible"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isEnabled"}, \text{"true"}) \wedge \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{x.textnode})$

$\forall \text{x,y,z} \in \text{T}_W : \text{x} = \text{option} \wedge \text{z} = \text{NearestInPath}(\text{x}, \text{"select"}) \wedge ((\text{y} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge \text{y.onpick} \neq \text{NULL}) \vee \text{x.onpick} \neq \text{NULL})$

$\rightarrow \text{Addnode}(\text{"textComponent"}, \text{idtext})$ where $\text{idtext} = \text{NodeAmount(Tt)}$

$\forall \text{x} \in \text{T}_H : \text{x} = \text{option} \wedge \text{x.disabled} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"isEnabled"}, \text{"false"})$

$\forall \text{x} \in \text{T}_H : \text{x} = \text{option} \wedge \text{x.disabled} = \text{NULL} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"isEnabled"}, \text{"true"})$

$\forall \text{x} \in \text{T}_{H/W} : \text{x} = \text{option} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"id"}, \text{iditem})$

$\forall \text{x} \in \text{T}_{H/W} : \text{x} = \text{option} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"name"}, \text{iditem})$

$\forall \text{x} \in \text{T}_{H/W} : \text{x} = \text{option} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"isVisible"}, \text{"true"})$

$\forall \text{x} \in \text{T}_{H/W} : \text{x} = \text{option} \wedge \text{x.value} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"varValue"}, \text{x.value})$

$\forall \text{x} \in \text{T}_{H/W} : \text{x} = \text{option} \wedge \text{x.textnode} \neq \text{NULL}$

$\rightarrow \text{AddAttribute}(\text{iditem}, \text{"defaultContent"}, \text{x.textnode})$

$\forall \text{x} \in \text{T}_H : \text{x} = \text{option} \wedge \text{x.accesskey} \neq \text{NULL}$

$\rightarrow \text{AddAttribute}(\text{iditem}, \text{"mnemonic"}, \text{x.accesskey})$

$\forall \text{x} \in \text{T}_W : \text{x} = \text{option} \rightarrow \text{AddAttribute}(\text{iditem}, \text{"isEnabled"}, \text{"true"})$

G9b - option (radio)

$\forall \text{x,y,z} \in \text{T}_W : \text{x} = \text{option} \wedge \text{x.onpick} = \text{NULL} \wedge \text{x.title} \neq \text{NULL} \wedge \text{y} = \text{NearestInPath}(\text{x}, \text{"select"}) : (\text{y.multiple} = \text{"false"} \vee \text{y.multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(\text{y}, \text{option}) \leq 6) \wedge (\text{z} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge \text{z.onpick} = \text{NULL})$

$\rightarrow \text{AddAttribute}(\text{iditem}, \text{"groupName"}, \text{x.title})$

$\forall \text{x,y} \in \text{T}_W : \text{x} = \text{option} \wedge \text{x.onpick} = \text{NULL} \wedge \text{x.value} \neq \text{NULL} \wedge \text{y} = \text{NearestInPath}(\text{x}, \text{"select"}) :$

$(\text{y.multiple} = \text{"false"} \vee \text{y.multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(\text{y}, \text{option}) \leq 6) \wedge$

$(\text{y.ivalue} = \text{x.value}) \wedge (\text{z} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge \text{z.onpick} = \text{NULL})$

$\rightarrow \text{AddAttribute}(\text{iditem}, \text{"DefaultState"}, \text{"checked"})$

$\forall \text{x,y} \in \text{T}_W : \text{x} = \text{option} \wedge \text{y} = \text{NearestInPath}(\text{x}, \text{"select"}) : (\text{y.multiple} = \text{"false"} \vee \text{y.multiple} = \text{NULL}) \wedge (\text{CountInChildNodes}(\text{y}, \text{option}) \leq 6) \wedge (\text{z} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge \text{z.onpick} = \text{NULL}) \rightarrow \text{checkAlignement}(\text{x}, \text{iditem})$

G9c - option (checkbox)

$\forall \text{x,y,z} \in \text{T}_W : \text{x} = \text{option} \wedge \text{x.onpick} = \text{NULL} \wedge \text{x.title} \neq \text{NULL} \wedge \text{y} = \text{NearestInPath}(\text{x}, \text{"select"}) : \text{y.multiple} = \text{"true"} \wedge (\text{z} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge \text{z.onpick} = \text{NULL})$

$\rightarrow \text{AddAttribute}(\text{iditem}, \text{"groupName"}, \text{x.title})$

$\forall \text{x,y,z} \in \text{T}_W : \text{x} = \text{option} \wedge \text{x.onpick} = \text{NULL} \wedge \text{value} \neq \text{NULL} \wedge \text{y} = \text{NearestInPath}(\text{x}, \text{"select"}) :$

$\text{y.multiple} = \text{"true"} \wedge (\text{y.ivalue} = \text{x.value}) \wedge (\text{z} \in \text{sibling}(\text{x}) : \text{y} = \text{option} \wedge \text{isInPath}(\text{y}, \text{z}) = \text{true} \wedge$

$\text{z.onpick} = \text{NULL}) \rightarrow \text{AddAttribute}(\text{iditem}, \text{"DefaultState"}, \text{"checked"})$

Appendix C

$\forall x,y,z \in T_W : x = \text{option} \wedge x.\text{onpick} = \text{NULL} \wedge y = \text{NearestInPath}(x, \text{"select"}) : y.\text{multiple} = \text{"true"} \wedge (z \in \text{sibling}(x) : y = \text{option} \wedge \text{isInPath}(y,z) = \text{true} \wedge z.\text{onpick} = \text{NULL}) \rightarrow \text{checkAlignement}(x, \text{iditem})$

G9d - option (textComponents)

$\forall x,y,z \in T_W : x = \text{option} \wedge z = \text{NearestInPath}(x, \text{"select"}) \wedge ((y \in \text{sibling}(x) : y = \text{option} \wedge \text{isInPath}(y,z) = \text{true} \wedge y.\text{onpick} \neq \text{NULL}) \vee x.\text{onpick} \neq \text{NULL}) \rightarrow \text{AddAttribute}(\text{iditem}, \text{"hyperLinkTarget"}, x.\text{onpick})$

$\forall x,y,z \in T_W : x = \text{option} \wedge z = \text{NearestInPath}(x, \text{"select"}) \wedge ((y \in \text{sibling}(x) : y = \text{option} \wedge \text{isInPath}(y,z) = \text{true} \wedge y.\text{onpick} \neq \text{NULL}) \vee x.\text{onpick} \neq \text{NULL}) \rightarrow \text{checkAlignement}(x, \text{iditem})$

$\forall x,y,z \in T_W : x = \text{option} \wedge z = \text{NearestInPath}(x, \text{"select"}) \wedge ((y \in \text{sibling}(x) : y = \text{option} \wedge \text{isInPath}(y,z) = \text{true} \wedge y.\text{onpick} \neq \text{NULL}) \vee x.\text{onpick} \neq \text{NULL}) \rightarrow \text{AddGrTr}(x, x.\text{onpick})$

G10 – textareas

$\forall x \in T_H : x = \text{textarea} \rightarrow \text{Addnode}(\text{"textComponent"}, \text{idtext})$ where $\text{idtext} = \text{NodeAmount}(Tt)$

$\forall x \in T_H : x = \text{textarea} \wedge x.\text{cols} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtext}, \text{"NumberOfColumns"}, x.\text{cols})$

$\forall x \in T_H : x = \text{textarea} \wedge x.\text{rows} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtext}, \text{"NumberOfColumns"}, x.\text{rows})$

$\forall x \in T_H : x = \text{textarea} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtext}, \text{"varName"}, x.\text{name})$

$\forall x \in T_H : x = \text{textarea} \wedge x.\text{accesskey} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtext}, \text{"mnemonic"}, x.\text{accesskey})$

$\forall x \in T_H : x = \text{textarea} \wedge x.\text{textnode} \neq \text{NULL}$

$\rightarrow \text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, x.\text{accesskey})$

$\forall x \in T_H : x = \text{textarea} \rightarrow \text{CheckAlignement}(x, \text{idtext})$

G11 – input type=button (button)

$\forall x \in T_H : x = \text{input} \wedge (x.\text{type} = \text{"button"} \vee x.\text{type} = \text{"submit"} \vee x.\text{type} = \text{"image"} \vee x.\text{type} = \text{"reset"}) \rightarrow \text{Addnode}(\text{"button"}, \text{idbutton})$ where $\text{idbutton} = \text{NodeAmount}(Tt) \wedge \text{AddAttribute}(\text{idbutton}, \text{"id"}, \text{idbutton}) \wedge \text{AddAttribute}(\text{idbutton}, \text{"name"}, \text{idbutton}) \wedge \text{AddAttribute}(\text{idbutton}, \text{"isVisible"}, \text{"true"})$

$\forall x \in T_H : x = \text{input} \wedge (x.\text{type} = \text{"button"} \vee x.\text{type} = \text{"submit"} \vee x.\text{type} = \text{"image"} \vee x.\text{type} = \text{"reset"}) \wedge x.\text{disabled} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"isEnabled"}, \text{"false"})$

$\forall x \in T_H : x = \text{input} \wedge (x.\text{type} = \text{"button"} \vee x.\text{type} = \text{"submit"} \vee x.\text{type} = \text{"image"} \vee x.\text{type} = \text{"reset"}) \wedge x.\text{disabled} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"isEnabled"}, \text{"true"})$

$\forall x \in T_H : x = \text{input} \wedge (x.\text{type} = \text{"button"} \vee x.\text{type} = \text{"submit"} \vee x.\text{type} = \text{"image"} \vee x.\text{type} = \text{"reset"}) \wedge x.\text{value} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, x.\text{value})$

$\forall x \in T_H : x = \text{input} \wedge x.\text{type} = \text{"submit"} \wedge x.\text{value} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, \text{"submit"})$

$\forall x \in T_H : x = \text{input} \wedge x.\text{type} = \text{"image"} \wedge x.\text{height} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"height"}, x.\text{height})$

$\forall x \in T_H : x = \text{input} \wedge x.\text{type} = \text{"image"} \wedge x.\text{width} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"width"}, x.\text{width})$

$\forall x \in T_H : x = \text{input} \wedge x.\text{type} = \text{"image"} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, x.\text{src})$

$\forall x \in T_H : x = \text{input} \wedge x.\text{type} = \text{"reset"} \wedge x.\text{value} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, \text{"reset"})$

Appendix C

$\forall x \in T_H: x = \text{input} \wedge (x.\text{type} = \text{"button"} \vee x.\text{type} = \text{"submit"} \vee x.\text{type} = \text{"image"} \vee x.\text{type} = \text{"reset"}) \wedge$
 $x.\text{accesskey} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"mnemonic"}, x.\text{accesskey})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{"submit"} \wedge \text{NearestInPath}(x, \text{form}).\text{action} \neq \text{NULL}$
 $\rightarrow \text{AddGraphTr}(\text{idbutton}, \text{NearestInPath}(x, \text{form}).\text{action})$
 $\forall x \in T_H: x = \text{input} \wedge (x.\text{type} = \text{"button"} \vee x.\text{type} = \text{"submit"} \vee x.\text{type} = \text{"image"} \vee x.\text{type} = \text{"reset"})$
 $\rightarrow \text{CheckAlignement}(x, \text{idbutton})$

G12 – button (button)

$\forall x \in T_H: x = \text{button} \rightarrow \text{Addnode}(\text{"button"}, \text{idbutton})$ where $\text{idbutton} = \text{NodeAmount}(Tt)$
 $\forall x \in T_H: x = \text{button} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"id"}, \text{idbutton})$
 $\forall x \in T_H: x = \text{button} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"name"}, \text{idbutton})$
 $\forall x \in T_H: x = \text{button} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"isVisible"}, \text{"true"})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{disabled} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"isEnabled"}, \text{"false"})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{disabled} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"isEnabled"}, \text{"true"})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{type} = \text{"reset"} \wedge x.\text{textnode} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, \text{"reset"})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{accesskey} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idbutton}, \text{"mnemonic"}, x.\text{accesskey})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{type} = \text{"submit"} \wedge x.\text{textnode} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, \text{"submit"})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{textnode} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idbutton}, \text{"defaultContent"}, x.\text{textnode})$
 $\forall x \in T_H: x = \text{button} \wedge x.\text{value} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idbutton}, \text{"varValue"}, x.\text{value})$
 $\forall x \in T_H: x = \text{button} \rightarrow \text{CheckAlignement}(x, \text{idbutton})$

G13 – checkbox (checkbox)

$\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \rightarrow \text{Addnode}(\text{"checkbox"}, \text{idcheck})$ where $\text{idcheck} = \text{NodeAmount}(Tt)$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \wedge x.\text{checked} = \text{true}$
 $\rightarrow \text{AddAttribute}(\text{idcheck}, \text{"defaultState"}, \text{"checked"})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \wedge x.\text{checked} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcheck}, \text{"defaultState"}, \text{"unchecked"})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \wedge x.\text{name} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idcheck}, \text{"varName"}, x.\text{name})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"id"}, \text{idcheck})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"name"}, \text{idcheck})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \rightarrow \text{AddAttribute}(\text{idcheck}, \text{"isVisible"}, \text{"true"})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \wedge (x.\text{disabled} = \text{false} \vee x.\text{disabled} = \text{NULL})$
 $\rightarrow \text{AddAttribute}(\text{idcheck}, \text{"isEnabled"}, \text{"true"})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \wedge (x.\text{disabled} = \text{true} \vee x.\text{disabled} \neq \text{NULL})$
 $\rightarrow \text{AddAttribute}(\text{idcheck}, \text{"isEnabled"}, \text{"false"})$
 $\forall x \in T_H: x = \text{input} \wedge x.\text{type} = \text{checkbox} \rightarrow \text{CheckAlignement}(x, \text{idcheck})$

Appendix C

G14 - radio (radio)

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \rightarrow \text{Addnode}(\text{"radioButton"}, \text{idrad})$ where $\text{idrad} = \text{NodeAmount}(Tt)$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \wedge x.checked = \text{true}$
 $\rightarrow \text{AddAttribute}(\text{idrad}, \text{"defaultState"}, \text{"true"})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \wedge x.name \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idrad}, \text{"groupName"}, x.name)$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \rightarrow \text{AddAttribute}(\text{idrad}, \text{"id"}, \text{idrad})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \rightarrow \text{AddAttribute}(\text{idrad}, \text{"name"}, \text{idrad})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \rightarrow \text{AddAttribute}(\text{idrad}, \text{"isVisible"}, \text{"true"})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \wedge (x.disabled = \text{false} \vee x.disabled = \text{NULL})$
 $\rightarrow \text{AddAttribute}(\text{idrad}, \text{"isEnabled"}, \text{"true"})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \wedge (x.disabled = \text{true} \vee x.disabled \neq \text{NULL})$
 $\rightarrow \text{AddAttribute}(\text{idrad}, \text{"isEnabled"}, \text{"false"})$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \wedge x.accesskey \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrad}, \text{"mnemonic"}, x.accesskey)$

$\forall x \in T_H : x = \text{input} \wedge x.type = \text{radio} \rightarrow \text{CheckAlignement}(x, \text{idrad})$

G15 - img (imageComponent)

$\forall x \in T_{H/W} : x = \text{img} \rightarrow \text{Addnode}(\text{ImageComponent}, \text{idimage})$ where $\text{idimage} = \text{NodeAmount}(Tt)$

$\forall x \in T_{H/W} : x = \text{img} \wedge x.height \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageHeight"}, x.height)$

$\forall x \in T_{H/W} : x = \text{img} \wedge x.width \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageWidth"}, x.width)$

$\forall x \in T_H : x = \text{img} \wedge \text{IsInPath}(a) \rightarrow \text{AddAttribute}(\text{idimage}, \text{"hyperLinkTarget"}, \text{NearestInPath}(x, a).href)$
 $\wedge \text{AddGraphTr}(\text{idimage}, \text{NearestInPath}(x, a).href)$

$\forall x \in T_H : x = \text{img} \wedge x.border \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"borderWidth"}, x.border)$

$\forall x \in T_{H/W} : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"id"}, \text{idimage})$

$\forall x \in T_{H/W} : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"name"}, \text{idimage})$

$\forall x \in T_{H/W} : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"isVisible"}, \text{"true"})$

$\forall x \in T_{H/W} : x = \text{img} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"isEnabled"}, \text{"true"})$

$\forall x \in T_{H/W} : x = \text{img} \wedge x.src \neq \text{NULL} \wedge x.localsrc = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idimage}, \text{"defaultContent"}, x.src)$

$\forall x \in T_W : x = \text{img} \wedge x.localsrc \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idimage}, \text{"defaultContent"}, x.localsrc)$

$\forall x \in T_{H/W} : x = \text{img} \wedge x.hspace \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageHorizSpace"}, x.hspace)$

$\forall x \in T_{H/W} : x = \text{img} \wedge x.vspace \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idimage}, \text{"imageVertSpace"}, x.vspace)$

$\forall x \in T_{H/W} : x = \text{img} \rightarrow \text{CheckAlignement}(x, \text{idimage})$

G16 - area (imagezone)

$\forall x \in T_H : x = \text{area} \rightarrow \text{AddNode}(\text{"imageZone"}, \text{idzone})$ where $\text{idzone} = \text{NodeAmount}(Tt)$
 $\wedge \text{AddArc}(i.id, \text{idzone})$ where $i = \text{itao}(\text{img}) : i.usemap = x.parentNode.name$
 $\wedge \text{AddGraphTr}(\text{idzone}, x.ParentNode.href)$

$\forall x \in T_H : x = \text{area} \wedge x.shape \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idzone}, \text{"shape"}, x.shape)$

$\forall x \in T_H : x = \text{area} \wedge x.coords \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idzone}, \text{"coordinates"}, x.coords)$

Appendix C

$\begin{aligned} \forall x \in T_W : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"big"}, x) = \text{true} \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"textSize"}, \text{"12"}) \\ \forall x \in T_W : x.textnode! = NULL \wedge (x.mode = \text{"wrap"} \vee \text{IsInPath}(p.wrap = \text{true}, x) = \text{true}) \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"wordWrapped"}, \text{"true"}) \\ \forall x \in T_W : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"small"}, x) = \text{true} \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"textSize"}, \text{"8"}) \end{aligned}$
--

G17b - marquee (textcomponent)

$\begin{aligned} \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).behavior! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollStyle"}, \text{NearestInPath}(\text{"marquee"}, x).behavior) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).direction! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollDirection"}, \text{NearestInPath}(\text{"marquee"}, x).direction) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x) = \text{true} \wedge \text{NearestInPath}(\text{"marquee"}, x).direction = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollDirection"}, \text{"left"}) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).height! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollHeight"}, \text{NearestInPath}(\text{"marquee"}, x).height) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).width! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollWidth"}, \text{NearestInPath}(\text{"marquee"}, x).width) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).scrollDelay! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollDelay"}, \text{NearestInPath}(\text{"marquee"}, x).scrollDelay) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).scrollAmount! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollAmount"}, \text{NearestInPath}(\text{"marquee"}, x).scrollAmount) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).hspace! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollHorizSpace"}, \text{NearestInPath}(\text{"marquee"}, x).hspace) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).vspace! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"scrollVertSpace"}, \text{NearestInPath}(\text{"marquee"}, x).vspace) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"marquee"}, x).bgColor! = NULL \\ \rightarrow \text{AddAttribute}(\text{idtext}, \text{"bgColor"}, \text{NearestInPath}(\text{"marquee"}, x).bgColor) \end{aligned}$

G17c - Links (textComponent)

$\begin{aligned} \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{NearestInPath}(\text{"a"}, x).href! = NULL \rightarrow \text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, \\ \text{NearestInPath}(\text{"a"}, x).href) \wedge \text{AddGraphTr}(\text{idtext}, \text{NearestInPath}(x, \text{"a"}).href) \\ \forall x \in T_H : x.textnode! = NULL \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge \\ \text{IsInPath}(\text{"a"}, x) = \text{true} \wedge \text{NearestInPath}(\text{"a"}, x).href! = NULL \wedge \text{body.vlink!} = NULL \end{aligned}$

Appendix C

$\rightarrow \text{AddAttribute}(\text{idtext}, \text{"linkVisitedColor"}, \text{body.vlink})$
 $\forall x \in T_H : x.\text{textnode} \neq \text{NULL} \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge$
 $\text{IsInPath}(\text{"a"}, x) = \text{true} \wedge \text{NearestInPath}(\text{"a"}, x).\text{href} \neq \text{NULL} \wedge \text{body.alink} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idtext}, \text{"activeLinkColor"}, \text{body.alink})$
 $\forall x \in T_H : x.\text{textnode} \neq \text{NULL} \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge$
 $\text{IsInPath}(\text{"a"}, x) = \text{true} \wedge \text{NearestInPath}(\text{"a"}, x).\text{href} \neq \text{NULL} \wedge \text{body.link} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idtext}, \text{"textColor"}, \text{body.link})$
 $\forall x \in T_H : x.\text{textnode} \neq \text{NULL} \wedge \text{IsInPath}(\text{"select"}, x) = \text{false} \wedge \text{IsInPath}(\text{"textarea"}, x) = \text{false} \wedge$
 $\text{IsInPath}(\text{"a"}, x) = \text{true} \wedge \text{NearestInPath}(\text{"a"}, x).\text{href} \neq \text{NULL} \wedge x.\text{accesskey} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idtext}, \text{"mnemonic"}, x.\text{accesskey})$

G18 – applets / flash (finalComponent)

$\forall x \in T_H : x = (\text{applet} \vee \text{embed} \wedge \text{isAudio}(x) = \text{false}) \rightarrow \text{Addnode}(\text{"finalComponent"}, \text{idfc})$
where idfc = NodeAmount(Tt)
 $\forall x \in T_H : x = (\text{applet} \vee (\text{embed} \wedge \text{isAudio}(x) = \text{false})) \rightarrow \text{AddAttribute}(\text{idfc}, \text{"id"}, \text{idfc})$
 $\forall x \in T_H : x = (\text{applet} \vee (\text{embed} \wedge \text{isAudio}(x) = \text{false})) \rightarrow \text{AddAttribute}(\text{idfc}, \text{"name"}, \text{idfc})$
 $\forall x \in T_H : x = (\text{applet} \vee (\text{embed} \wedge \text{isAudio}(x) = \text{false})) \rightarrow \text{AddAttribute}(\text{idfc}, \text{"isVisible"}, \text{"true"})$
 $\forall x \in T_H : x = (\text{applet} \vee (\text{embed} \wedge \text{isAudio}(x) = \text{false})) \rightarrow \text{AddAttribute}(\text{idfc}, \text{"isEnabled"}, \text{"true"})$
 $\forall x \in T_H : x = (\text{applet} \vee (\text{embed} \wedge \text{isAudio}(x) = \text{false})) \wedge x.\text{src} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idfc}, \text{"location"}, x.\text{src})$
 $\forall x \in T_H : x = (\text{applet} \vee (\text{embed} \wedge \text{isAudio}(x) = \text{false})) \rightarrow \text{CheckAlignement}(x, \text{idfc})$

G19 – bgsound (auditoryoutput)

$\forall x \in T_H : x = \text{bgsound}$
 $\rightarrow \text{Addnode}(\text{"auditoryOutput"}, \text{idaudio})$ **where idaudio = NodeAmount(Tt)**
 $\forall x \in T_H : x = \text{bgsound} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"id"}, \text{idaudio})$
 $\forall x \in T_H : x = \text{bgsound} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idaudio})$
 $\forall x \in T_H : x = \text{bgsound} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isEnabled"}, \text{"true"})$
 $\forall x \in T_H : x = \text{bgsound} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isVisible"}, \text{"true"})$
 $\forall x \in T_H : x = \text{bgsound} \wedge x.\text{loop} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isLooped"}, x.\text{loop})$
 $\forall x \in T_H : x = \text{bgsound} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"tone"}, x.\text{src})$
 $\forall x \in T_H : x = \text{bgsound} \rightarrow \text{CheckAlignement}(x, \text{idaudio})$

G20 – embed (auditoryoutput)

$\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true}$
 $\rightarrow \text{Addnode}(\text{"auditoryOutput"}, \text{idaudio})$ **where idaudio = NodeAmount(Tt)**
 $\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"id"}, \text{idaudio})$
 $\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idaudio})$
 $\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isEnabled"}, \text{"true"})$
 $\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true} \wedge x.\text{hidden} = \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isVisible"}, \text{"false"})$
 $\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true} \wedge x.\text{hidden} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isVisible"}, \text{"true"})$
 $\forall x \in T_H : x = \text{embed} \wedge \text{isAudio}(x) = \text{true} \wedge x.\text{loop} \neq \text{NULL}$

Appendix C

$\rightarrow \text{AddAttribute}(\text{idbutton}, \text{"isLooped"}, \text{x.loop})$
 $\forall x \in T_H: x = \text{embed} \wedge \text{IsAudio}(x) = \text{true} \wedge \text{x.src} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idbutton}, \text{"tone"}, \text{x.src})$
 $\forall x \in T_H: x = \text{embed} \wedge \text{IsAudio}(x) = \text{true} \rightarrow \text{CheckAlignement}(\text{x}, \text{idaudio})$

G21 – do/go (textComponent)

$\forall x, y \in T_W: x = \text{do} \wedge \text{x.type} = \text{accept} \wedge \text{y} = \text{go} \wedge \text{y} \in \text{childNodes}(x) \rightarrow \text{AddNode}$
(“TextComponent”, idtext) where idtext=NodeAmount(Tt)
 $\forall x, y \in T_W: x = \text{do} \wedge \text{x.type} = \text{accept} \wedge \text{x.label} \neq \text{NULL} \wedge \text{y} = \text{go} \wedge \text{y} \in \text{childNodes}(x) \rightarrow$
 $\text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{x.label}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"})$
 $\forall x, y \in T_W: x = \text{do} \wedge \text{x.type} = \text{accept} \wedge \text{y.href} \neq \text{NULL} \wedge \text{y} = \text{go} \wedge \text{y} \in \text{childNodes}(x) \rightarrow$
 $\text{AddAttribute}(\text{idtext}, \text{"defaultContent"}, \text{x.label}) \wedge \text{AddAttribute}(\text{idtext}, \text{"isbold"}, \text{"true"})$
 $\text{AddAttribute}(\text{idtext}, \text{"hyperLinkTarget"}, \text{y.href}) \wedge \text{AddGraphTr}(\text{idtext}, \text{y.href})$

G22 – br

$\forall x \in T_H: x = \text{br} \rightarrow \text{Addnode}(\text{"br"}, \text{idbr}) \text{ where idbr} = \text{NodeAmount}(\text{Tt})$

G23 –vxml (vocalGroup)

$\forall x \in T_V: x = \text{vxml} \rightarrow \text{Addnode}(\text{"vocalGroup"}, \text{idvxml}) \text{ where idvxml} = \text{NodeAmount}(\text{Tt})$
 $\forall x \in T_V: x = \text{vxml} \rightarrow \text{AddAttribute}(\text{idvxml}, \text{"id"}, \text{idvxml})$
 $\forall x \in T_V: x = \text{vxml} \rightarrow \text{AddAttribute}(\text{idvxml}, \text{"name"}, \text{idvxml})$
 $\forall x \in T_V: x = \text{vxml} \rightarrow \text{AddAttribute}(\text{idvxml}, \text{"filename"}, \text{x.filename})$

G24 – field (vocalInput)

$\forall w, x \in T_V: w = \text{field} \wedge \text{x} = \text{prompt} \wedge \text{x} \in \text{childNodes}(w) \wedge \text{x.textnode} \neq \text{NULL} \wedge (\text{x.count} = \text{NULL} \vee \text{x.count} = 1)$
 $\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where idprompt} = \text{NodeAmount}(\text{Tt})$
 $\forall w, x \in T_V: w = \text{field} \wedge \text{x} = \text{prompt} \wedge \text{x} \in \text{childNodes}(w) \wedge \text{x.textnode} \neq \text{NULL} \wedge (\text{x.count} = \text{NULL} \vee \text{x.count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$
 $\forall w, x \in T_V: w = \text{field} \wedge \text{x} = \text{prompt} \wedge \text{x} \in \text{childNodes}(w) \wedge \text{x.textnode} \neq \text{NULL} \wedge (\text{x.count} = \text{NULL} \vee \text{x.count} = 1) \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idprompt})$
 $\forall w, x \in T_V: w = \text{field} \wedge \text{x} = \text{prompt} \wedge \text{x} \in \text{childNodes}(w) \wedge \text{x.bargein} = \text{false} \wedge \text{x.textnode} \neq \text{NULL} \wedge (\text{x.count} = \text{NULL} \vee \text{x.count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"isInterruptible"}, \text{"false"})$
 $\forall w, x \in T_V: w = \text{field} \wedge \text{x} = \text{prompt} \wedge \text{x} \in \text{childNodes}(w) \wedge \text{x.textnode} \neq \text{NULL} \wedge (\text{x.count} = \text{NULL} \vee \text{x.count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, \text{x.textnode})$

 $\forall x \in T_V: x = \text{field} \rightarrow \text{Addnode}(\text{"vocalInput"}, \text{idfield}) \text{ where idfield} = \text{NodeAmount}(\text{Tt})$
 $\forall x \in T_V: x = \text{field} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"id"}, \text{idfield})$
 $\forall x \in T_V: x = \text{field} \wedge \text{x.name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"name"}, \text{x.name})$
 $\forall x \in T_V: x = \text{field} \wedge \text{x.name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"name"}, \text{idfield})$
 $\forall x \in T_V: x = \text{field} \wedge \text{x.type} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"grammar"}, \text{x.type})$
 $\forall x \in T_V: x = \text{field} \wedge \text{x.expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idfield}, \text{"currentValue"}, \text{x.expr})$

Appendix C

$$\begin{aligned} &\forall x \in T_V : x = \text{field} \wedge x.type = \text{NULL}, \exists y, z \in T_V : y = \text{NearestInPath}(\text{form}, x) \wedge z = \text{grammar} \wedge z \in \\ &\text{childnodes}(y) \wedge z.textnode \neq \text{NULL} \wedge \text{grammar} \notin \text{childnodes}(x) \rightarrow \text{AddAttribute}(\text{idfield}, \\ &\text{"grammar"}, z.textnode) \wedge \text{AddAttribute}(\text{idfield}, \text{"isOrderIndependent"}, \text{"true"}) \\ &\forall x \in T_V : x = \text{field} \wedge x.type = \text{NULL}, \exists y, z \in T_V : y = \text{NearestInPath}(\text{form}, x) \wedge z = \text{grammar} \wedge z \in \\ &\text{childnodes}(y) \wedge z.src \neq \text{NULL} \wedge \text{grammar} \notin \text{childnodes}(x) \rightarrow \text{AddAttribute}(\text{idfield}, \text{"grammar"}, \\ &x.src) \wedge \text{AddAttribute}(\text{idfield}, \text{"isOrderIndependent"}, \text{"true"}) \end{aligned}$$

G25 – menu (vocalMenu)

$$\begin{aligned} &\forall x \in T_V : x = \text{menu} \rightarrow \text{Addnode}(\text{"vocalMenu"}, \text{idmenu}) \text{ where } \text{idmenu} = \text{NodeAmount}(Tt) \\ &\forall x \in T_V : x = \text{menu} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"id"}, \text{idmenu}) \\ &\forall x \in T_V : x = \text{menu} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"name"}, x.name) \\ &\forall x \in T_V : x = \text{menu} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"id"}, x.name) \\ &\forall x \in T_V : x = \text{menu} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"name"}, \text{idmenu}) \\ &\forall x \in T_V : x = \text{menu} \wedge x.dtmf \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"keyboardShortcut"}, x.dtmf) \\ &\forall x \in T_V : x = \text{menu} \wedge x.expr \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"currentValue"}, x.expr) \\ &\forall x \in T_V : x = \text{menu} \wedge x.scope = \text{"document"} \\ &\quad \rightarrow \text{AddAttribute}(\text{idmenu}, \text{"isOrderIndependent"}, \text{"true"}) \end{aligned}$$

G26 – block (vocalGroup)

$$\begin{aligned} &\forall x \in T_V : x = \text{block} \rightarrow \text{Addnode}(\text{"vocalGroup"}, \text{idblock}) \text{ where } \text{idblock} = \text{NodeAmount}(Tt) \\ &\forall x \in T_V : x = \text{block} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idblock}, \text{"id"}, \text{idblock}) \\ &\forall x \in T_V : x = \text{block} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idblock}, \text{"name"}, x.name) \\ &\forall x \in T_V : x = \text{block} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idblock}, \text{"id"}, x.name) \\ &\forall x \in T_V : x = \text{block} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idblock}, \text{"name"}, \text{idblock}) \\ &\forall x \in T_V : x = \text{block} \wedge x.textnode \neq \text{NULL} \\ &\quad \rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(Tt) \\ &\forall x \in T_V : x = \text{block} \wedge x.textnode \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt}) \\ &\forall x \in T_V : x = \text{block} \wedge x.textnode \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"name"}, \text{idprompt}) \\ &\forall x \in T_V : x = \text{block} \wedge x.textnode \neq \text{NULL} \\ &\quad \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, x.textnode) \end{aligned}$$

The next set of rules derives the initial node into a `vocalGroup` and a `vocalInput`. This container indicates that its elements have to be processed before the rest of the form and it is usually specified for mixed-initiative forms. When a `grammar` is specified at the `form` level, the initial element `prompts` the user for form-wide information.

Therefore, when this element is derived, the `vocalInput`'s attribute `isOrderIndependent` is set to true as it allows filling the fields of the form in any order. The `vocalGroup` created thus contains a `vocalInput` but also a `vocalPrompt` before this `vocalInput`. The children of the initial element are checked to find a `prompt` element, and a `vocalPrompt` is generated before the creation of the `vocalInput` to represent the elements in a logical order (the rule set G35 for

Appendix C

prompts checks if the **prompt** element is not embedded in an **initial** or **field** element to avoid a double-processing of the node).

The attribute **expr** of the **initial** node must be NULL (or undefined in VoiceXML) to activate the node in a vocal browser, therefore the value of this attribute is checked in every rule of G27. If it is not the case, the element is not derived as the run-time modification of its value to undefined (and thus the activation of the **initial** element) can not be recorded in UsiXML.

Finally, the **initial** element is not directly translated in a **vocalGroup** but in a **vocalGroupInit** because this special name is used to mark the element for a future transformation (the rest of the form will be put in another **vocalGroup** see set of rules G53).

G27 – initial (vocalGroup-VocalInput)

<p> $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL}$ $\rightarrow \text{Addnode}(\text{"vocalGroupInit"}, \text{idinitial}) \text{ where } \text{idinitial} = \text{NodeAmount}(\text{Tt})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{name} = \text{NULL} \wedge x.\text{expr} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idinitial}, \text{"id"}, \text{idinitial})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{name} = \text{NULL} \wedge x.\text{expr} = \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{idinitial}, \text{"name"}, \text{idinitial})$ $\forall w, x \in T_V : w = \text{initial} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \text{IsInPath}(\text{field}, x) = \text{false} \rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \wedge \text{AddArc}(\text{idinitial}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(\text{Tt})$ $\forall w, x \in T_V : w = \text{initial} \wedge w.\text{expr} = \text{NULL} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$ $\forall w, x \in T_V : w = \text{field} \wedge w.\text{expr} = \text{NULL} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idprompt})$ $\forall w, x \in T_V : w = \text{initial} \wedge w.\text{expr} = \text{NULL} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{bargain} = \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \text{IsInPath}(\text{field}, x) = \text{false}$ $\rightarrow \text{AddAttribute}(\text{idprompt}, \text{"isInterruptible"}, \text{"false"})$ $\forall w, x \in T_V : w = \text{initial} \wedge w.\text{expr} = \text{NULL} \wedge x = \text{prompt} \wedge x \in \text{childNodes}(w) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \text{IsInPath}(\text{field}, x) = \text{false}$ $\rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, x.\text{textnode})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL} \rightarrow \text{Addnode}(\text{"vocalInput"}, \text{idvii}) \wedge \text{AddArc}(\text{idinitial}, \text{idvii}) \text{ where } \text{idvii} = \text{NodeAmount}(\text{Tt})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL} \wedge x.\text{name} \neq \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{idvii}, \text{"name"}, x.\text{name})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL} \wedge x.\text{name} \neq \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{idvii}, \text{"name"}, x.\text{name})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idvii}, \text{"name"}, \text{idvii})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idvii}, \text{"id"}, \text{idvii})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL}, \exists y, z \in T_V : y = \text{NearestInPath}(\text{form}, x) \wedge (z = \text{grammar} \wedge z \in \text{childNodes}(y) \wedge z.\text{textnode} \neq \text{NULL}) \wedge \text{grammar} \notin \text{childNodes}(x) \rightarrow \text{AddAttribute}(\text{idvii}, \text{"grammar"}, z.\text{textnode}) \wedge \text{AddAttribute}(\text{idvii}, \text{"isOrderIndependent"}, \text{"true"})$ $\forall x \in T_V : x = \text{initial} \wedge x.\text{expr} = \text{NULL}, \exists y, z \in T_V : y = \text{NearestInPath}(\text{form}, x) \wedge (z = \text{grammar} \wedge z \in \text{childNodes}(y) \wedge z.\text{src} \neq \text{NULL}) \wedge \text{grammar} \notin \text{childNodes}(x)$ $\rightarrow \text{AddAttribute}(\text{idvii}, \text{"grammar"}, x.\text{src}) \wedge \text{AddAttribute}(\text{idvii}, \text{"isOrderIndependent"}, \text{"true"})$ </p>
--

Appendix C

G28 – Form (vocalForm)

$\forall x \in T_V: x = \text{form} \rightarrow \text{Addnode}(\text{"vocalForm"}, \text{idform})$ where $\text{idform} = \text{NodeAmount}(Tt)$
$\forall x \in T_V: x = \text{form} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"id"}, x.name)$
$\forall x \in T_V: x = \text{form} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"id"}, \text{idform})$
$\forall x \in T_V: x = \text{form} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"name"}, x.name)$
$\forall x \in T_V: x = \text{form} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"name"}, \text{idform})$
$\forall x \in T_V: x = \text{form} \wedge x.expr \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idform}, \text{"currentValue"}, x.expr)$
$\forall x, y \in T_V: x = \text{form} \wedge x.scope = \text{"document"}$ $\rightarrow \text{AddAttribute}(\text{idform}, \text{"isOrderIndependent"}, \text{"true"})$
$\forall x, z \in T_V: x = \text{form} \wedge z = \text{grammar} \wedge z \in \text{childnodes}(x)$ $\rightarrow \text{AddAttribute}(\text{idform}, \text{"isOrderIndependent"}, \text{"true"})$

The group of rules G28 is only applicable to the voiceXML language (T_V). It derives a **form** node into a **vocalForm** element in the CUI model. If the **name** attribute is present, its value will be used to set the values of the **name** and **id** UsiXML attributes. In the other case, the **name** and **id** attributes are computed by counting the number of node already created in the target tree. The value of the **expr** attribute is copied in the **currentValue** attribute in the target tree. Finally, if a **grammar** node exists in the children of the **form**, then the attribute **isOrderIndependent** is set to true in the CUI model, as a grammar defined at the level of the **form** will be active until the **form** node is closed. The information related to the **grammar** are added later (in G38) to the **vocalForm**.

G29 – transfer (vocalNavigation)

$\forall x \in T_V: x = \text{transfer}$ $\rightarrow \text{Addnode}(\text{"vocalNavigation"}, \text{idtransfer})$ where $\text{idtransfer} = \text{NodeAmount}(Tt)$
$\forall x \in T_V: x = \text{transfer} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"navigationType"}, \text{"transfer"})$
$\forall x \in T_V: x = \text{transfer} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"id"}, x.name)$
$\forall x \in T_V: x = \text{transfer} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"id"}, \text{idtransfer})$
$\forall x \in T_V: x = \text{transfer} \wedge x.name \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"name"}, x.name)$
$\forall x \in T_V: x = \text{transfer} \wedge x.name = \text{NULL} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"name"}, \text{idtransfer})$
$\forall x \in T_V: x = \text{transfer} \wedge x.expr \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"currentValue"}, x.expr)$
$\forall x \in T_V: x = \text{transfer} \wedge x.bridge = \text{"true"} \rightarrow \text{AddAttribute}(\text{idtransfer}, \text{"isBridgeable"}, \text{"true"})$
$\forall x \in T_V: x = \text{transfer} \wedge x.dest \neq \text{NULL} \rightarrow \text{AddAudiTr}(\text{idtransfer}, x.dest)$
$\forall x \in T_V: x = \text{transfer} \wedge x.transferaudio \neq \text{NULL}$ $\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt})$ where $\text{idprompt} = \text{NodeAmount}(Tt) \sim \text{optional}$
$\forall x \in T_V: x = \text{transfer} \wedge x.transferaudio \neq \text{NULL} \rightarrow \text{Addattribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$
$\forall x \in T_V: x = \text{transfer} \wedge x.transferaudio \neq \text{NULL} \rightarrow \text{Addattribute}(\text{idprompt}, \text{"name"}, \text{idprompt})$
$\forall x \in T_V: x = \text{transfer} \wedge x.transferaudio \neq \text{NULL}$ $\rightarrow \text{AddCUIDiagCont}(\text{idtransfer}, \text{idprompt}, \text{" "})$

G30 – filled (vocalPrompt)

$\forall x \in T_V: x = \text{filled} \wedge x.textnode \neq \text{NULL}$

Appendix C

→ Addnode (“vocalPrompt”, idfilled) where idprompt =NodeAmount(Tt)
 $\forall x \in T_V : x = \text{filled} \wedge x.\text{textnode} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{id filled}, \text{“id”}, \text{idfilled})$
 $\forall x \in T_V : x = \text{filled} \wedge x.\text{textnode} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{“name”}, \text{id filled})$
 $\forall x \in T_V : x = \text{filled} \wedge x.\text{textnode} \neq \text{NULL}$
→ AddAttribute (idprompt, “defaultContent”, x.textnode)

G31 – subdialog (vocalGroup)

$\forall x \in T_V : x = \text{subdialog} \rightarrow \text{Addnode}(\text{“vocalGroup”}, \text{idsub})$ where idsub =NodeAmount(Tt)
 $\forall x \in T_V : x = \text{subdialog} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idsub}, \text{“id”}, \text{idsub})$
 $\forall x \in T_V : x = \text{subdialog} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idsub}, \text{“name”}, x.\text{name})$
 $\forall x \in T_V : x = \text{subdialog} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idsub}, \text{“id”}, x.\text{name})$
 $\forall x \in T_V : x = \text{subdialog} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idsub}, \text{“name”}, \text{idsub})$
 $\forall x \in T_V : x = \text{subdialog} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idsub}, \text{“insertfile”}, x.\text{src})$

G32 – object (finalComponent)

$\forall x \in T_V : x = \text{object}$
→ Addnode (“finalComponent”, idobject) where idobject =NodeAmount(Tt)
 $\forall x \in T_V : x = \text{object} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idobject}, \text{“id”}, \text{idobject})$
 $\forall x \in T_V : x = \text{object} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idobject}, \text{“id”}, x.\text{name})$
 $\forall x \in T_V : x = \text{object} \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idobject}, \text{“name”}, x.\text{name})$
 $\forall x \in T_V : x = \text{object} \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idobject}, \text{“name”}, \text{idobject})$
 $\forall x \in T_V : x = \text{object} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idobject}, \text{“currentValue”}, x.\text{expr})$
 $\forall x \in T_V : x = \text{object} \wedge x.\text{classid} \neq \text{NULL} \wedge x.\text{codebase} = \text{NULL}$
→ AddAttribute (idobject, “defaultContent”, x.codeid)
 $\forall x \in T_V : x = \text{object} \wedge x.\text{classid} \neq \text{NULL} \wedge x.\text{codebase} \neq \text{NULL}$
→ AddAttribute (idobject, “defaultContent”, x.codebase+x.classid)

G33 – link (vocalNavigation)

$\forall x \in T_V : x = \text{link} \rightarrow \text{Addnode}(\text{“vocalNavigation”}, \text{idlink})$ where idlink =NodeAmount(Tt)
 $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{“id”}, \text{idlink})$
 $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{“name”}, \text{idlink})$
 $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{“NavigationType”}, \text{“link”})$
 $\forall x \in T_V : x = \text{link} \rightarrow \text{AddAttribute}(\text{idlink}, \text{“isBridgeable”}, \text{“false”})$
 $\forall x \in T_V : x = \text{link} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idlink}, \text{“currentValue”}, x.\text{expr})$
 $\forall x \in T_V : x = \text{link} \wedge x.\text{dtmf} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idlink}, \text{“keyboardShortcut”}, x.\text{dtmf})$
 $\forall x \in T_V : x = \text{link} \wedge x.\text{next} \neq \text{NULL} \rightarrow \text{AddAudiTr}(\text{idlink}, x.\text{next})$

G34 – audio (vocalPrompt)

$\forall x \in T_V : x = \text{audio}$
→ Addnode (“vocalPrompt”, idaudio) where idaudio =NodeAmount(Tt)
 $\forall x \in T_V : x = \text{audio} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{“id”}, \text{idaudio})$
 $\forall x \in T_V : x = \text{audio} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{“name”}, \text{idaudio})$

Appendix C

$\begin{aligned} &\forall x \in T_V : x = \text{audio} \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"defaultContent"}, x.\text{expr}) \\ &\forall x \in T_V : x = \text{audio} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"defaultContent"}, x.\text{src}) \\ &\forall x, p \in T_V : x = \text{audio} \wedge p = \text{prompt} \wedge p.\text{bargain} = \text{false} \wedge \text{IsInPath}(p, x) = \text{true} \\ &\quad \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"isInterruptible"}, \text{"false"}) \end{aligned}$

G35 – prompt (vocalPrompt)

$\begin{aligned} &\forall x \in T_V : x = \text{prompt} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \\ &\text{IsInPath}(\text{field}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \\ &\quad \rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(\text{Tt}) \\ &\forall x \in T_V : x = \text{prompt} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \\ &\text{IsInPath}(\text{field}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt}) \\ &\forall x \in T_V : x = \text{prompt} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \\ &\text{IsInPath}(\text{field}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idprompt}) \\ &\forall x \in T_V : x = \text{prompt} \wedge x.\text{bargain} = \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \\ &\wedge \text{IsInPath}(\text{field}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \\ &\quad \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"isInterruptible"}, \text{"false"}) \\ &\forall x \in T_V : x = \text{prompt} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \wedge \\ &\text{IsInPath}(\text{field}, x) = \text{false} \wedge \text{IsInPath}(\text{initial}, x) = \text{false} \\ &\quad \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, x.\text{textnode}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \\ &\wedge \text{parentNode}(y) = x \\ &\quad \rightarrow \text{Addnode}(\text{"vocalInput"}, \text{idrecord}) \text{ where } \text{idrecord} = \text{NodeAmount}(\text{Tt}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"id"}, \text{idrecord}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"id"}, x.\text{name}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{name} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"name"}, x.\text{name}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{name} = \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"name"}, \text{idrecord}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{expr} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"currentValue"}, x.\text{expr}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{maxtime} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"elapsedTime"}, x.\text{maxtime}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x \wedge x.\text{dtmfterm} \neq \text{NULL} \\ &\quad \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"keyboardShortcut"}, x.\text{dtmfterm}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x, \exists w, z \in T_V : w = \text{NearestInPath}(\text{form}, x) \wedge z = \text{grammar} \wedge z \in \text{childnodes}(w) \wedge \\ &z.\text{textnode} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"grammar"}, z.\text{textnode}) \\ &\forall x, y \in T_V : y = \text{prompt} \wedge y.\text{textnode} \neq \text{NULL} \wedge (y.\text{count} = \text{NULL} \vee y.\text{count} = 1) \wedge x = \text{record} \wedge \\ &\text{parentNode}(y) = x, \exists w, z \in T_V : w = \text{NearestInPath}(\text{form}, x) \wedge z = \text{grammar} \wedge z \in \text{childnodes}(w) \wedge \\ &z.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idrecord}, \text{"grammar"}, z.\text{src}) \end{aligned}$

Appendix C

The next set is an optional set of rules; it means that the designer can choose to apply it to recover the element but without the conditional structure. In this case, the **reprompt** element is transformed into a **vocalPrompt** element. The effect of the **reprompt** tag in a VoiceXML specification is the repetition of the last played **prompt**. The rule G36 searches the nearest **prompt** in the VoiceXML code, and if this nearest **prompt** should be played without condition (**count=NULL** or **count=1**), then the **reprompt** element is mapped as a **vocalPrompt** containing the same **textnode** as the nearest **prompt**.

G36 – Reprompt (vocalPrompt) ~ optional

$\forall y \in T_V : y = \text{reprompt}, \exists x \in T_V : x = \text{prompt} \wedge \text{isInPath}(x,y) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1)$ $\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(Tt)$ $\forall y \in T_V : y = \text{reprompt}, \exists x \in T_V : x = \text{prompt} \wedge \text{isInPath}(x,y) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$ $\forall y \in T_V : y = \text{reprompt}, \exists x \in T_V : x = \text{prompt} \wedge \text{isInPath}(x,y) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idprompt})$ $\forall y \in T_V : y = \text{reprompt}, \exists x \in T_V : x = \text{prompt} \wedge \text{isInPath}(x,y) \wedge x.\text{bargain} = \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"isInterruptible"}, \text{"false"})$ $\forall y \in T_V : y = \text{reprompt}, \exists x \in T_V : x = \text{prompt} \wedge \text{isInPath}(x,y) \wedge x.\text{textnode} \neq \text{NULL} \wedge (x.\text{count} = \text{NULL} \vee x.\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, x.\text{textnode})$

G37 – submit (vocalNavigation)

$\forall x \in T_V : x = \text{submit} \rightarrow \text{Addnode}(\text{"vocalNavigation"}, \text{idsubmit})$ $\text{ where } \text{idsubmit} = \text{NodeAmount}(Tt)$ $\forall x \in T_V : x = \text{submit} \rightarrow \text{AddAttribute}(\text{idsubmit}, \text{"id"}, \text{idsubmit})$ $\forall x \in T_V : x = \text{submit} \rightarrow \text{AddAttribute}(\text{idsubmit}, \text{"name"}, \text{idsubmit})$ $\forall x \in T_V : x = \text{submit} \rightarrow \text{AddAttribute}(\text{idsubmit}, \text{"NavigationType"}, \text{"submit"})$ $\forall x \in T_V : x = \text{submit} \rightarrow \text{AddAttribute}(\text{idsubmit}, \text{"isBridgeable"}, \text{"false"})$ $\forall x \in T_V : x = \text{submit} \wedge x.\text{next} \neq \text{NULL} \rightarrow \text{AddAudiTr}(\text{idsubmit}, x.\text{next})$
--

G38 – grammar ()

$\forall x,z \in T_V : x = \text{grammar} \wedge z = \text{parentNode}(x) \wedge z = \text{form} \wedge x.\text{scope} = \text{document}, \exists y \in T_t : y = \text{itao}(z) \rightarrow \text{Addattribute}(y.\text{id}, \text{"isOrderIndependent"}, \text{"true"})$ $\forall x \in T_V : x = \text{grammar}, \exists y \in T_t : y = \text{itao}(\text{parentNode}(x)) \wedge y = \text{VocalForm}$ $\rightarrow \text{Addattribute}(y.\text{id}, \text{"isOrderIndependent"}, \text{"true"})$ $\forall x \in T_V : x = \text{grammar}, \exists y \in T_t : y = \text{itao}(\text{parentNode}(x)) \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{parentNode}(x) = \text{field} \vee \text{parentNode}(x) = \text{record} \vee \text{parentNode}(x) = \text{initial})$ $\rightarrow \text{Addattribute}(y.\text{id}, \text{"grammar"}, x.\text{textnode})$ $\forall x \in T_V : x = \text{grammar}, \exists y \in T_t : y = \text{itao}(\text{parentNode}(x)) \wedge x.\text{src} \neq \text{NULL} \wedge (\text{parentNode}(x) = \text{field} \vee \text{parentNode}(x) = \text{record} \vee \text{parentNode}(x) = \text{initial})$
--

Appendix C

→ AddAttribute (y.id, “grammar”, x.src)

G39 – enumerate(vocalPrompt)

$\forall x,y \in T_V : x = \text{enumerate}, y = \text{value} \wedge x.\text{textnode} \neq \text{NULL} \wedge y \notin \text{childnodes}(x)$
→ Addnode (“vocalPrompt”, idenum) where idenum = NodeAmount(Tt)
 $\forall x,y \in T_V : x = \text{enumerate}, y = \text{value} \wedge x.\text{textnode} \neq \text{NULL} \wedge y \notin \text{childnodes}(x)$
→ AddAttribute (idenum, “id”, idenum)
 $\forall x,y \in T_V : x = \text{enumerate}, y = \text{value} \wedge x.\text{textnode} \neq \text{NULL} \wedge y \notin \text{childnodes}(x)$
→ AddAttribute (idenum, “name”, idenum)
 $\forall x,y \in T_V : x = \text{enumerate}, y = \text{value} \wedge x.\text{textnode} \neq \text{NULL} \wedge y \notin \text{childnodes}(x)$
→ AddAttribute (idenum, “defaultContent”, x.textnode)

G40 – enumerate + options(vocalPrompt)

$\forall x,y \in T_V : x = \text{enumerate} \wedge y = \text{option} \wedge y \in \text{sibling}(\text{parentNode}(x))$
→ Addnode (“vocalPrompt”, idenum) where idenum = NodeAmount(Tt)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge y = \text{option} \wedge y \in \text{sibling}(\text{parentNode}(x))$
→ AddAttribute (idenum, “id”, idenum)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge y = \text{option} \wedge y \in \text{sibling}(\text{parentNode}(x))$
→ AddAttribute (idenum, “name”, idenum)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge y = \text{option} \wedge y \in \text{sibling}(\text{parentNode}(x)) \wedge y.\text{dtmf} \neq \text{NULL}$
→ AddAttribute (idenum, “keyboardShortcut”, x.dtmf)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge y = \text{option} \wedge y \in \text{sibling}(\text{parentNode}(x)) \wedge y.\text{textnode} \neq \text{NULL} \wedge y.\text{dtmf} \neq \text{NULL}$ → AddAttribute (idenum, “defaultContent”, x.dtmf + x.textnode)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge y = \text{option} \wedge y \in \text{sibling}(\text{parentNode}(x)) \wedge y.\text{textnode} \neq \text{NULL} \wedge y.\text{dtmf} = \text{NULL}$ → AddAttribute (idenum, “defaultContent”, Dist(y,w) + x.textnode)

G41 – enumerate + choices(vocalPrompt)

$\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y))$
→ Addnode (“vocalMenuItem”, idenum) where idenum = NodeAmount(Tt)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y)$
→ AddAttribute (idenum, “id”, idenum)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y)$
→ AddAttribute (idenum, “name”, idenum)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y) \wedge y.\text{dtmf} \neq \text{NULL}$ → AddAttribute (idenum, “keyboardShortcut”, y.dtmf)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y) \wedge y.\text{next} \neq \text{NULL}$ → AddAudiTr(y, y.next)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y) \wedge y.\text{expr} \neq \text{NULL}$ → AddAttribute(idenum, “currentValue”, y.expr)
 $\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{“menu”}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y) \wedge y.\text{textnode} \neq \text{NULL} \wedge y.\text{dtmf} \neq \text{NULL}$ → AddAttribute (idenum, “defaultContent”, y.dtmf + y.textnode)

Appendix C

$$\forall x,y \in T_V : x = \text{enumerate} \wedge z = \text{NearestInPath}(x, \text{"menu"}) \wedge y = \text{choice} \wedge \text{isInPath}(z,y) \wedge y.\text{textnode} \neq \text{NULL} \wedge y.\text{dtmf} = \text{NULL} \rightarrow \exists w = \text{choice} : (w \in \text{siblingsBefore}(y) \vee w=y) \wedge \text{leftsibling}(w) = \text{NULL}, \text{AddAttribute}(\text{idenum}, \text{"defaultContent"}, \text{Dist}(y,w) + y.\text{textnode})$$

G42 – break(vocalPrompt)

$$\forall x,y \in T_V : x = \text{break} \wedge \text{parentNode}(x) = \text{prompt} \rightarrow \text{Addnode}(\text{"Pause"}, \text{idpause}) \text{ where } \text{idpause} = \text{NodeAmount}(Tt)$$
$$\forall x,y \in T_V : x = \text{break} \wedge \text{parentNode}(x) = \text{prompt} \wedge x.\text{size} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idpause}, \text{"size"}, x.\text{size})$$
$$\forall x,y \in T_V : x = \text{break} \wedge \text{parentNode}(x) = \text{prompt} \wedge x.\text{strength} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idpause}, \text{"strength"}, x.\text{strength})$$
$$\forall x,y \in T_V : x = \text{break} \wedge \text{parentNode}(x) = \text{prompt} \wedge x.\text{time} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idpause}, \text{"time"}, x.\text{time})$$

G43 – goto (vocalNavigation)

$$\forall x \in T_V : x = \text{goto} \rightarrow \text{Addnode}(\text{"vocalNavigation"}, \text{idgoto}) \text{ where } \text{idgoto} = \text{NodeAmount}(Tt)$$
$$\forall x \in T_V : x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"id"}, \text{idgoto})$$
$$\forall x \in T_V : x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"name"}, \text{idgoto})$$
$$\forall x \in T_V : x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"NavigationType"}, \text{"goto"})$$
$$\forall x \in T_V : x = \text{goto} \rightarrow \text{AddAttribute}(\text{idgoto}, \text{"isBridgeable"}, \text{"false"})$$
$$\forall x \in T_V : x = \text{goto} \wedge x.\text{nextitem} \neq \text{NULL} \rightarrow \text{AddAudiTr}(\text{idgoto}, x.\text{nextitem})$$

This group of rules is specific to VoiceXML trees. It derives a **goto** element into a **vocalNavigation** node. **Name** and **id** are automatically associated with the element, so as the **navigationType** and **isBridgeable** attributes which are set to **goto** and **false** respectively. Finally, the **nextItem** attribute is used to add an **auditoryTransition** to the CUI model.

G44– prosody (vocalPrompt)

$$\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, x).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, x).\text{count} = 1) \rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprosody}) \text{ where } \text{idprosody} = \text{NodeAmount}(Tt)$$
$$\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, x).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, x).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprosody}, \text{"id"}, \text{idprosody})$$
$$\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, x).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, x).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprosody}, \text{"name"}, \text{idprosody})$$
$$\forall x \in T_V : x = \text{prosody} \wedge \text{NearestInPath}(\text{prompt}, x).\text{bargain} = \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, x).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, x).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprosody}, \text{"isInterruptible"}, \text{"false"})$$
$$\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, x).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, x).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprosody}, \text{"defaultContent"}, x.\text{textnode})$$
$$\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, x).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, x).\text{count} = 1) \wedge x.\text{contour} \neq \text{NULL}$$

Appendix C

$\rightarrow \text{AddAttribute}(\text{idprosody}, \text{"pitch"}, \text{x.contour})$
 $\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \wedge x.\text{range} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idprosody}, \text{"pitch"}, \text{x.range})$
 $\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \wedge x.\text{pitch} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idprosody}, \text{"pitch"}, \text{x.pitch})$
 $\forall x \in T_V : x = \text{prosody} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \wedge x.\text{volume} \neq \text{NULL}$
 $\rightarrow \text{AddAttribute}(\text{idprosody}, \text{"volume"}, \text{x.volume})$

G45 – sentence, s, p (vocalPrompt)

$\forall x \in T_V : (x = \text{sentence} \vee x = \text{p} \vee x = \text{s}) \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1)$
 $\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idprompt}) \text{ where } \text{idprompt} = \text{NodeAmount}(\text{Tt})$
 $\forall x \in T_V : (x = \text{s} \vee x = \text{p} \vee x = \text{sentence}) \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idprompt}, \text{"id"}, \text{idprompt})$
 $\forall x \in T_V : (x = \text{s} \vee x = \text{p} \vee x = \text{sentence}) \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idaudio}, \text{"name"}, \text{idprompt})$
 $\forall x \in T_V : (x = \text{s} \vee x = \text{p} \vee x = \text{sentence}) \wedge \text{NearestInPath}(\text{prompt}, \text{x}).\text{bargain} = \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1)$
 $\rightarrow \text{AddAttribute}(\text{idprompt}, \text{"isInterruptible"}, \text{"false"})$
 $\forall x \in T_V : (x = \text{s} \vee x = \text{p} \vee x = \text{sentence}) \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1)$
 $\rightarrow \text{AddAttribute}(\text{idprompt}, \text{"defaultContent"}, \text{x.textnode})$

G46 – emphasis (vocalPrompt)

$\forall x \in T_V : x = \text{emphasis} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1)$
 $\rightarrow \text{Addnode}(\text{"vocalPrompt"}, \text{idemph}) \text{ where } \text{idemph} = \text{NodeAmount}(\text{Tt})$
 $\forall x \in T_V : x = \text{emphasis} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idemph}, \text{"id"}, \text{idemph})$
 $\forall x \in T_V : x = \text{emphasis} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idemph}, \text{"name"}, \text{idemph})$
 $\forall x \in T_V : x = \text{emphasis} \wedge \text{NearestInPath}(\text{prompt}, \text{x}).\text{bargain} = \text{false} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1)$
 $\rightarrow \text{AddAttribute}(\text{idemph}, \text{"isInterruptible"}, \text{"false"})$
 $\forall x \in T_V : x = \text{emphasis} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \rightarrow \text{AddAttribute}(\text{idemph}, \text{"defaultContent"}, \text{x.textnode})$
 $\forall x \in T_V : x = \text{emphasis} \wedge x.\text{textnode} \neq \text{NULL} \wedge (\text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = \text{NULL} \vee \text{NearestInPath}(\text{prompt}, \text{x}).\text{count} = 1) \wedge x.\text{level} \rightarrow \text{AddAttribute}(\text{idemph}, \text{"intonation"}, \text{x.level})$

Multiple-trees rules

These transformations use different trees (i.e. html, wml or vxml files) as input to be processed. These multiple-trees transformation occur when a **frameset**, **iframe** or **subdialog** node is detected. For the transformation of (i)frames, the tree containing the **frameset** (or **iframe**) node is named T_H^0 and the other trees (content of the **frames** or **iframe**) are named T_H^i . In parallel, a tree containing only a set of **boxes** (corresponding to the layout of the **frames**) is created in T_T^0 and the targeted **frames** are recorded in several other trees (T_T^i).

For the transformation of **subdialogs** the file containing the **subdialog** is named T_T^0 while the targeted files are put in other trees (T_T^i). The derivation process is applied normally, but when the different files to be displayed in **frames** (or the targeted **subdialog/iframe**) are processed, a tree merging operation occurs (all the trees are merged into one tree, and thus one specification). The **frameset** node implies the creation of a **window**, which is subdivided into several **boxes** (G47). A special attribute `-targetFile-` is added to these **boxes** and is compared to the `filename` attribute of **windows** in the target tree (G48). If the two names are identical, the **window** is appended to the **box**. The same occurs with **iframes**, with the difference that the “framed” tree is appended to a **window** instead of **box**.

G47 – Frameset (window)

$\forall x \in T_H^0 : x = \text{frameset} \rightarrow \text{Addnode}(\text{“window”}, \text{idwin}) \text{ to } T_t^0 \text{ where } \text{idwin} = \text{NodeAmount}(T_t^0)$
 $\forall x \in T_H^0 : x = \text{frameset} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“id”}, \text{idwin})$
 $\forall x \in T_H^0 : x = \text{frameset} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“name”}, \text{idwin})$
 $\forall x \in T_H^0 : x = \text{frameset} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“isEnabled”}, \text{“true”})$
 $\forall x \in T_H^0 : x = \text{frameset} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“isVisible”}, \text{“true”})$

G48 – Frame(box)

$\forall x \in T_H^0 : x = \text{frame} \wedge y = \text{NearestInPath}(x, \text{frameset}) \wedge y.\text{cols} \neq \text{NULL} \rightarrow \text{ConstrBox}(\text{“box”}, \text{“horizontal”}, \text{idbox}) \text{ to } T_t^0 \text{ where } \text{idbox} = \text{NodeAmount}(T_t^0)$
 $\forall x \in T_H^0 : x = \text{frame} \wedge y = \text{NearestInPath}(x, \text{frameset}) \wedge y.\text{rows} \neq \text{NULL} \rightarrow \text{ConstrBox}(\text{“box”}, \text{“vertical”}, \text{idbox}) \text{ to } T_t^0 \text{ where } \text{idbox} = \text{NodeAmount}(T_t^0)$
 $\forall x \in T_H^0 : x = \text{frame} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“targetfile”}, x.\text{src})$
 $\forall x \in T_H^0 : x = \text{frame} \wedge x.\text{scrolling} \neq \text{NULL} \wedge x.\text{scrolling} \neq \text{“no”} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“isScrollable”}, \text{“true”})$
 $\forall x \in T_H^0 : x = \text{frame} \wedge x.\text{scrolling} \neq \text{NULL} \wedge x.\text{scrolling} = \text{“no”} \rightarrow \text{AddAttribute}(\text{idwin}, \text{“isScrollable”}, \text{“false”})$
 $\forall x \in T_H^0 : x = \text{frame} \wedge x.\text{frameborder} \neq \text{NULL} \wedge x.\text{frameborder} \neq 0 \rightarrow \text{AddAttribute}(\text{idwin}, \text{“borderWidth”}, x.\text{frameborder})$

Appendix C

G48b – iFrame(window)

$\forall x \in T_H^0: x = \text{iframe} \rightarrow \text{AddNode}(\text{"window"}, \text{idwin})$ where $\text{idwin} = \text{NodeAmount}(T_t)$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{name} \neq \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{"name"}, x.\text{name}, \text{idwin}) \wedge \text{AddAttribute}(\text{"id"}, x.\text{name}, \text{idwin})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{name} \neq \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{"name"}, \text{idwin}, \text{idwin}) \wedge \text{AddAttribute}(\text{"id"}, \text{idwin}, \text{idwin})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{src} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"targetfile"}, x.\text{src})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{width} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"targetfile"}, x.\text{width})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{height} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idwin}, \text{"targetfile"}, x.\text{height})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{marginwidth} \neq \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{idwin}, \text{"windowTopMargin"}, x.\text{marginwidth})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{marginheight} \neq \text{NULL}$ $\rightarrow \text{AddAttribute}(\text{idwin}, \text{"windowLeftMargin"}, x.\text{marginheight})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{scrolling} \neq \text{NULL} \wedge x.\text{scrolling} \neq \text{"no"}$ $\rightarrow \text{AddAttribute}(\text{idwin}, \text{"isScrollable"}, \text{"true"})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{scrolling} \neq \text{NULL} \wedge x.\text{scrolling} = \text{"no"}$ $\rightarrow \text{AddAttribute}(\text{idwin}, \text{"isScrollable"}, \text{"false"})$ $\forall x \in T_H^0: x = \text{iframe} \wedge x.\text{frameborder} \neq \text{NULL} \wedge x.\text{frameborder} \neq 0$ $\rightarrow \text{AddAttribute}(\text{idwin}, \text{"borderWidth"}, x.\text{frameborder})$
--

G49 - Tree merging

$\forall x \in \text{to } T_t^i, y \in T_t^0: x = \text{window} \wedge (y = \text{box} \vee y = \text{window}) \wedge x.\text{filename} = y.\text{targetfile}$ \rightarrow CloneNode(x.id, idnew, T_t^0) where $\text{idnew} = \text{NodeAmount}(T_t^0) \wedge$ RemoveNode(x, x.id) \wedge RemoveArc(ParentNode(x).id, x.id) \wedge z=root(T_t^i) \wedge Remove Node(z,z.id) \wedge AddArc(y.id, idnew)
$\forall x \in \text{to } T_t^i, y \in T_t^0: x = \text{vocalGroup} \wedge y = \text{VocalGroup} \wedge x.\text{filename} = y.\text{insertFile}$ \rightarrow CloneNode(x.id, idnew, T_t^0) where $\text{idnew} = \text{NodeAmount}(T_t^0) \wedge$ RemoveNode(x, x.id) \wedge RemoveArc(ParentNode(x).id, x.id) \wedge z=root(T_t^i) \wedge Remove Node(z,z.id) \wedge AddArc(y.id, idnew)

This last operation (upper part) copies the different **windows** (corresponding to different HTML files) into the tree T_t^0 , and empties the trees T_t^i . The output of this last step is one tree containing several **windows** and a set of empty trees. For VoiceXML, this tree merging operation copies an entire tree (T_t^i) into the tree possessing the **insertFile** attribute (T_t^0). The **subdialog** node has a simpler processing: when it has been detected, it is transformed into a **vocalGroup** (see group rules G31) and the target file is appended to this **vocalGroup**.

Intra-tree rules

These mappings occur after all the inter-graph and multiple-trees operation. These rules represent the correction of the produced tree (model), transformation of some elements (e.g. the change of `imageComponents` into `textComponents`, or the transformation of set of `radioButtons` in a `ComboBox`, ...) or the recovery of the original layout of the UI by reorganizing the `boxes`.

G50-Cleaning of the target tree

$\forall x \in T_t: x = \text{Box} \wedge x.\text{isLeaf} \rightarrow \text{RemoveNode}(\text{"box"}, x.\text{id}) \wedge \text{RemoveArc}(\text{parentNode}(x), x)$ $\forall x \in T_t: x = \text{window} \vee x = \text{vocalGroup} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"filename"})$ $\forall x \in T_t: x.\text{path} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"path"})$ $\forall x \in T_t: x = \text{box} \wedge x.\text{targetfile} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"targetfile"})$ $\forall x \in T_t: x = \text{vocalGroup} \wedge x.\text{insertfile} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"insertFile"})$ $\forall x \in T_t: x = \text{vocalGroup} \wedge x.\text{isLeaf} \rightarrow \text{RemoveNode}(\text{"vocalGroup"}, x.\text{id}) \wedge \text{RemoveArc}(\text{parentNode}(x), x)$
--

G51-Reverse Engineering options

Following the options selected by the user/tool, several transformations are applied after the construction of the target tree. Some examples are given here to illustrate these transformations.

► Transform images into text

$\forall x \in T_t: x = \text{"ImageComponent"} \rightarrow \text{ModifyNode}(x, \text{"textComponent"}, x.\text{id}, x.\text{id})$ $\forall x \in T_t: x = \text{"ImageComponent"} \wedge y = \text{intro}(x) \wedge y.\text{alt} \neq \text{NULL} \rightarrow \text{ModifyAttribute}(x.\text{id}, \text{"defaultContent"}, \text{"defaultContent"}, y.\text{alt})$ <p>Faire un fo attribut nettoyé ensuite</p> $\forall x \in T_t: x = \text{"ImageComponent"} \wedge y = \text{intro}(x) \wedge y.\text{alt} = \text{NULL} \rightarrow \text{ModifyAttribute}(x.\text{id}, \text{"defaultContent"}, \text{"defaultContent"}, \text{"Image"})$ $\forall x \in T_t: x = \text{img} \wedge x.\text{imageHeight} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"imageHeight"},)$ $\forall x \in T_t: x = \text{img} \wedge x.\text{imageWidth} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"imageWidth"})$ $\forall x \in T_t: x = \text{img} \wedge x.\text{borderWidth} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"borderWidth"})$ $\forall x \in T_t: x = \text{img} \wedge x.\text{imageHorizSpace} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"imageHorizSpace"})$ $\forall x \in T_t: x = \text{img} \wedge x.\text{imageVertSpace} \neq \text{NULL} \rightarrow \text{RemoveAttribute}(x.\text{id}, \text{"imageVertSpace"})$
--

► Folding of consecutive labels (textcomponents)

$\forall x \in T_t: x = \text{"textComponent"}, y = \text{righthSibling}(x) \wedge y = \text{"textComponent"} \wedge x.\text{defaultContent} \neq \text{NULL} \wedge y.\text{defaultContent} \neq \text{NULL} \wedge x.\text{hyperLinkTarget} = \text{NULL} \wedge y.\text{hyperLinkTarget} = \text{NULL} \rightarrow \text{ModifyAttribute}(x.\text{id}, \text{"defaultContent"}, \text{"defaultContent"},$
--

Appendix C

$x.defaultContent + y.defaultContent) \wedge RemoveNode("textComponent", y.id) \wedge RemoveArc(parentNode(y), y.id)$

► Transform Radio Buttons into a drop down listbox

$\forall x_0 \dots x_n \in T_t: x_0 \dots x_n = "radio" \wedge x_0.groupName = \dots = x_n.groupname$
 $\rightarrow AddNode("comboBox", comboid) \text{ where } comboid = NodeAmount(T_t)$
 $\wedge ModifyArc(ParentNode(x_0), x_0, comboid)$
 $\wedge AddAttribute(comboid, "id", comboid)$
 $\wedge AddAttribute(comboid, "name", comboid)$
 $\wedge AddAttribute(comboid, "isEnabled", "true")$
 $\wedge AddAttribute(comboid, "isEditable", "false")$
 $\wedge AddAttribute(comboid, "isVisible", "true")$
 $\wedge AddAttribute(comboid, "isDropDown", "true")$
 $\wedge z_0 = CloneNode(x_0.id, idnew) \dots z_n = CloneNode(x_n.id, idnew+n) \text{ where } idnew = NodeAmount(T_t^0)$
 $\wedge RemoveNode("radio", x_0.id) \dots RemoveNode("radio", x_n.id)$
 $\wedge RemoveArc(parentNode(x_0).id, x_0.id) \wedge RemoveArc(parentNode(x_n).id, x_n.id)$
 $\wedge ModifyNode("radio", "item", z_0.id, z_0.id) \dots ModifyNode("radio", "item", z_n.id, z_n.id)$
 $\wedge RemoveAttribute(z_0, "groupName") \dots RemoveAttribute(z_n, "groupName")$
 $\wedge RemoveAttribute(z_0, "defaultState") \dots RemoveAttribute(z_n, "defaultState")$
 $\wedge AddArc(comboid, z_0.id) \dots AddArc(comboid, z_n.id)$

► Remove Images with a surface bigger than 800 pixels

$\forall y \in T_t: y = "ImageComponent" \wedge ((y.imageHeight) \times (y.imageWidth) > 800) \rightarrow RemoveNode(y, y.id) \wedge RemoveArc(parentNode(y), y)$

G52- Layout recovery

The layout recovery rules are applied in sequence. The process is divided into five steps and advances one step further when the rule(s) of one step cannot be applied (anymore) to the target tree. This process uses the **br** tags - recorded during the inter-tree transformations - to divide the user interface into **boxes**. The principle of the algorithm is to add a vertical **box** containing several horizontal **boxes** when a **br** tag is found in the target tree. Adding these nested boxes allows representing the different lines of the UI. A **br** node in HTML adds a blank line in the UI and therefore, elements before and after the **br** are put on different lines, which is represented in UsiXML by putting them into two different horizontal **boxes**, horizontal **boxes** that are put themselves into a vertical **box**.

Example:

<code><text/>
</code>		<code><vertical box></code>
<code><input/><text/>
</code>	IS DERIVED AS	<code><horizontal box><text/></hb></code>
<code><input/><image/></code>		<code><horizontal box><input/><text/></hb></code>
		<code><horizontal box><input/><image/></hb></code>
		<code></vb></code>

Appendix C

STEP 1

In this step, every **br** node that has siblings on its right (i.e. after the the **br** node) is renamed into a **bix** node and the other **br** nodes (without right siblings) are removed from the tree.

$$\forall x,y \in T_t: y=br \wedge x=parentNode(y) \wedge righthSibling(y) \neq NULL \rightarrow RemoveNode(y, y.id) \wedge z=ConstrBox("bix", "horizontal", idbix) \text{ where } idbix = \sum node \in T_t \wedge ModifyArc(x.id, y.id, idbix)$$
$$\forall x,y \in T_t: y=br \wedge x=parentNode(y) \wedge righthSibling(y) = NULL \rightarrow RemoveNode(y, y.id) \wedge RemoveArc(x,y).$$

STEP 2

In this step, every sibling on the right of a **bix** node is appended as child of the **bix** (except for table, cell, box or another **bix**).

$$\forall x \in T_t: x=bix \wedge ((righthSibling(x) \neq table \vee righthSibling(x) \neq bix \vee righthSibling(x) \neq cell \vee righthSibling(x) \neq box) \wedge righthSibling(x) \neq NULL) \rightarrow CloneNode(rightSibling(x).id, idnew) \wedge RemoveNode(rightSibling(x), rightSibling(x).id) \wedge RemoveArc(ParentNode(rightSibling(x)), rightSibling(x)) \wedge AddArc(x.id, idnew) \text{ where } idnew = \sum node \in T_t$$

STEP 3

A vertical **box** is created as a first child for each parent node of **bix** nodes. All the children of this node (parent of the **bix**) are moved as children of the new vertical **box**. This vertical **box** (renamed **bux** to differentiate it from a “normal” **box**) will contain every siblings of the **bix** node including the **bix** node.

$$\forall x \in T_t: x=bix \wedge parentNode(x) \neq "bux" \rightarrow ConstrBox("bux", "vertical", idbox) \text{ where } idbox = \sum node \in T_t \wedge (\forall z_i \in childNodes(parentNode(x)), CloneNode(z_i.id, idnew) \wedge RemoveNode(z_i, z_i.id) \wedge RemoveArc(ParentNode(x).id, z_i.id) \wedge AddArc(idbox, idnew) \text{ where } idnew = \sum node \in T_t)$$

STEP 4

The vertical **boxes** created in the last step are divided into horizontal **boxes** to represent the different lines composing the UI. The children of **buxes** are scanned and for this step, 3 cases can occur:

1. This is the first child and it is not a **bix**. An horizontal **box** is created and this (first) element and is appended as its child.
2. This is a **bix** element. Nothing happens as the elements following the **bix** are already put on a lower level of hierarchy.
3. This node is not in the 2 first categories, and the element is appended to the previous horizontal **box** (the previous arc is removed and replace by a new arc representing this change in the hierarchy).

Appendix C

$$\begin{aligned} &\forall x \in T_t: \text{parentNode}(x) = \text{"bux"} \wedge (\text{leftSibling}(x) = \text{NULL}) \wedge x \neq \text{bax} \wedge x \neq \text{bix} \\ &\rightarrow \text{ConstrBox}(\text{"bax"}, \text{"horizontal"}, \text{idbox}) \wedge \text{CloneNode}(x.\text{id}, \text{idnew}) \wedge \text{RemoveNode}(x, x.\text{id}) \wedge \\ &\text{RemoveArc}(\text{ParentNode}(x).\text{id}, x.\text{id}) \wedge \text{AddArc}(\text{parentNode}(x), \text{idbox}) \wedge \text{AddArc}(\text{idbox}, \text{idnew}) \\ &\text{where } \text{idbox}, \text{idnew} = \sum \text{node} \in T_t \\ &\forall x \in T_t: \text{parentNode}(x) = \text{"bux"} \wedge (\text{leftSibling}(x) \neq \text{NULL} \wedge \text{leftSibling}(x) \neq \text{"bix"}) \wedge x \neq \text{bax} \wedge \\ &x \neq \text{bix} \rightarrow y = \text{leftSibling}(x), \text{RemoveArc}(\text{parentNode}(x).\text{id}, x.\text{id}) \wedge \text{AddArc}(y.\text{id}, x.\text{id}) \end{aligned}$$

STEP 5

Cleaning of the target tree. The buxes (vertical **boxes**), baxes and bixes (horizontal **boxes**) are renamed into boxes.

$$\begin{aligned} &\forall x \in T_t: x = \text{"bix"} \rightarrow \text{ModifyNode}(x, x.\text{id}, \text{"box"}, x.\text{id}) \\ &\forall x \in T_t: x = \text{"bax"} \rightarrow \text{ModifyNode}(x, x.\text{id}, \text{"box"}, x.\text{id}) \\ &\forall x \in T_t: x = \text{"bux"} \rightarrow \text{ModifyNode}(x, x.\text{id}, \text{"box"}, x.\text{id}) \\ &\forall x \in T_t: x = \text{box} \wedge x.\text{isLeaf} \rightarrow \text{RemoveNode}(\text{"box"}, \text{idbox}) \wedge \text{RemoveArc}(\text{parentNode}(x), x) \end{aligned}$$

G53- Mixed Initiative

This set of rules transforms a `vocalGroupInit` element belonging to the target tree into a "normal" `vocalGroup`, and puts the rest of the `form` into another `vocalGroup`. `VocalGroupInit` elements represent `initial` elements belonging to mixed-initiative `forms` from the `voiceXML` file. These `initial` elements are processed first, before the rest of the `form`. The "-init" is added to the element's name to be processed a second time (in G53) after the entire tree has been parsed. Step 1 creates a new `vocalGroup` and put all the right-siblings of the `vocalGroupInit` in this new `vocalGroup`, and step 2 modifies the `vocalGroupInit` name into `vocalGroup`.

STEP 1

$$\begin{aligned} &\forall x, y \in T_t: x = \text{vocalGroupInit} \wedge y = \text{rightSibling}(x) \wedge y \neq \text{NULL} \rightarrow \text{Addnode} \\ &(\text{"vocalGroup"}, \text{idvgroup}) \text{ where } \text{idvgroup} = \text{NodeAmount}(T_t^0) \\ &\quad \rightarrow \text{AddAttribute}(\text{idvgroup}, \text{"id"}, \text{idvgroup}) \\ &\quad \rightarrow \text{AddAttribute}(\text{idvgroup}, \text{"name"}, \text{idvgroup}) \\ &\quad \rightarrow \text{ModifyArc}(\text{ParentNode}(y).\text{id}, y.\text{id}, \text{idvgroup}) \\ &\quad \rightarrow \text{AddArc}(\text{idvgroup}, y.\text{id}) \\ &\forall x, y \in T_t: x = \text{vocalGroupInit} \wedge y = \text{rightSibling}(x) \wedge y \neq \text{NULL} \wedge y.\text{id} \neq \text{idvgroup} \\ &\quad \rightarrow \text{RemoveArc}(\text{ParentNode}(y).\text{id}, y.\text{id}) \wedge \text{AddArc}(\text{idvgroup}, y.\text{id}) \end{aligned}$$

STEP 2

$$\begin{aligned} &\forall x, y \in T_t: x = \text{vocalGroupInit} \wedge y = \text{rightSibling}(x) \wedge y = \text{vocalGroup} \wedge \text{rightSibling}(y) = \text{NULL} \\ &\rightarrow \text{Modifynode}(x.\text{id}, \text{"vocalGroupInit"}, \text{"vocalGroup"}, \text{idvgroup}) \\ &\text{where } \text{idvgroup} = \text{NodeAmount}(T_t^0) \end{aligned}$$

Appendix C

G54 – Pause Transformation

This set of rules transforms a pause element into an AudioAdjacency relation between the two elements surrounding the pause. After the transformation has been processed, the pause element is removed from the target tree (step 2).

STEP 1

$\forall x,y,z \in T_V : x = \text{pause}, y=\text{leftSibling}(x), z=\text{rightSibling}(x) \rightarrow \text{Addnode}(\text{"AudioAdjacency"}, \text{idaudadj}) \text{ where } \text{idaudadj} = \text{NodeAmount}(T_t^0)$ $\forall x,y,z \in T_V : x = \text{pause}, y=\text{leftSibling}(x), z=\text{rightSibling}(x) \rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"id"}, \text{idaudadj})$ $\rightarrow \text{AddNode}(\text{"Source"}, \text{idSource}) \text{ where } \text{idSource} = \text{NodeAmount}(T_t)$ $\rightarrow \text{AddAttribute}(\text{idSource}, \text{"source"}, y.\text{id})$ $\rightarrow \text{AddNode}(\text{"Target"}, \text{idTarget}) \text{ where } \text{idTarget} = \text{NodeAmount}(T_t)$ $\rightarrow \text{AddAttribute}(\text{idTarget}, \text{"target"}, z.\text{id})$ $\rightarrow \text{AddArc}(0, \text{"1"}, \text{idaudadj})$ $\rightarrow \text{AddArc}(\text{idaudadj}, \text{idSource})$ $\rightarrow \text{AddArc}(\text{idaudadj}, \text{idTarget})$ $\rightarrow \text{RemoveArc}(\text{parentNode}(x).\text{id}, x.\text{id})$ $\forall x \in T_V : x = \text{pause} \wedge x.\text{size} = \text{"small"} \rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, \text{"2000"})$ $\forall x \in T_V : x = \text{pause} \wedge x.\text{size} = \text{"medium"} \rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, \text{"5000"})$ $\forall x \in T_V : x = \text{pause} \wedge x.\text{size} = \text{"large"} \rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, \text{"10000"})$ $\forall x \in T_V : x = \text{pause} \wedge (x.\text{strength} = \text{"x-weak"} \vee x.\text{strength} = \text{"weak"})$ $\rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, \text{"350"})$ $\forall x \in T_V : x = \text{pause} \wedge x.\text{strength} = \text{"medium"} \rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, \text{"700"})$ $\forall x \in T_V : x = \text{pause} \wedge (x.\text{strength} = \text{"strong"} \vee x.\text{strength} = \text{"x-strong"})$ $\rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, \text{"1000"})$ $\forall x \in T_V : x = \text{pause} \wedge x.\text{time} \neq \text{NULL} \rightarrow \text{AddAttribute}(\text{idaudadj}, \text{"delayTime"}, x.\text{time}/1000)$
--

STEP2

$\forall x \in T_V : x = \text{pause} \wedge \text{parentNode}(x) = \text{NULL} \rightarrow \text{RemoveNode}(x, x.\text{id})$
--

Appendix D

The context model

The WhoAmI (WAI) context recognition system can be exploited to generate some parts of the context model (see figure D-1). This is completed by the use of Javascripts, which send back some information about the connected platform, such as the number of colours or the screensize.

The attributes of the context model expressed in UsiXML and their recognition method are shown in the following tables. The different attributes of the context model are displayed in the left part of the table and their corresponding source of information in the right part.

Remarks:

The environment element can not be detected by WAI, JS nor html code.

The only attribute of the UserStereotype that can be written is the id (=the IP of the user)

All the elements and attributes from the following tables belong to the Platform element

JS means information found thanks to a javascript, WAI means information found thanks to the WhoAmI system

Appendix D

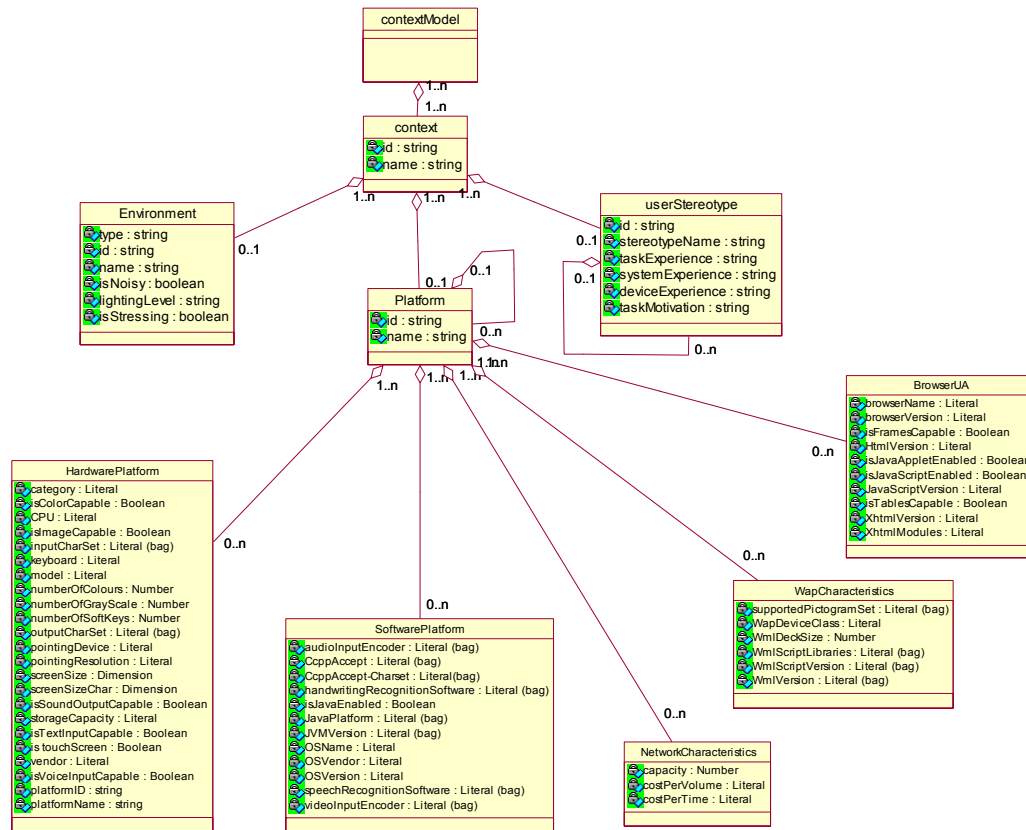


Fig. D-1 The context model

Hardware Platform	
Category	Is deduced from the JS and WAI information
IsColorCapable	True if NumberOfColours>2
NumberOfColours	JS-Math.pow (2,document.form.colordepth.value)
ScreenSize	JS- screen.height x screen.width
PlatformID	JS - document.form.platform.value
PlatformName	JS - document.form.platform.value

Software Platform	
isJavaEnabled	JS - navigator.javaEnabled()
OSName	JS- navigator.appVersion.indexOf / WAI
OSVersion	JS – parse navigator.appVersion / WAI

Appendix D

Platform	
Id	WAI-useradress+userip
Name	WAI-useradress+userip

BrowserUA	
BrowserName	JS-navigator.userAgent.indexOf /WAI
BrowserVersion	JS-substr(navigator.userAgent.indexOf,8) / WAI
JavascriptVersion	JS-complex script/WAI
isJavascriptCapable	JS-complex script/WAI
isFrameCapable	JS-complex script/WAI
isJavaAppletEnabled	JS - navigator.javaEnabled()/WAI

Appendix E

UsiXML compliant tools

Different tools exist for the generation of UI code based on a UsiXML specification (see <http://www.usixml.org> for a complete list). Some of them are presented in this appendix.

Teresa is developed at ISTI-CNR and supports the generation of XHTML, VoiceXML and WML starting from a task model, an abstract or concrete UI model expressed in TeresaXML or a concrete UI specified in UsiXML.

The tool (see figure E-1) follows a task-based approach for the generation of UI for multiple devices [Mor03]. The tool gives the possibility to import UI descriptions at the concrete or abstract UI levels, or to generate a UI starting from a task model. The abstract and concrete UI entered as inputs are expressed in TeresaXML, but the tool also accept UsiXML specifications (which are then translated into TeresaXML). The output of this tool is a FUI for mobile phones, desktop computers or a vocal UI (in VoiceXML).

Appendix E

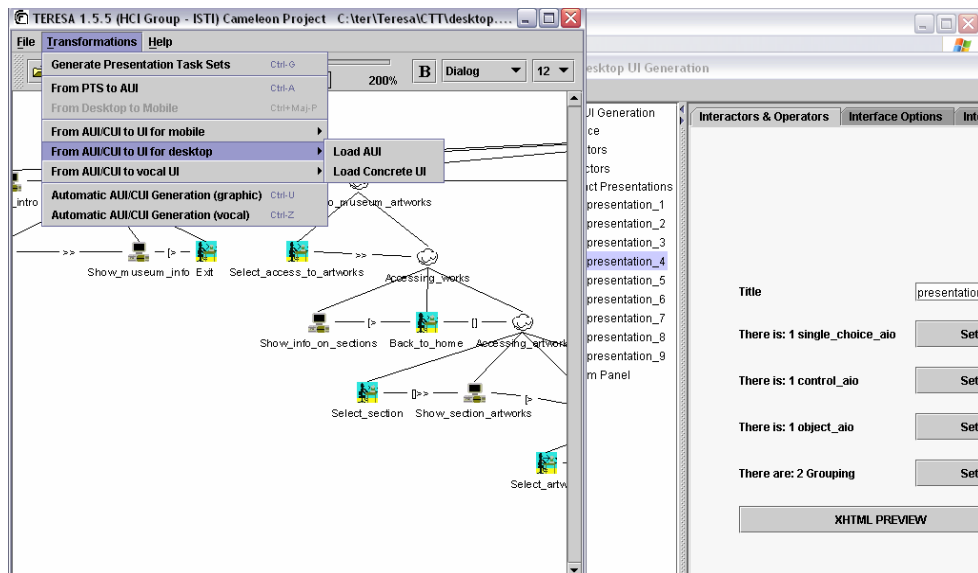


Figure E-1 The generation of Uis with Teresa

The process of generation is decomposed into four parts:

- 1) The creation of a unique task model. For each task, the designer specifies the interaction objects needed to accomplish the task, and the platforms on which the task is available.
- 2) A separate task model is then automatically generated for each platform.
- 3) Based on this task model, an abstract UI is then created for each platform, by identifying the enabled task sets [Pat02] (tasks that are enabled at the same time) and the interactors needed to fulfil the tasks.
- 4) The final UI code is generated. The user has the freedom to modify the heuristics and models at each step of the process to fine tune the UI.

Another tool, **Grafixml** (<http://www.UsiXML.org/index.php?page=grafixml.xml>), is currently developed at Louvain. With this tool, the designer can draw in direct manipulation any graphical UI by directly placing CIOs and editing their properties in a property sheet.

This tool allows users to draw a concrete UI and then automatically generate UsiXML code from the graphical representation (figure E-2) or to produce an UI in XHTML, XUL or Java from a UsiXML specification.

A third tool, **TransformiXML**, applies graph transformations contained in graph grammars to perform transformations of UsiXML-compliant UIs to produce a new UI specification. Such transformation can occur between any level (task and

Appendix E

domain, abstract user interface, concrete user interface) to support forward engineering, reverse engineering, middle-out approach, adaptation, and the wide spreading approach. The tool allows managing a development library (a library containing a catalog of transformation rules)

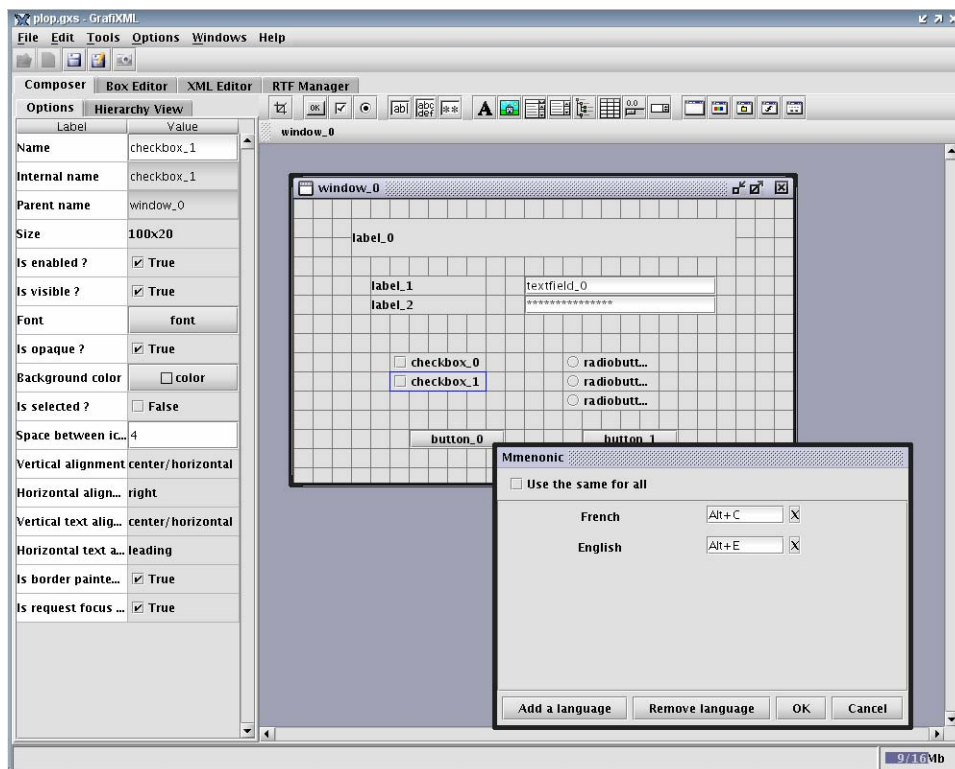


Fig. E-2 GrafXML

The fourth tool is **FlashiXML** (see figure E-3). FlashiXML is a rendering engine of UsiXML-compliant UIs in a vectorial mode that is SVG-compatible. Any UsiXML-compliant UI can be opened and rendered in this interpreter so as to create the truly working UIs with presentation and dialog. In this environment, the UI can be resized at any time to address some constraints imposed by the computing platforms and to support some properties of Graceful Degradation of UIs, a sub-property of the Plasticity property.

In this way, any UsiXML-compliant UI can be rendered on any computing platform equipped with a SVG or Flash plug-in. FlashiXML is developed by Youri Vanden Berghe in his Master thesis in Computer Science at UCL.

Appendix E

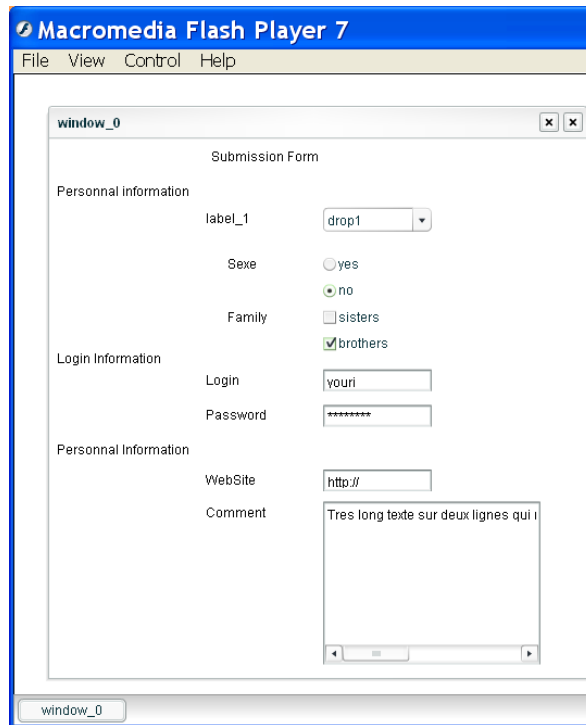


Fig E-3 FlashiXML

The last tool presented in annex E, **IdealXML** (figure E-4) [Mont04], allows the designer to draw the domain model, the abstract UI model, the task model and the mappings between these three models graphically. The tool is then able to generate the UsiXML specification of these descriptions.

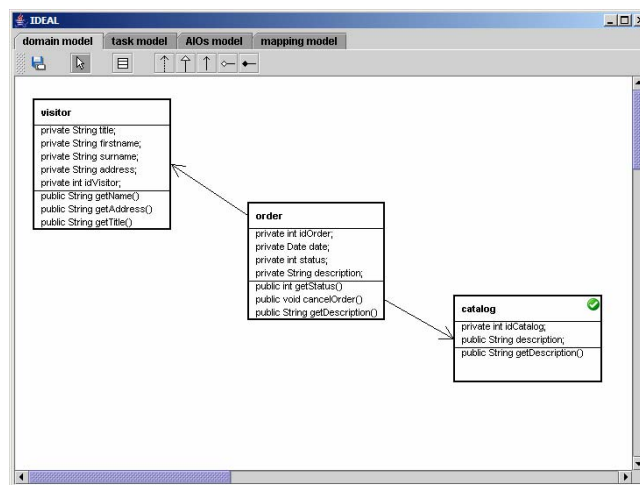


Fig. E-4 IdealXML Domain Modeller

Appendix F

Validation files

This annex contains URLs of the web pages reverse engineered during the experiments related to validation (chapter 9).

1. <http://www.isys.ucl.ac.be/>
2. <http://www.google.be/>
3. <http://www.europa.eu.int>
4. <http://www.microsoft.com/>
5. http://www.euronews.net/create_html.php?page=home
6. <http://www.rcost.unisannio.it/wcre2006/calls/cftp.htm>
7. <http://www.univ-valenciennes.fr>
8. <http://www.isys.ucl.ac.be/bchi/research/reversi/RevXMLUI.php>
9. <http://www.chi2006.org/>
10. <http://www.cnn.com>
11. <http://www.icp.ucl.ac.be/ICP.html>
12. http://www.lesoir.be/rubriques/la_une/page_5682.shtml
13. <http://www.thalys.com/>
14. <http://www.brusselsairport.be/>
15. <http://www.lodging.com/>
16. <http://www.fhr.fr/#debut>
17. http://www.mediamarkt.be/MM_new/index_fr.php
18. <http://www.ing.unlpam.edu.ar/laweb05/>
19. <http://www.infospace.com/>
20. <http://today.reuters.fr/news/default.aspx>

Appendix G

Retargeting rules

This appendix contains retargeting rules designed for the targeted reverse engineering (or retargeting) of HTML UI towards platforms with limited capabilities, such as PDAs or mobile phones.

The rules are classified by source elements with the possible transformations in the target model on the right part of tables. Some rules at the model level are given at the end of this appendix.

Images	Keep the image
	Replace by alternative text
	Convert in simplified image
	Create a link to the image
	Convert in .WBMP
	Suppress images with a height greater than x and a width greater than y pixels
	Suppress images having a file size smaller than x bytes (positioning images)

Text Links	Keep links
	Transform into a button
	Keep only links that are internal to the site

Appendix G

	Suppress download link (having an extension different of .htm, .php, .asp, .html, .cfm, jsp ...)
	Suppress colour attributes for links

Marquees	Keep/suppress all marquees
	Suppress marquees longer than x character
	Transform into a static label

Tables	Keep table
	Transform into a list
	Transform into a paragraph
	Keep only table with a border greater than 0 pixels
	Remove every table

Labels	Keep all properties
	Transform colours in monochrome
	Transformer en system-character sets
	Transform size attributes into WAP size attributes
	Transformer every size in standard size
	Transform text into links giving access to the text when label size is superior to x pixels
	Suppress style attributes such as isBold, isItalic...
	Convert header labels in underlined and bold labels
	Suppress margin of labels

Image Link	Keep/suppress the image-link
	Transform into a button (alternative text becomes the label of the of the button, if it is not present, the targeted page)
	Keep only if the links is internal to the website and not a download link
	Transform into a text-link (alternative text becomes the label of the of the text-link, if it is not present, the targeted page)
	Convert image into the .WBMP
	Convert in image with less colors
	Suppress borders
	Put a condition of the preceding retargeting operations if if the image has a height greater than x and a width greater than y pixels

Appendix G

Image Maps	Keep/suppress all image-maps
	Convert into a list of links
	Convert into a list of links but keep the image next to these links.
	Transform image maps containing one link into an image-link
	Suppress links external to the website and/or a download link
	Convert in a list of links accessible through another link if there are more than x links in the image map.
Horizontal Rules	Create Container
	Transform into a label composed of – (indent)
	Keep/ Suppress horizontal rules
Multi-line edit box	Keep / suppress multi-line edit boxes
	Transform in to single-line edit box
	Add link that give access to the multi line edit box on another page if size is bigger than x rows and y columns.
RadioButtons	Keep / suppress radio button
	Convert into drop down lisbox (combobox)
	Convert into drop down listbox if the number of radio buttons is greater than x elements
CheckBoxes	Keep / Suppress checkboxes
	Convert into drop down lisbox (combobox)
	Convert into drop down listbox if the number of checkbox is greater than x elements
Adresses	Keep as label
	Convert in a label with the following style properties (choose: bold, italic, underlined...)
	Suppress all the addresses
Frames	Convert into separate linked windows
	Keep/Suppress Frames
	Keep only frames containing links

Appendix G

Auditory component	Keep/Suppress these elements
	Add link giving access to the sound

At the model level

These operations are not related to a specific element, but to a group of elements or the entire structure of the produced model.

- Remove all colours
- Remove all styles
- Set maximum number of element per window
- Set a maximum level of hierarchy (for example WML accepts only 9 levels of hierarchy, cHTML 4 levels)
- Split models longer than x elements into two models and add navigation
- Split specifications longer than x elements in different specifications

Appendix H

Examples

H-1 Example of a reverse engineering with Vaquita

This small excerpt of UI comes from the CHI 2001 registration form (Fig.H-1).

Figure H-1 An excerpt of the CHI 2001 registration form

The corresponding XIML code produced by Vaquita is shown below. The code has been edited after the reverse engineering in order to add semantic relations between elements and name of presentation elements have been modified to be more explicit. Some comments are also given in the XIML code.

```
<PRESENTATION_ELEMENT ID="ChooseWayLabel">
<NAME>Choose way of paying Label</NAME> <FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelString">
Please choose one of these ways of paying for your
order. <ATTRIBUTE_STATEMENT>
<RELATION_STATEMENT DEFINITION="Label_describes"
REFERENCE="CreditPayTable">((1))
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelPosition">Top</ATTRIBUTE_STATEMENT>
</RELATION_STATEMENT> </FEATURES> </PRESENTATION_ELEMENT>
```

((1)) This first label describes the table allowing the user to give credit card details. A relation_statement is therefore added in the label specification to record this semantic link. The relation is set in the attributes of the label (in this manner, the source is already defined), and the target is given in the attribute reference of the relation (CrediPayTable). The relation also possesses an attribute, AIOLabelPosition set to top, as the label is positioned on top the table that it describes.

```
<PRESENTATION_ELEMENT ID="PHdirectRadioBt">((2))
<NAME>Pay by Credit card Radio Button</NAME><FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="RadioBtDefaultState">checked
</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="RadioBtNumber">2</ATTRIBUTE_STATEMENT>
```


Appendix H

```
</FEATURES></PRESENTATION_ELEMENT>
```

((2)) Specification of the first radio button (the other one is not visible on this excerpt). This element possesses two attributes derived from the HTML code (see table 5-2), the number of `radioButtons` with the same `name` (2 in this case) and the fact that it is selected by default.

```
<PRESENTATION_ELEMENT ID="PayCreditLabel"> ((3))
<NAME>Pay by credit card Label</NAME><FEATURES>
<RELATION_STATEMENT DEFINITION="Label_describes" REFERENCE="PHDirectRadioBt">
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelPosition">Right</ATTRIBUTE_STATEMENT>
</RELATION_STATEMENT>
<RELATION_STATEMENT DEFINITION="Label_describes" REFERENCE="CreditPayTable">
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelPosition">Left</ATTRIBUTE_STATEMENT>
</RELATION_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelString">
Pay by credit card through the Internet. </ATTRIBUTE_STATEMENT> </FEATURES>
</PRESENTATION_ELEMENT>
```

((3))This label is the text that describes the radio button**((2))**. Therefore, a relation of the type `label describes` is defined between the `label` and the `radio button`. The position of the `label` compared to the `radio button` is `right`. Another relation is added between the `label` and the `table` allowing giving credit card details, but with attribute `AIOLabelPosition` set to `left` in this second case.

```
<PRESENTATION_ELEMENT ID="CreditPayTable"> ((4))
<NAME>Pay by Credit Card Table</NAME> <FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="AIOTableRow">4</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIOTableCol">2</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIOTableBorder">2</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIOTablePosition">left</ATTRIBUTE_STATEMENT>
</FEATURES>
```

((4))This element represents the `table` on the right part of figure 5-6. It possesses four attributes: the two first attributes are computed (counting of the number of columns and rows) and the third is the table border width expressed in pixels. As the table is defined with the `align` attribute set to `left`, this information is kept in the model (in the `AIOTablePosition`), but this is the only type of information about the layout that is reverse engineered in Vaquita.

```
<PRESENTATION_ELEMENT ID="CPTable_Row1"> ((5))
<Name> First row of Pay by Credit Card Table </NAME>
```

((5)) The table structure is kept in the UsiXML specification, as the border of the table is greater than 0 pixel.

```
<PRESENTATION_ELEMENT ID="CPTable_Cell1_1">
<NAME> First Cell </NAME>
<PRESENTATION_ELEMENT ID="CreditTypeLabel">
<NAME>Type of credit card Label</NAME> <FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelString">
Credit Card Type </ATTRIBUTE_STATEMENT></FEATURES>
</PRESENTATION_ELEMENT>
</PRESENTATION_ELEMENT>
<PRESENTATION_ELEMENT ID="CPTable_Cell1_2">
<NAME>Second Cell </NAME>
<PRESENTATION_ELEMENT ID="PayCcTypeDropDownListBox"> ((6))
<NAME>Credit card type Selection List</NAME> <FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">Visa</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">
Mastercard </ATTRIBUTE_STATEMENT>
```

Appendix H

```
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">American  
Express</ATTRIBUTE_STATEMENT>  
<ATTRIBUTE_STATEMENT DEFINITION="AIODLItemSel">Visa</ATTRIBUTE_STATEMENT>  
</FEATURES></PRESENTATION_ELEMENT>
```

((6)) The `select` tag is translated by a `drowdownlistbox`. Its attributes `AIODLListItem` are the various items that can be selected in the `drowdownlistbox`. The choice that is selected by default is specified in the `AIODLItemSel` attribute.

```
</PRESENTATION_ELEMENT></PRESENTATION_ELEMENT>  
<PRESENTATION_ELEMENT ID="CPTable_Row2">  
<Name>Second row of Pay by Credit Card Table </NAME>  
<PRESENTATION_ELEMENT ID="CPTable_Cell2_1">  
<NAME> First Cell </NAME>  
<PRESENTATION_ELEMENT ID="NbrCreditLabel">  
<NAME>credit card Number Label</NAME> <FEATURES>  
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelString">  
Credit Card Number<ATTRIBUTE_STATEMENT> ((7))  
</FEATURES></PRESENTATION_ELEMENT>
```

((7)) This label should be linked to its corresponding edit box (located in next cell), but as these presentation elements are contained in a table, the relation is not added because this information is already given by the table structure.

```
</PRESENTATION_ELEMENT>  
<PRESENTATION_ELEMENT ID="CPTable_Cell2_2">  
<NAME>Second Cell </NAME>  
<PRESENTATION_ELEMENT ID="PayCcNumTextBox"> ((8))  
<NAME>Credit Card Number textbox</NAME> <FEATURES>  
<ATTRIBUTE_STATEMENT DEFINITION="AIOEDXSize">25</ATTRIBUTE_STATEMENT>  
</FEATURES> </PRESENTATION_ELEMENT>
```

((8)) this presentation element represents the input node that asks the user to give its credit card number. It possesses only one attribute, `AIOEDXSize`, which indicates maximum length of the input (expressed in number of characters).

```
</PRESENTATION_ELEMENT> </PRESENTATION_ELEMENT>  
<PRESENTATION_ELEMENT ID="CPTable_Row3">  
<Name>Third row of Pay by Credit Card Table </NAME>  
<PRESENTATION_ELEMENT ID="CPTable_Cell3_1">  
<NAME> First Cell </NAME>  
<PRESENTATION_ELEMENT ID="ExpirCreditLabel">  
<NAME>Expiration date of the credit card Label</NAME> <FEATURES>  
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelString">  
Expiration Date: </ATTRIBUTE_STATEMENT>  
</FEATURES></PRESENTATION_ELEMENT>  
</PRESENTATION_ELEMENT>  
<PRESENTATION_ELEMENT ID="CPTable_Cell3_2"> ((9))  
<NAME>Second Cell </NAME>  
<PRESENTATION_ELEMENT ID="CcXMonthDropDownListBox">  
<NAME>Expiration of CC - Month Selection List</NAME> <FEATURES>  
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">  
Jan.</ATTRIBUTE_STATEMENT>  
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">Feb.</ATTRIBUTE_STATEMENT>  
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">Mar.</ATTRIBUTE_STATEMENT>...  
<ATTRIBUTE_STATEMENT DEFINITION="AIODLItemSel">Jan.</ATTRIBUTE_STATEMENT>  
</FEATURES></PRESENTATION_ELEMENT>
```

((9)) This cell contains the two drop down list boxes allowing the user to select the expiration date of the credit card.

```
<PRESENTATION_ELEMENT ID="CcXYearDropDownListBox">  
<NAME> Expiration of CC - Year Selection List</NAME> <FEATURES>  
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">  
2000</ATTRIBUTE_STATEMENT>
```

Appendix H

```
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">2001</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIODLListItem">2002</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIODLItemSel">2000</ATTRIBUTE_STATEMENT>
</FEATURES></PRESENTATION_ELEMENT> </PRESENTATION_ELEMENT>
</PRESENTATION_ELEMENT>
<PRESENTATION_ELEMENT ID="CPTable_Row4">
<Name>Fourth row of Pay by Credit Card Table </NAME>
<PRESENTATION_ELEMENT ID="CPTable_Cell4_1">
<NAME> First Cell </NAME>
<PRESENTATION_ELEMENT ID="NACreditLabel">
<NAME>Name and address of the owner of the credit card Label</NAME> <FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="AIOLabelString">
Name and address of cardholder (if different from registrant info)
</ATTRIBUTE_STATEMENT></FEATURES></PRESENTATION_ELEMENT>
</PRESENTATION_ELEMENT>
<PRESENTATION_ELEMENT ID="CPTable_Cell4_2">
<NAME>Second Cell </NAME>
<PRESENTATION_ELEMENT ID="PayCCNameEDM" ( (10) )>
<NAME>Multi-line textbox for the name of the cardholder</NAME> <FEATURES>
<ATTRIBUTE_STATEMENT DEFINITION="AIOEDMCol">25</ATTRIBUTE_STATEMENT>
<ATTRIBUTE_STATEMENT DEFINITION="AIOEdMRow">3</ATTRIBUTE_STATEMENT>
</FEATURES></PRESENTATION_ELEMENT>
</PRESENTATION_ELEMENT></PRESENTATION_ELEMENT></PRESENTATION_ELEMENT>
```

((10))The PayCCNameEDM represents the `textarea` tag from the HTML code. It is translated by an **extended edit box** (see table 5-3) and possesses 2 attributes defining its vertical and horizontal size (AIOEDMCol and AIOEdMRow expressed in number of characters).

H-2 Example of reverse engineering of WML

The first example is the reverse engineering of the yahoo weather service. The UI of this WML page is shown on figure H-2 in two different browsers.

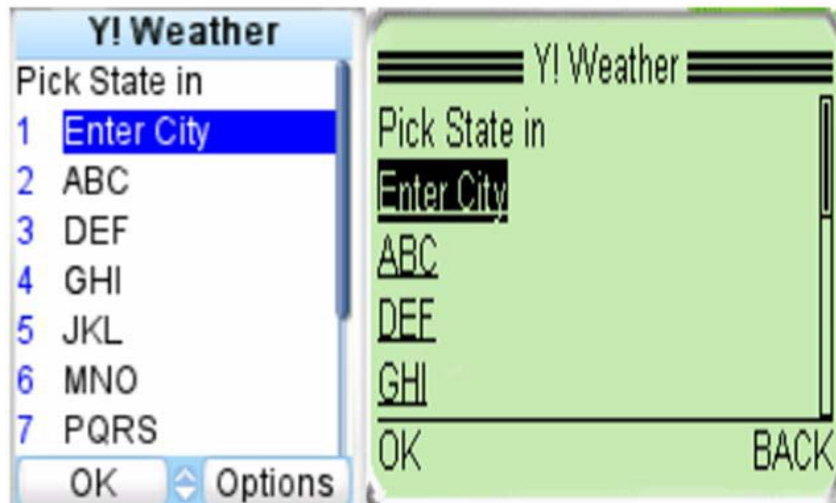


Figure H-2 Yahoo mobile weather service

Figure H-2 represents a weather report application. The user has eight options, to choose the first character of the city in which the user is interested. When he has done his selection, he can choose to validate his choice by pushing on the ok button-link, or he can still navigate in this WML site. The WML code is shown on table H-1 (left) so as its corresponding CUI specification (right). Some comments about the code and the UsiXML specification are given below the table.

<u>WML</u>	<u>UsiXML</u>
<pre> <wml><head> <meta http-equiv="Cache-Control" content="max-age=0" forua="true"/> </head> <template> ((1)) <do type="prev" label="BACK"><prev/></do> </template> <card id="c1" title="Y! Weather">((2)) <do type="accept" label="OK"> <go href="/raw?XL=Idv13w&J4X =g&F1d=\$(let) &IM=&JY MRdvo=&JRdv=&sd=L8vwdMP" />((3)) </do> <p> Pick State in <select name="let"> <option title="OK" >Enter </pre>	<pre> <CuiModel xmlns="http://www.UsiXML.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance"> <window id="window_1" borderTitle="Y! Weather" isVisible="true" isenabled="true" name="Y! Weather"> <box id="box_1" isVisible="true" isenabled="true">((2)) <textComponent id="textComponent_0" glueHorizontal="left" isVisible="true" isenabled="true" hyperlinkTarget="/raw?XL= Idv13w&J4X=g&F1d=\$(let) &IM=&JY MRdvo=&JRdv=&sd=L8vwdMP " defaultContent="OK"/>((3)) <textComponent id="textComponent_1" glueHorizontal="left" isVisible="true" isenabled="true" defaultContent="Pick State in"/> <comboBox id="combo_1" isVisible="true" isenabled="true" isEditable="false"/>((4)) </pre>

Appendix H

<pre> City</option>((4)) <option title="OK" >ABC</option> <option title="OK" >DEF</option> <option title="OK" >GHI</option> <option title="OK" >JKL</option> <option title="OK" >MNO</option> <option title="OK" >PQRS </option> <option title="OK" >TUV</option> <option title="OK" >WXYZ </option> </select> </p> </card> </wml> </pre>	<pre> <item id="item_1" defaultContent="Enter City" isVisible="true" isEnabled="true" /> <item id="item_2" defaultContent="ABC" isVisible="true" isEnabled="true" /> <item id="item_3" defaultContent="DEF" isVisible="true" isEnabled="true" /> <item id="item_4" defaultContent="GHI" isVisible="true" isEnabled="true" /> <item id="item_5" defaultContent="JKL" isVisible="true" isEnabled="true" /> <item id="item_6" " defaultContent="MNO" isVisible="true" isEnabled="true" /> <item id="item_7" " defaultContent="PQRS" isVisible="true" isEnabled="true" /> <item id="item_8" " defaultContent="TUV" isVisible="true" isEnabled="true" /> <item id="item_9" " defaultContent="WXYZ" isVisible="true" isEnabled="true" /> </comboBox></box></window> <graphicalTransition id="textLink_0" type="open">((5)) <source sourceId="textComponent_0"/> <target targetId="="/raw?XL= Idv13w&amp;J4X=g&amp;Fld=\$(let) &amp;IM=&amp; JYMRdvo=&amp;JRdv=&amp;sd=L8vwdMP"/> </graphicalTransition></CuiModel> </pre>
---	---

Table H-1 Reverse engineering of Yahoo mobile service

- (1) The **template** node is not recovered in the UsiXML specification, as all the conditions of the group of rules G1b are satisfied except one: to have an **href** attribute in the **<do>** node. The **template** node contains a reference to the previous visited card (**<prev>**) but the concept of history can not be represented in UsiXML.
- (2) The **<card>** node is translated as a new **window**. The **title** attribute of the **card** is used twice, once for the **name** attribute of the **window** and once for the **borderTitle** attribute of the window (set of rule G1b)
- (3) The association of the **do** and **go** node are conditions of the set of rules G21. As the **go** node possess a **href** attribute, these two nodes are represented by a **textComponent** in the UsiXML specification representing a link to another card. A **graphicalTransition** is also added in the CUI model (see remark 5)
- (4) The **select** tag is translated by a **comboBox**, as the number of choices is greater than 6, no multiple selections are allowed and none of the options posses an **onpick** attribute. The set of rules G8 and G9 are used to derive this component.
- (5) All the **graphicalTransitions** are specified at the end of the CUI model. It means that all the links of the page have to be reported in this section. In this case, only one link is recovered (as the **template** tag cannot be recovered). The **source** object of this **graphicalTransition** is specified (**textComponent_0**) so as its **target** (an URL external to the deck).

Appendix H

The next example is the reverse engineering of the wordaholic's mobile games (see two representation on different browsers of the WML UI in figure H-3). The user has again three choices, but this time the choices results are not sent back to the server, but give an access to another card. On the left part of figure H-3, the choices are represented as a listbox while on the other side, they are already represented as links. The code of the WML card and the CUI model are given in table H-2.

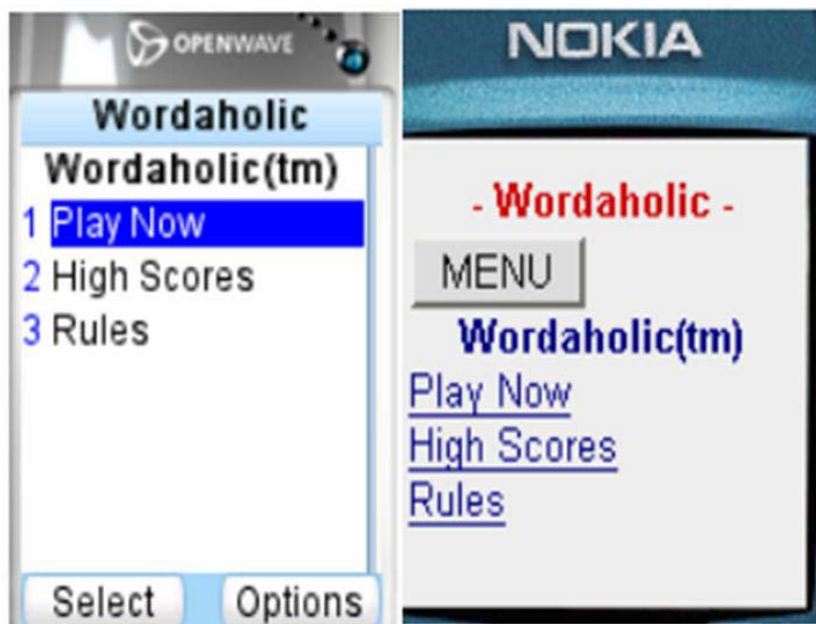


Figure H-3 WAP's Wordaholic game

WML	UsiXML
<pre><wml><head> <meta http- equiv="Cache-Control" content="max-age=0" forua="true"/> </head> <card title="Wordaholic"> <p align="center"> Wordaholic (tm) </p> ((1)) <p align="left" mode="nowrap"> <select title="options"> <option onpick="href1">Play Now</option> ((2)) <option onpick="href2">High Scores</option> <option onpick="href3">Rules </option> </p> </card></pre>	<pre><CuiModel xmlns="http://www.UsiXML.org" xmlns:xsi="http://www.w3.org/2001/XMLSchema- instance"> <window id="window_1" borderTitle="Wordaholic" isVisible="true" isEnabled="true" name="Wordaholic"> <box id="box_1" isVisible="true" isEnabled="true"> <textComponent id="textComponent_1" glueHorizontal="center" isVisible="true" isEnabled="true" isBold="true" defaultContent="Wordaholic (tm)"/> ((1)) <textComponent id="textComponent_2" ((2)) glueHorizontal="left" isVisible="true" isEnabled="true" defaultHyperLinkTarget="href1" defaultContent="Play Now"/> <textComponent id="textComponent_3" glueHorizontal="left" isVisible="true" isEnabled="true" defaultHyperLinkTarget="href2" defaultContent="High Scores"/> <textComponent id="textComponent_4" glueHorizontal="left" isVisible="true" isEnabled="true" defaultHyperLinkTarget="href3" defaultContent="Rules"/></pre>

Appendix H

```
</wml> | </box> </window>  
        | <graphicalTransition id="textLink_1"  
        | type="open"><source sourceId="textComponent_2"/>  
        | <target targetId="href1"/>  
        | </graphicalTransition> ((3))... </CuiModel>
```

Table H-2 Reverse engineering of Wap's Wordaholic game

((1)) The “wordaholic (tm)” label is embedded in a **** node, and is therefore represented as **textComponent** with the attribute **isbold** set to **true** in the UsiXML specification (rule set G17).

((2)) As in the previous example, a **select** node is specified in the WML code. Here the rule G9d is applied, as the the options depending on this select node posses an **onpick** attribute. Therefore they are derived into **textComponents** with an **hyperLinkTarget** attribute set at the same value as in the **onpick** attribute.

((3)) A set of **graphicalTransition** are also added at the end of the model, where each **option's** id is specified as the **source** and the corresponding **value** of their **onpick** attribute as the **target** of the transition.

H-3 Example of reverse engineering of VoiceXML

As no forward engineering UsiXML-based tools for vocal languages exist, this example section will only be dedicated to the reverse engineering of VoiceXML examples.

The first example is a simplified version (duplicated tags representing the same concept have been removed) of the “virtual poll system” adapted by A. Stanciulescu in its VoiceXML version.

The following code represents a simple vocal form asking several questions (name, zip code, age, and some questions about courses) to the user. At the end of the form, the user is asked if he wants to send the form.

VoiceXML code:

```
(1)<vxml version="2.0" xml:lang="en-US" xmlns="http://www.w3.org/2001/vxml">
  <meta name="GENERATOR" content="Voice Toolkit for WebSphere Studio" />
  (2)<form id="welcome"> <block> Welcome to the polling system. </block> </form>
  <form id="nameandzipcodeandgender">
    (3)<record name="rec1" beep="true" maxtime="3s">
      <prompt>Please say your name after the sonor signal</prompt>
    </record>
    <filled><prompt>Your name is: <audio expr="rec1"/></prompt>
    </filled>
    <record name="rec2" beep="true" maxtime="3s">
      <prompt>Please say your zipcode after the sonor signal</prompt>
    </record>
    <filled> <prompt>Your zip code is: <audio expr="rec2"/></prompt>
    </filled>
  </form>
  <form id="age">
    (4) <field name="f_age">
      <grammar src="../Virtual%20polling%20system/age.grxml" mode="voice" />
      <prompt>What is your age category?</prompt>
      <filled>Your age category is: <value expr="f_age"/></filled>
    </field>
    <catch event="noinput">
      <prompt>There was no input</prompt> <goto next="#age"/></catch>
  <block> Now we can start the questionnaire. Please answer to the following questions.
</block></form>
  <form id="question1">
    <field name="f_q1">
      <grammar src="../Virtual%20polling%20system/answer.grxml" mode="voice" />
      <prompt>Did you enjoy out teaching course?</prompt>
    (5)<filled>Your answer is: <value expr="f_q1"/></filled>
    </field>
```


Appendix H

```
<catch event="noinput">
  <prompt>There was no input</prompt><goto next="#question1"/> </catch>
</form>
...
<form id="send_questionnaire">
  <field name="send" type="boolean">
    <prompt> Would you like to send the questionnaire?</prompt>
    <filled>
      ((6)) <if cond="" send==true">
        You have chosen to sent the questionnaire.
      <else/> The questionnaire wasn't send.</if>
    </filled>
    <catch event="noinput">
      <prompt>There was no input</prompt>
      ((7))<goto next="#send_questionnaire"/>
    </catch> </field> </form> </vxml>
```

UsiXML CUI code

```
...
((1)) <vocalGroup id= "_1" name= "_1">
((1)) The vxml node is derived into a vocalGroup node by applying the set of rules
G22. The filename attribute added in G22 is deleted after the parsing of the entire
tree (see group of rule G50).
((2)) <vocalForm id= "welcome" name= "welcome">
<vocalGroup id= "_3" name= "_3">
<vocalPrompt id= "_4" name= "_4" defaultContent="Welcome to the polling system."/>
</vocalGroup></vocalForm>
<vocalForm id= "nameandzipcodeandgender" name= "nameandzipcode">
((2)) The first form is translated by a vocalForm in the target tree (set of rules
G28). The id of the form is copied in the id and name attributes in the UsiXML
specification.
((3))<vocalPrompt id= "_6" name= "_6" defaultContent="Please say your name after the
signal"/>
((3))<vocalInput id= "rec1" name= "rec1" elapsedTime="3s" />
((3)) When the record node has been detected, nothing was added to the UsiXML
specification. The next node is a prompt node that is transformed thanks to the
rule set G35. This rule set checks if the prompt is embedded in a record node (and
not in a field or initial node), and if it is the case, adds a vocalInput node just after
the creation of the vocalPrompt containing the question "Please say your name
after the signal". The maxtime attribute is mapped in the elapsedTime attribute in
the UsiXML specification. This attribute represents the time the browser will
record the user's answer.
<vocalPrompt id= "_8" name= "_9" defaultContent= "your name is:"/>
<vocalPrompt id= "_9" name= "_9" defaultContent="rec1"/>
<vocalPrompt id= "_10" name= "_10" defaultContent=" Please say your zipcode after the sonor
signal"/>
<vocalInput id= "rec2" name= "rec2" elapsedTime="3s" />
<vocalPrompt id= "_13" name= "_13" defaultContent= "your zipcode is:"/>
```

Appendix H

```
<vocalPrompt id= "_14" name= "_14" defaultContent="rec2"/>
<vocalForm id="age" name="age">
((4))<vocalPrompt id= "_16" name= "_16" defaultContent=" What is your age category?"/>
((4))<vocalInput
                                id= "f_age"
                                name= "f_age"
grammar="../Virtual%20polling%20system/age.grxml"/>
((4) The field node is transformed into a vocalInput (rules set G24). As the field
node possesses a grammar attribute, its value is recorded in the vocalInput's
grammar attribute. Similarly to the previous commented element, the prompt
contained in the field is processed before the field itself.
<vocalPrompt id= "_18" name= "_18" defaultContent="Your category is"/>
<vocalPrompt id= "_19" name= "_19" defaultContent="f_age"/>
<!--optional
<vocalPrompt id= "_20" name= "_20" defaultContent="There was no input"/>
<vocalNavigation id="_21" name="_21" navigationType="goto" isBridgeable="false"/> -->
<vocalGroup id="_22" name="_22">
<vocalPrompt id= "_23" name= "_23" defaultContent="Now we can start the questionnaire.
Please answer to the following questions."/>
</vocalGroup>
</vocalForm>
<vocalForm id="question_1" name="question_1">
<vocalPrompt id= "_25" name= "_25" defaultContent=" Did you enjoy out teaching course?"/>
<vocalInput id= "f_q1" name= "f_q1"
grammar="../Virtual%20polling%20system/answer.grxml" />
<vocalPrompt id= "_27" name= "_27" defaultContent="Your answer is :"/>
((5))<vocalPrompt id= "_28" name= "_28" defaultContent="f_q1"/>
</vocalForm>
```

((5)) The **filled** element is a **conditional** element that is not taken into account. It means that if the preceding **field** is filled, the content of the **filled** element is executed. We can consider this condition as implicit, since the control flow is blocked when the user is asked to say something. The content of the filled element is a **textnode** (rule set G30) and an **audio** element. The first is translated by a **vocalPrompt** with its **defaultContent** attribute set to "Your answer is:". The audio element is transformed into a **vocalPrompt** too, but as its **textnode** does not exist, its **expr** attribute is used to set the value of the **defaultContent** attribute: **f_q1**, which is a variable name. There is no way to differentiate in UsiXML the **defaultContent** attribute representing a **textnode** or a variable name.

```
...
<vocalForm id="send_questionnaire" name="send_questionnaire">
<vocalPrompt id= "_35" name= "_35" defaultContent="Would you like to send the
questionnaire?"/>
<vocalInput id= "send" name= "send" grammar="boolean" />
((6) <!--optional
<vocalPrompt id= "_37" name= "_37" defaultContent=" You have choosed to sent the
questionnaire."/>
<vocalPrompt id= "_38" name= "_38" defaultContent=" The questionnaire wasn't send."/>
<vocalPrompt id= "_39" name= "_39" defaultContent="There was no input"/>
```

((6)) All the nodes here are children of the **if** tag or the **catch** tag. As it cannot be expressed in UsiXML, the children of the **if/catch** nodes are reverse engineered,

Appendix H

but under the form of a comment (between `<!-- -->` tags). The three **textnodes** (the rule applied belongs to the general rules for isolated **textnodes**) reflecting the response to the different possibilities are recorded sequentially in the comment. The designer could reuse them in a future reification but would have to specify the conditions again.

```
((7))<vocalNavigation id="_40" name="_40" navigationType="goto" isBridgeable="false"/>
-- ></vocalForm> </vocalGroup>
<auditoryTransition type=open id="_41">
<source source="_21"/>
<target target="age"/>
</auditoryTransition>
((7))<auditoryTransition type=open id="_42">
<source source="_40"/>
<target target="send_questionnaire"/>
</auditoryTransition>
</cuiModel>
```

((7)) As for the previous element, this element will be put in comments tags as it depends on the catch tag. The goto element (rule set G43) is translated by a `vocalNavigation` element. The attribute `navigationType` is set to `goto` and `isBridgeable` attribute is set to `false` as the current document lose control if the goto node targets another document. An `auditoryTransition` is also created at the bottom of the CUI Model, taking as source the current element's id (`_40`) and as target the id of the element to which the control flow will continue.

The second example comes from [Voic02]. This example shows the case of a mixed initiative form and represents weather forecasts service. First, an introductory sentence and an advertisement are played. After that, the user can say the city and country for which he wants the weather forecast. At that moment, he can say both in any order. If he doesn't say anything, the question is asked again. If the initial question is asked three times without response, the control flow exits the initial element and the browser asks the questions sequentially (which country and then which city). At that moment, the user can still answer in any order. A default case is also defined in the VoiceXML code: if the user answers Paris, the country is set to France automatically. Finally, the user is asked a confirmation, and if he confirms his choices, an advertisement is played followed by the forecast for the chosen city.

VoiceXML code:

```
((1)) ...<form id="info_meteo">
((1))<grammar src="ville-et-pays.grxml" type="application/srgs+xml"/>
<block>
<prompt bargein="false"> Bienvenue sur ce service d'informations météorologiques !
((2)) <audio src="http://www.pubs-en-ligne.example.com/wis.wav"/>
</prompt>
</block>
```

Appendix H

```
((3)) <initial name="debut">
  <prompt> Quels sont la ville et le pays dont vous voulez connaître la météo ?
  </prompt>
  <help> Veuillez prononcer le nom de la ville et celui du pays pour lesquels
  vous souhaitez un bulletin météorologique
  </help>
  <noinput count="1"> <reprompt/></noinput>
  <noinput count="2"> <reprompt/>
  <assign name="debut" expr="true"/></noinput>
</initial>
((4))<field name="pays">
  <prompt>Quel pays ?</prompt>
  <help> Veuillez prononcer le nom du pays dont vous voulez connaître la météo.
  </help> </field>
<field name="ville">
  ((5))<prompt>Veuillez prononcer le nom de la ville située en <value expr="pays"/>
  dont vous voulez connaître la météo.</prompt>
  <help>Veuillez prononcer le nom de la ville dont vous
  voulez connaître la météo.</help>
  <filled>
    <if cond="ville == 'Paris' && pays == undefined">
      <assign name="pays" expr="France"/>
    </if> </filled> </field>
  <field name="continuer" modal="true">
    <grammar type="application/srgs+xml" src="/grammars/boolean"/>
    <prompt>Voulez-vous entendre le bulletin météorologique pour
    <value expr="ville"/>, <value expr="pays"/> ?
    </prompt>
    <filled>
      <if cond="continuer">
        ((6)) <prompt bargein="false">
          <audio src="http://www.pubs-en-ligne.example.com/wis2.wav"/>
          </prompt>
          <submit next="/servlet/meteo" namelist="ville pays"/>
        </if>
        <clear namelist="debut ville pays continuer"/>
      </filled> </field></form></vxml>
```

UsiXML code:

```
<vocalGroup id= "_1" name= "_1">
  ((1))<vocalForm id= "info_meteo" name= "info_meteo" isOrderIndependent="true">
  <vocalGroup id= "_3" name= "_3">
```

(1) The `form` element is transformed into a `vocalForm` (rules G28). The value of the attribute `id` is copied into the `name` and `id` attributes of the `vocalForm`. As a `grammar` node belongs to the child nodes of the `form`, an attribute `isOrderIndependent` set to `true` is added to this element (G38).

```
<vocalPrompt id= "_4" name= "_4" defaultContent=" Bienvenue sur ce service d'informations
météorologiques !" isInterruptible="false" />
((2))<vocalPrompt id= "_5" name= "_5" defaultContent="http://www.pubs-en-
ligne.example.com/wis.wav" isInterruptible="false"/>
```

Appendix H

((2)) Following the group of rules G34, the `audio` element is derived into a `vocalPrompt`. Its `defaultContent` contains the value stored in the `src` attribute (an URL to a `.wav` file). As the `audio` element is a child of a `prompt` element with its `bargein` attribute set to `true` in the VoiceXML tree, an attribute `isInterruptible` is added to the `vocalPrompt`. Its value is set to `false` as the user cannot interrupt this advertisement.

```
</vocalGroup>
```

```
((3))<vocalGroup id= “_24” name= “_24” isOrderIndependent=“true”>
```

((3)) The initial element initiates the creation of three elements (a `vocalGroup(Init)`, a `vocalPrompt` and a `vocalInput`) as it satisfies all the conditions of G27. The first element created is the `VocalGroupInit` and it will contain the two other elements in the target tree. When an `initial` node is detected, its children are scanned in order to find a `prompt` element. This prompt is processed, and a `vocalPrompt` is created with the `defaultContent` attribute set to value of the prompt’s `textnode`. Then a `vocalInput` is created, and as the form it belongs to possesses a `grammar`, the `vocalInput` inherits this `grammar` and its attribute `isOrderIndependent` is set to `true`.

After the entire parsing of the source tree, special rules specific to the target tree are applied. The set of rules G53 is applied in this case, and transforms a `vocalGroupInit` into a “normal” `vocalGroup` while creating another `vocalGroup` in which the rest of the form is put.

```
<vocalPrompt id= “_7” name= “_7” defaultContent=“Quels sont la ville et le pays dont vous voulez connaître la météo ?”/>
```

```
<vocalInput id= “debut” name= “debut” isOrderIndependent=“true” grammar=“ville-et-pays.grxml”/>
```

```
<!-- <vocalPrompt id= “_9” name= “_9” defaultContent=“Veuillez prononcer le nom de la ville et celui du pays pour lesquels vous souhaitez un bulletin météorologique”/>
```

```
-->
```

```
<!-- <vocalPrompt id= “_10” name= “_10” defaultContent=“Quels sont la ville et le pays dont vous voulez connaître la météo ?”/> -->
```

```
<!-- <vocalPrompt id= “_11” name= “_11” defaultContent=“Quels sont la ville et le pays dont vous voulez connaître la météo ?”/> -->
```

```
</vocalGroup>
```

```
<vocalGroup id= “_12” name= “_12”>
```

```
<vocalPrompt id= “_13” name= “_13” defaultContent=“ Quel pays ?”/>
```

```
((4))<vocalInput id= “pays” name= “pays” isOrderIndependent=“true” grammar=“ville-et-pays.grxml”/>
```

((4))The element field is transformed into a `vocalInput`. But as for the previous remark **((3))**, the prompt element is processed first. As the field element does not have a `grammar` defined in its child nodes, the `grammar` for the `vocalInput` will be the nearest form’s `grammar` (G24). The `vocalInput` also possesses the `isOrderIndependent` set to `true` as it inherits a form `grammar`.

The `vocalInput` has been put in an “artificial” `vocalGroup`, as the form containing the field has an `initial` element in its children (see remark **((3))**).

Appendix H

```
<!-- <vocalPrompt id= "_15" name= "_15" defaultContent=" Veuillez prononcer le nom du
pays dont vous voulez connaître la météo."/> -->
<vocalPrompt id= "_16" name= "_16" defaultContent=" Veuillez prononcer le nom de la ville
située en dont vous voulez connaître la météo"/>
<vocalInput id= "ville" name= "ville" isOrderIndependent="true" grammar="ville-et-
pays.grxml"/>
<!-- <vocalPrompt id= "_18" name= "_18" defaultContent=" Veuillez prononcer le nom de la
ville dont vous voulez connaître la météo."/> -->
((5))<vocalPrompt id= "_19" name= "_19" defaultContent=" Voulez-vous entendre le bulletin
météorologique pour"/>
<vocalInput id= "continuer" name= "continuer" grammar="/grammars/boolean"/>
```

((5)) This prompt is translated by a `vocalPrompt` following the rules set G35. The `textnode`, composing the `defaultContent` attribute of the `vocalPrompt`, contains a value node that cannot be recovered following the proposed method. Therefore, the content of the `vocalPrompt` in the UsiXML specification is not very meaningful.

```
((6))<!-- <vocalPrompt id= "_21" name= "_21"
isInterruptible="false" defaultContent="http://www.pubs-en-ligne.example.com/wis2.wav"/>
<vocalNavigation id="_22" name="_22" navigationType="submit" isBridgeable="false"/> -- >
</vocalGroup> </vocalForm> </vocalGroup>
<!-- <auditoryTransition type=open id="_23">
<source source="_22"/>
<target target="/servlet/meteo"/>
</auditoryTransition-->...
```

((6)) All the last nodes of the VoiceXML tree and embedded in an if condition. Therefore, the sending of the `form` (the `submit` node) and the last advertisement (the `audio` node) are put into comments in the UsiXML specification.

In this last example, there are several losses due to the reverse engineering method and output language, as complex control flows can not be represented. In the resulting UsiXML specification, the user would never reach the inputs “pays” and “ville” but would stay in the input “début” until he gives a correct answer. Then he would skip the two next inputs and go to the “continuer” input field.

He would also not be able to send the form and prompts `_16` and `_19` would be played incompletely (as they use variables to constitute the sentence). In this case, the designer should modify the produced model manually (or the reified code) to correct elements and the structure of the vocal UI.

H-4 Example of reverse engineering of .RC files

The UI that has been reverse engineered for this case study is a “find” dialog box (see fig. H-4). The UI is composed of a comboBox, several checkboxes, two radio buttons and three buttons.



```

01:400 DIALOG 30, 73, 275, 84
02:STYLE DS_SETFONT | DS_MODALFRAME | DS_3DLOOK | WS_POPUPWINDOW | WS_DLGFRAME
03:CAPTION "Find"
04:LANGUAGE LANG_ENGLISH, SUBLANG_ENGLISH_US
05:FONT 8, "MS Shell Dlg"
06:{
07:   LTEXT "Fi&nd what:", -1, 5, 7, 45, 8
08:   COMBOBOX 222,50,5,145,50,NOT CBS_SIMPLE|CBS_DROPDOWN| CBS_AUTOHSCROLL
09:   AUTOCHECKBOX "Match &whole word only", 232, 5, 22, 120, 10, WS_GROUP
10:   AUTOCHECKBOX "Match &case", 233, 5, 34, 130, 10, WS_GROUP
11:   AUTOCHECKBOX "Regular &expression", 239, 5, 46, 120, 10, WS_GROUP
12:   AUTOCHECKBOX "Wrap aroun&d", 240, 5, 58, 120, 10, WS_GROUP
13:   AUTOCHECKBOX "Transform &backslash expressions", 241, 5, 70, 160, 10, WS_GROUP
14:   GROUPBOX "Direction", -1, 135, 22, 60, 34, WS_GROUP
15:   AUTORADIOBUTTON "&Up", 234, 140, 30, 45, 12, NOT WS_TABSTOP | WS_GROUP
16:   AUTORADIOBUTTON "&Down", 235, 140, 42, 45, 12, NOT WS_TABSTOP
17:   DEFPUSHBUTTON "&Find Next", 1, 205, 5, 65, 14, WS_GROUP
18:   PUSHBUTTON "&Mark All", 245, 205, 23, 65, 14
19:   PUSHBUTTON "Cancel", 2, 205, 41, 65, 14
20:}

```

Figure H-4 Find dialog box resource script

The first excerpt of code reverse engineered is the `dialogBox` specification, corresponding to the five first lines of code, which is derived thanks to the rules described in section 8.3 into:

```

<dialogBox id="1" name="400" defaultContent="Find" isVisible="true"
isEnabled="true" height="172" width="423">
<Box id="2" name="2" type="vertical" isVisible="true" isEnabled="true">

```

The `id` is automatically generated and takes the value 1, while the `name` is equivalent to the id number of the `dialogbox`, i.e. 400. The `defaultContent` value can be found in the `CAPTION` element. As nothing is specified in the resource script to negate the attributes `isVisible` and `isEnabled`, they are set to true. Finally, the size is computed: `height` is equal to 172 pixels $((84 * 274 / 21 * 8) + 35)$ and `width` is equal to 423 pixels $((275 * 3304 / 275 * 8) + 10)$. A vertical `box` is then appended to the `dialogBox` and will contain the rest of the UI specification.

Appendix H

The second excerpt analyzed is the derivation of the AUTOCHECKBOX element on line 10 into a `checkBox`:

```
<Box id="7" name="7" type="horizontal" isVisible="true" isEnabled="true">
<checkBox id="8" name="239" isVisible="true" isEnabled="true"
defaultMnemonic="c" />
<textComponent id="9" name="9" defaultContent="Match case" isVisible="true"
isEnabled="true" textSize="8" textFont="Ms Shell Dlg" /> </box>
```

The `name` attributes corresponds to the first integer from the line 10, which represents the identifier of the Windows UI component. The other attributes are generated automatically, except for the `defaultMnemonic` attribute: its value is the character next to the ampersand symbol (c in this case) in the `textnode` of the `AUTOCHECKBOX`. As this `textnode` is not null, a `textComponent` is also created and its `defaultContent` is set to the `AUTOCHECKBOX textnode` value without the ampersand symbol (string parsing functions here are similar to those described in section 8.3 about `radioButtons`). `TextSize` and `TexFont` attributes are set thanks to the values defined at the dialog level, in the `FONT` element (line 5). The detection of an `AUTOCHECKBOX` triggers the creation of a horizontal `box` and the two elements are appended to this `box`, to represent the fact that these two elements should be displayed together.

The last excerpt of code is the reverse engineering of lines 14 to 17, representing a `groupbox`, two `radio buttons` and one `button`.

```
<Box id="18" name="18" type="vertical" isVisible="true" isEnabled="true"
height="90" width="55" defaultBorderTitle="Direction">
<Box id="19" name="19" type="horizontal" isVisible="true" isEnabled="true">
<radioButton id="20" name="234" isVisible="true" isEnabled="true"
defaultMnemonic="u" />
<textComponent id="21" name="9" defaultContent="Up" isVisible="true"
isEnabled="true" textSize="8" textFont="Ms Shell Dlg" /> </box>
<Box id="22" name="22" type="horizontal" isVisible="true" isEnabled="true">
<radioButton id="23" name="235" isVisible="true" isEnabled="true"
defaultMnemonic="d" />
<textComponent id="24" name="24" defaultContent="Down" isVisible="true"
isEnabled="true" textSize="8" textFont="Ms Shell Dlg" /> </box> </box>
<button id="25" name="205" isVisible="true" isEnabled="true"
defaultMnemonic="f" defaultContent="Find Next" />
```

The `groupbox` element is translated by a `box` in the UsiXML specification. As the `CtrlId` attribute of the `groupbox` is -1 (it signifies that the element has no particular identifier in the resource script), its `id` is generated automatically in UsiXML. The `textnode` "direction" is copied in the `defaultBorderTitle` attribute. `Height` and `width` for this component are computed thanks to the following formulas: `height= (h x 274) / (21 x fontsize)` and `width= (w x 3304)/(275 x fontsize)`. As the `fontsize` is equal to 8, `h` to 34 and `w` to 60, the `height` is equal to 90 pixels and `width` to 50 pixels in the CUI model. The other attributes for this `box` are generated automatically. Note that this `box` contains the two couple `radioButtons /textComponents` in the UsiXML specification, but this information could not be

Appendix H

recovered with the current method as the hierarchy and the layout are not reverse engineered (see section 8.2).

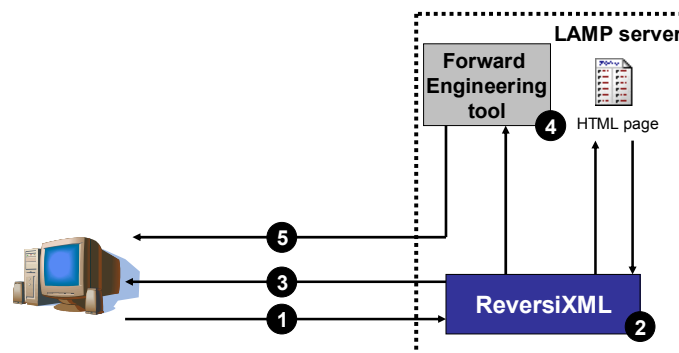
Then the derivation of the **AUTORADIOBUTTON** results in the creation of a horizontal **box** containing a **radioButton** and a **textComponent** in UsiXML, following the rules described in section 8.3. The name of the **radioButton** is derived from the first integer of line 15(**CtrlId**) and its **defaultMnemonic** attribute corresponds to the character following the ampersand in its **textnode**. A **textComponent** is also added to the CUI model, with the **textnode** as its **defaultContent**. As for the checkbox's **textComponent**, the **textFont** and **textSize** attributes are derived from the **FONT** element (line 5). The second **AUTORADIOBUTTON** is derived similarly.

Finally, the **DEFPUSHBUTTON** is derived into a **button** in the CUI model. The **CtrlId** attribute is equal to "205" and represents the **name** of this component in UsiXML. Its **defaultContent** corresponds to the **textnode** "Find Next" and its **defaultMnemonic** is the first character after the '&' symbol (f in this case). Other attributes of the button element are generated automatically.

Appendix I Evolution of the architecture of the tool

This appendix describes a study of the different step for the evolution of the architecture of the tool. The architecture designed for the tool is evolutionary and incremental, so that it can be adapted to new requirements and also insure backward compatibility with previous versions. The focus of this appendix is to show how the different actors (users/servers) are involved in the reengineering process. To do this, each step is illustrated by a diagram representing the transfers from the files and models graphically across the system and by a sequence diagram to show precisely by who and where operations are executed. The current implementation of the tool is the version 1.1, as it represents a good compromise between resources costs (in terms of implementation) and the capacities of the tool.

ReversiXML 1.0, in its first version, was only able to process a page at a time and to send the results back to the user without modification of the original structure and composition of the UI (see figure I-1).



Appendix I

- 1 *Web page request* : the user asks for a web page which is located on the same server as ReversiXML
- 2 *Reverse Engineering*: the reverse engineering is performed on the server, without modification of the original page
- 3 *Send Reverse Engineered Results*: the results are sent back to the user or can directly be used in step 4 to produce a new UI
- 4 *Forward Engineering*: the results of step 2 are used by a forward engineering tool (e.g. teresaXML or grafiXML) to generate UI code. This transformation is also performed on the server.
- 5 *Send Reverse Engineered Results*: the new UI is sent back to the user.

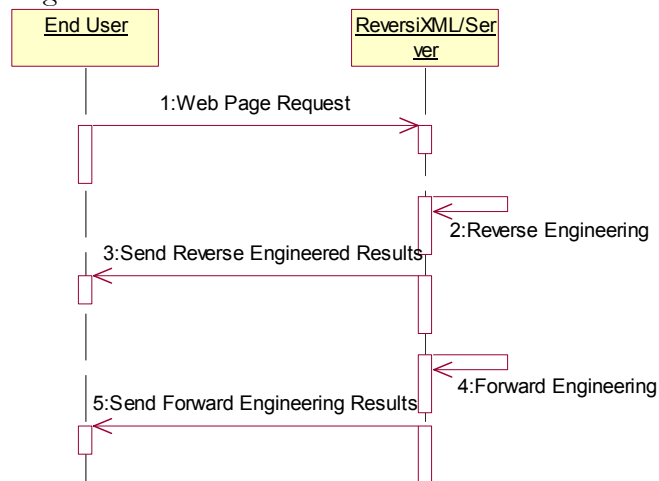
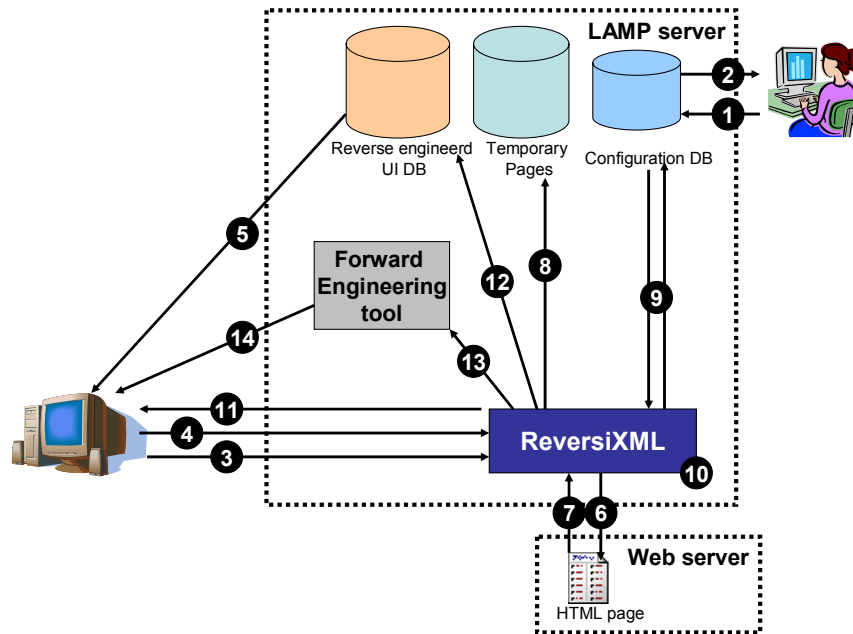


Figure I-1 Sequence diagram for ReversiXML 1.0

In a second step (figure I-2) some features were added to *ReversiXML 1.1*: the possibility to create configuration files and so make the process of reverse engineering flexible and adaptable to a specific platform.

Databases(DB) are also added to the reengineering system in order to accelerate the process. A DB containing all the reverse engineered models is held on the server for a period of time (as the source page may evolve).

Appendix I



- 1 *Create configuration file*: the designer defines a new configuration file associated with a specific context
- 2 *Send confirmation of creation*: when the file has been successfully added to the database
- 3 *Web page request*: the user makes a request for a web page
- 4 *Send user profile*: a basic description of the connected platform is automatically sent to the server with the request of the web page (step 3)
- 5 *Send Reverse Engineered Web Page [if already analysed page]*: the database containing the web pages that have already been reverse engineered is checked. If the page is in this database, the differences between current page and page saved on server is checked, if there is no difference, it is sent back to the user.
- 6 *Ask Web Page [if never analysed page]*: if the UI model of the web page does not exist in the database, a request is sent to a distant server.
- 7 *Send Web Page*: the code of the web page is sent back to the ReversiXML's server
- 8 *Save Page on the Server*: The page is saved into a temporary repository of web page for two reasons: firstly, it speeds up the process by avoiding downloading again the same page, secondly, it eases the reverse engineering by working on a local file.
- 9 *Choose Configuration File*: By using the user profile sent during step 4, a

Appendix I

- configuration file is selected in the configuration knowledge base.
- 10 *Reverse Engineering* : the reverse engineering is performed on the server, following the options contained in the configuration file
- 11 *Send Reverse Engineered Results*: the results are sent back to the user or can directly be used in step 13 to produce a new UI
- 12 *Save Results in Knowledge Base*: the output of ReversiXML is then stored in database for a future reuse.
- 13 *Forward Engineering*: the results of step 10 are used by a forward engineering tool (e.g. TeresaXML or GrafiXML) to generate UI code. This transformation is also performed on the server.
- 14 *Send Reverse Engineered Results*: the new UI is sent back to the user.

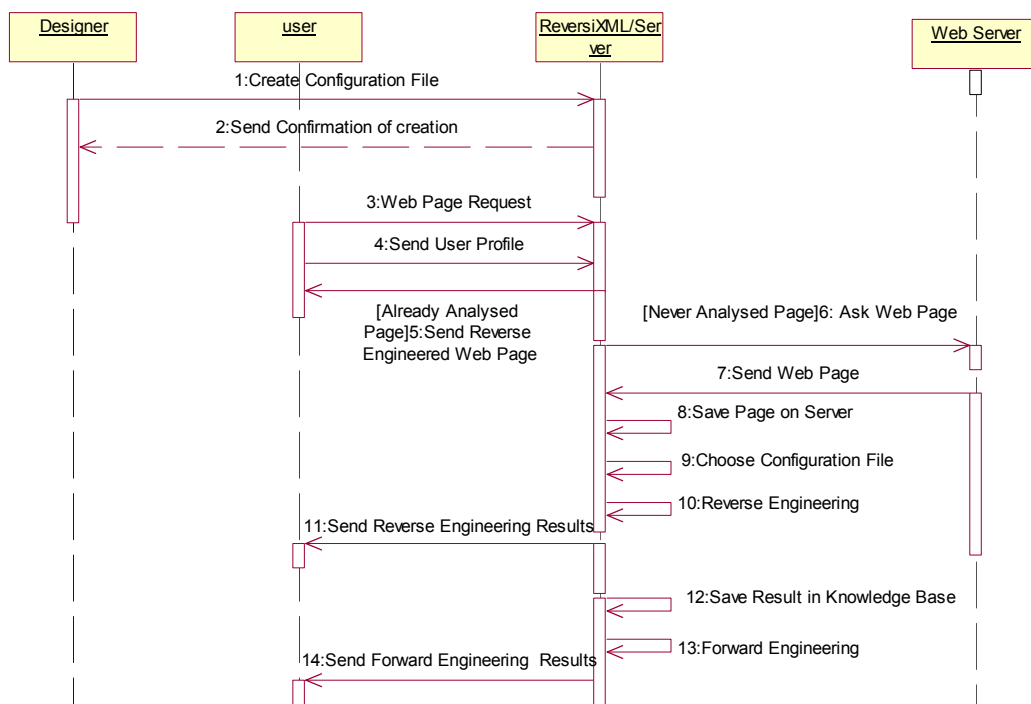


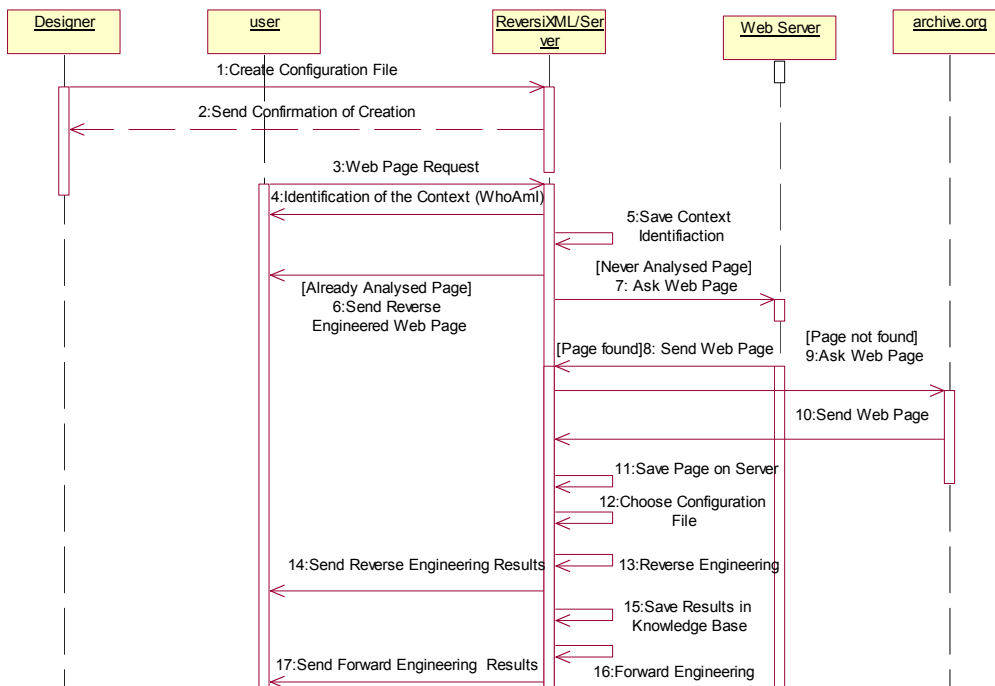
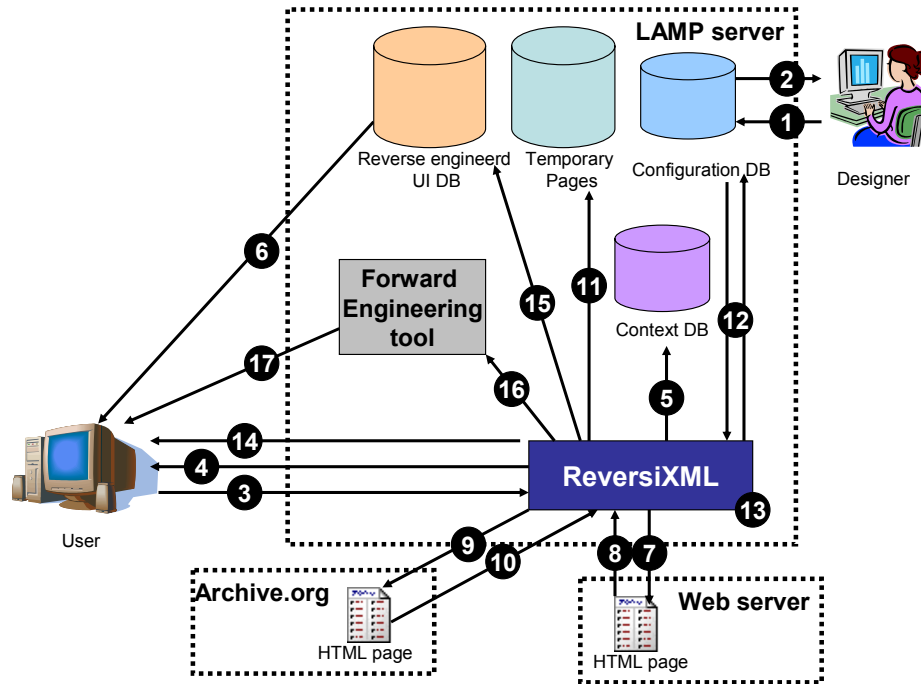
Figure I-2 ReversiXML 1.1

The WhoAmI system (developed by IS3) has been added for the envisioned *version 1.2* of ReversiXML (see figure I-3). Thanks to this system, a more accurate identification of the context of use is possible (user and platform). IS3 developed this system for commercial purposes, and thanks to a database, store the various user/platform profiles connecting to their websites. A similar database could also be added to store this information.

Finally, the possibility is let to the user to reverse engineer web pages that have

Appendix I

been suppressed, thanks to www.archive.org. This site store different version from thousands of sites, and keep them after their suppression.



Appendix I

- 1 *Create configuration file*: the designer defines a new configuration file associated with a specific context.
- 2 *Send confirmation of creation*: when the file has been successfully added to the database.
- 3 *Web page Request*: the user enters a web page URL to reverse or reengineer.
- 4 *Identification of the Context (WhoAmI)*: the WhoAmI system detects several attributes from the platform and from the connected user.
- 5 *Save Context Identification*: all the information about the connected platform/user is stored in a knowledge base.
- 6 *Send Reverse Engineered Web Page [if already analysed page]*: the database containing the web pages that have already been reverse engineered is checked. If the page is in this database, the differences between current page and page saved on server is checked, if there is no difference, it is sent back to the user.
- 7 *Ask Web Page [if never analysed page]*: if the UI model of the web page does not exist in the database, a request is sent to a distant server.
- 8 *Send Web Page*: the web page is sent back to the ReversiXML's server
- 9 *Ask Web Page [if the page was not found]*: the file is downloaded from archive.org if it cannot be found on the Web.
- 10 *Send Web Page*: the web page is sent back to the ReversiXML's server
- 11 *Save Page on the Server*: The page is saved into a temporary repository of web pages for two reasons: firstly, it speeds up the process by avoiding downloading again the same page, secondly, it eases the reverse engineering by working on a local file.
- 12 *Choose Configuration File*: By using the context identified during step 4, a configuration file is selected in the configuration knowledge base.
- 13 *Reverse Engineering*: the reverse engineering is performed on the server, following the options contained in the configuration file
- 14 *Send Reverse Engineered Results*: the results are sent back to the user or can directly be used in step 16 to produce a new UI
- 15 *Save Results in Knowledge Base*: the output of ReversiXML is then stored in database for a future reuse.
- 16 *Forward Engineering*: the results of step 13 are used by a forward engineering tool (e.g. teresaXML or GrafiXML) to generate UI code. This transformation is also performed on the server.
- 17 *Send Reverse Engineered Results*: the new UI is sent back to the user.

Figure I-3 ReversiXML 1.2

Appendix J

Boxes amount reduction

As the algorithm for the generation of **boxes** is very exhaustive, i.e. for each HTML tag modifying the layout a **box** is created without checking if it makes sense, the results should be corrected to minimize the number of **boxes**. Some rules are already implemented in ReversiXML but this “**box** correction step” could be enhanced.

For example the next excerpt of HTML code would be translated into too many boxes with the current method:

```
<table><tr><td><table><tr>
<td>content</td></tr><tr><td>something
</td><td></td></tr></table></tr></td></table>
```

This would be reverse engineered in the following manner (in a simplified UsiXML - box V and box H stands for vertical and horizontal boxes):

```
<box V><box H><box H><box V><box H>
<box H><textComponent/></box></box><box H><box H>
<textComponent/>
</box><box H></box></box></box></box></box></box>
```

There are two embedded **tables**, but one of these **tables** is useless as it only contains the other **table**. The second **table** also contains an empty cell which can be removed; it would be possible to represent exactly the same layout with half the number of **boxes**.

Therefore, new rules should be implemented to simplify the structure of the specification. Here are two examples of new simplification rules:

R1: Backward suppression of boxes containing exactly one child that is a leaf itself.

Backward means that the rule should be applied from leaf elements towards the root of the UsiXML specification.

Appendix J

For example:

```
<box v 0>
  <box h 1>
    <box v 2>
      <box h 3>
        <box h 4>
          <element 1>
        </box>
      </box>
    </box>
    <box v 5>
      <box h 6>
        <element 2>
      </box>
    </box>
  </box>...
```

This specification would be transformed into:

```
<box v 0><box h 1><element 1><element 2></box>...
```

Firstly, the horizontal **boxes** h6 and h4 embedding element 1 and 2 are suppressed. As **box** h3 and v5 also satisfy the condition after the first application of the rule (suppression of h4 and h6), they are also removed. Then **box** v2 satisfies also the condition, as element 1 is now its only child. As **box** h1 contains two elements, the rule cannot be applied anymore.

R2: Backward suppression of child **boxes** when the parent and child **boxes** are consecutive (the child **box** is the first child of the parent **box**) and of the same type.

<pre><box h 1> <box v 2> <box v 3> <element 1> <box h 4> <box h 5> <element 2> <element 3> </box> </box> </box> </box> <box v 6> <box v 7> <element 4> <box h 8> <element 5> </box> </box> </box> </box></pre>	<p>Could be transformed in:</p> <pre><box h 1> <box v 2> <element 1> <box h 4> <element 2> <element 3> </box> </box> <box v 6> <element 4> <element 5> </box> </box></pre>
--	--

The horizontal **boxes** h5 and h4 satisfy the condition, so h 5 is removed. Then **boxes** v3 and v7 can be removed, as their parents, respectively the **boxes** v2 and v6, are of the same type. Finally **box** h8 can also be removed as it satisfies the condition of R1 (cf. supra).

Appendix K

External validation material

This appendix contains the document containing the instructions given to the students of the course LINF 2356 for a complete reengineering using Grafixml and ReversiXML. The results of this study have been used in the external validation (section 9.2).

LINF 2356 : INTERFACES HOMME-MACHINE (Prof. J. Vanderdonck)
Assistant : Laurent Bouillon (bouillon@isys.ucl.ac.be)
Travaux pratiques: session n°4

Objectifs

- Effectuer la réingénierie complète d'une interface pour deux plates-formes différentes
- Se familiariser avec l'outil de rétro-ingénierie ReversiXML et l'outil d'ingénierie-avant Grafixml.

Échéance : mai 2005 – envoyer le rapport à bouillon@isys.ucl.ac.be sous format numérique, portant votre nom ainsi que le numéro du TP, ou sous format papier au bureau A.110. Envoyer également les fichiers produits par les deux outils à la même adresse.

Instructions

-Les deux outils, ReversiXML et GrafiXML, sont accessibles depuis leurs sites webs. ReversiXML à partir de <http://www.isys.ucl.ac.be/bchi/research/reversi/RevXMLUI.php>, et GrafiXML depuis l'adresse <http://www.UsiXML.org/index.php?view=page&idpage=10>, puis en cliquant sur le lien <http://www.isys.ucl.ac.be/bchi/members/bmi/grafixml/grafixml.jnlp>.

-Un environnement java est nécessaire pour l'exécution de l'application Grafixml (<http://java.sun.com/j2se/1.4.2/download.html>) ainsi que javaWebStart (téléchargeable à <http://java.sun.com/products/javawebstart/>)

-La documentation sur UsiXML est disponible à <http://www.UsiXML.org/?download=UsiXML-documentation-draft.pdf> (voir l'interface concrète - CUI).

Appendix K

Enoncé textuel

Rédigez un rapport sur la réingénierie complète des pages suivantes:

- <http://www.ucl.ac.be/repertoi.html>
- <http://ultra2.sia.ucl.ac.be:8000/GIPE/SilverStream/Pages/pgRechercheCours.html>
- <http://ultra2.sia.ucl.ac.be:8000/GIPE/SilverStream/Pages/pgRechercheOffre.html>

Sur base des spécifications produites par l'outil ReversiXML, utilisez le modèle concret dans GrafiXML pour pouvoir modifier l'interface de manière à ce qu'elle soit utilisable sur une plate-forme de type pocket pc (plate-forme A). Choisissez également une des deux dernières pages et effectuez la réingénierie complète pour une plate-forme de type gsm (plate-forme B). Pour cet exercice, les modifications de la spécification originale peuvent être liées à deux types de contraintes: les capacités d'affichage (taille, couleurs, ...) et la disponibilité des interacteurs sur une plate-forme donnée. Les caractéristiques à respecter sont reprises en fin de document.

Il est possible d'alléger cette partie du travail en sélectionnant directement les options adéquates dans l'outil ReversiXML (envoyez dans ce cas également les fichiers de configuration).

Le rapport devra contenir les points suivants:

- Extraits de la spécification UsiXML produite par ReversiXML en décrivant ce qu'elles représentent au niveau de l'interface finale.
- Commentaires sur les modifications apportées à ces spécifications, notamment au niveau des options sélectionnées dans ReversiXML et des interacteurs utilisés pour remplacer les interacteurs inexistantes sur une plate-forme donnée.
- Recopies d'écrans des interfaces élaborées avec GrafiXML pour les deux plates-formes demandées.
- Une analyse critique du processus de réingénierie (une page maximum)

Les plate-formes ciblées sont caractérisées par les paramètres suivants (à noter que les contraintes liées à la taille ne doivent pas être respectées à la lettre, mais servent à donner un ordre d'idée de l'espace disponible):

Attribute	Description	Platform A	Platform B
ColorCapable	Whether the device display supports color	Yes	No

Appendix K

Image Capable	Whether the device supports the display of images	Yes	No
ScreenSize	The size of the device's screen in units of pixels	240 x 320	96x65
ScreenSizeChar	Size of the device's screen in units of characters	24X32	9x6
JavaAppletEnabled	Indicates whether the browser supports Java applets.	Yes	No
JavaScriptEnabled	Indicates whether the browser supports JavaScript.	Yes	No
TablesCapable	Indicates whether the browser is capable of displaying HTML tables	Yes	Yes
Category	Category of the device	Pocket PC	Mobile Phone
NumberOfColours	Number of colours the display supports	256	2

Disponibilité des objets d'interaction:

Pour la version PocketPC, tous les interacteurs sont disponibles.

Par contre pour la version gsm, les interacteurs radio button, checkbox, imageComponent, colorPicker, datePicker, filePicker, hourPicker, drawingCanvas et slider sont indisponibles. Remplacez ces objets par d'autres interacteurs afin de conserver toutes les possibilités de l'interface originale.

Appendix L

Glossary

- AIO*: Abstraction Interaction Object, a component of the AUI model, i.e. a specification independent of the computing platform and the modality [Vand93].
- AST*: Abstract syntax tree. Finite, labeled, directed tree, where the internal nodes are labeled by operators, and the leaf nodes represent the operands of the node operators. Thus, the leaves have nullary operators, i.e., variables or constants. It is used in a parser as an intermediate between a parse tree and a data structure, the latter which is often used as a compiler or interpreter's internal representation of a computer program while it is being optimized and from which code generation is performed. The range of all possible such structures is described by the abstract syntax. An AST differs from a parse tree by omitting nodes and edges for syntax rules that do not affect the semantics of the program. The classic example of such an omission is grouping parentheses, since in an AST the grouping of operands is explicit in the tree structure [Wiki].
- AUI*: Abstract User Interface. Specification of a UI independent of the platform and the modality. A canonical expression of the renderings and manipulation of the domain concepts and functions in a way that is independent of the concrete interactors available on the targets. The elements used in the abstract UI are abstractions of existing widgets.
- AWT*: Abstract Windowing Toolkit. The AWT is the user interface toolkit provided as part of the Java language class library.
- CIO*: Concrete interaction object, a component of the CUI model, i.e. a specification independent of the computing platform [Vand93].
- CSS*: Cascading Style Sheet is a language that allows authors and users to

Appendix L

attach style to structure documents. By separating the presentation from the content CSS simplifies web site authoring and maintenance.

- CTT* ConcurTaskTree. CTT is a notation for task model specifications to overcome limitations of notations previously used to design interactive applications. Its main purpose is to be an easy-to-use notation that can support the design of real industrial applications, which usually means applications with medium-large dimensions [Ctt99].
- CUI:* Concrete User Interface. Specification of a UI independent of the platform. This model concretizes an abstract UI into Concrete Interaction Objects (CIOs) so as to define widgets layout and interface navigation. This interface is now composed of existing UI widgets.
- Dialog model:* Model containing all the information relative to the behavior of the UI (i.e. navigation, states...), representing how the user can interact with the components of the UI and the reactions that the UI communicates via these objects.
- DOM:* Document Object Model. Representation of an XML file in an object-oriented fashion. DOM provides an application programming interface to access and modify the content, structure and style of a document [Dom].
- DLL:* Dynamic Link Library. Computer library that implements the concept of dynamic linking. Dynamic linking means that the data in a library is not copied into a new executable or library at compile time, but remains in a separate file on disk. Only a minimal amount of work is done at compile time by the linker, it only records what libraries the executable needs and the index names or numbers. The majority of the work of linking is done at the time the application is loaded or during the execution of the process [Wiki].
- Forward engineering:* Syn. reification. Process of generating a more concrete specification (up to the final code of an information system) based on specification at higher level of abstraction.

Appendix L

<i>Functional Core:</i>	A software can be decomposed into two major parts, the UI and the functional core. The functional core contains all the functionalities, processes... achieved by the software while the UI allows to access to these functionalities.
<i>GUI</i>	Graphical User Interface. GUI is a method of interacting with a computer through a metaphor of direct manipulation of graphical images and widgets in addition to text. GUIs display visual elements such as icons, windows and other gadgets [Wiki].
<i>HTML:</i>	Hyper-Text Markup Language. Markup language designed for the creation of web pages with hypertext and other information to be displayed in a web browser.
<i>Lateral engineering:</i>	Translation process between two UI descriptions at the same level of abstraction.
<i>LOTOS</i>	Language Of Temporal Ordering Specification (LOTOS) is a formal specification language based on temporal ordering used for protocol specification in ISO OSI standards. It was published as ISO 8807 in 1990 and describes the order in which events occur.
<i>Mapping:</i>	Specification of a mechanism for transforming the elements of a model conforming to a particular metamodel into elements of another model that conforms to another (possibly the same) metamodel.
<i>Markup language:</i>	A markup language combines text and extra information about the text. The extra information, for example about the text's structure or presentation, is expressed using markup, which is intermingled with the primary text.
<i>MDA:</i>	MDA defines an approach to IT system specification that separates the specification of system functionality from the specification of the implementation of that functionality on a specific technology platform, and provides a set of guidelines for structuring specifications expressed as models. The MDA approach and the standards that support it allow the same model specifying system functionality to be realized on multiple platforms through auxiliary mapping standards, or through point mappings to specific

Appendix L

platforms, and allow different applications to be integrated by explicitly relating their models, enabling integration and interoperability and supporting system evolution as platform technologies come and go [OMG02].

Modality: Path of communication between the human and the computer, such as voice or touch.

Model: Representation of a system expressed in a given formalism or language, i.e., a set of signs or terms able to encode the modeled domain semantics.

Model-based approach (MBA): Production method of information system based on the representation of the system into various abstract models reflecting its different aspects expressed in a high level formalism. For UI, the models are generally related to the presentation of the UI, the dialog of the UI and optionally other models such as the user model, platform model, domain model, task model, application model or context model.

Presentation model: Model containing all the information relative to the static appearance of the user interface (i.e. composition, layout, colors, style...).

QTK QtK is a advanced tool for creation of UI which belongs to the Mozart Programming System, an advanced platform of development of distributed and intelligent applications. QtK is a C++ wrapper for Gtk+ (a multi-platform toolkit for creating GUI) that is Qt (library of widgets and basic functions) compatible.

Reengineering: Examination and the alteration of a subject system to reconstitute it in a new form and the subsequent implementation of the new form. Combination of reverse and forward engineering.

Retargeting: Reverse engineering achieved for a particular context of use, by applying reverse engineering rules designed for the targetted context [Boui02b].

Reverse engineering Syn. abstraction. Process of analyzing a subject system to identify the system's components and their interrelationships and create

Appendix L

- (RE):* representations of the system in another form or at a higher level of abstraction [Chik90]. More generally, it is the process of generating a more abstract specification based on specification at a lower level of abstraction.
- SGML* The Standard Generalized Markup Language is an ISO standard meta-language for the description of marked-up electronic text that defines device/system independent methods for representing texts in electronic form. SGML is a meta-language, a way of formally describing a markup language [Lope05]
- Sobel Operator:* The Sobel operator is an operator used in image processing, particularly within edge detection algorithms. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel operator is either the corresponding gradient vector or the norm of this vector [Wiki].
- SVG:* Scalable Vector Graphics (SVG) is an XML markup language for describing two-dimensional vector graphics, both static and animated (either declarative or scripted).
- Tidy:* HTML code cleaning tool developed by D.Raggett [tidy], allowing to remove useless tags, to close “open” tags (i.e. without a closing markup), put attributes values into quotation marks etc. By choosing a particular set of corrections, it is possible to obtain HTML code respecting the XML syntax.
- Transcoding:* Process of translation of a final UI into a final UI for another context of use without abstraction step, in opposition to reverse engineering.
- UAProf:* User Agent Profile. Its specification is concerned with capturing capability and preference information for wireless devices. This information can be used by content providers to produce content in an appropriate format for the specific device [Wiki].
- UI:* User Interface. The user interface is the aggregate of means by which users interact with a particular machine, device, computer program or other complex tool. The user interface provides means

Appendix L

of input and output [Wiki].

- UIML:* User Interface Markup Language. Meta-language that can describe any user interface in a manner that is device-independent and user interface metaphor independent. UIML can describe user interfaces that are popular today -- for traditional desktop, web, mobile, embedded, and voice applications. UIML can also describe user interface for custom devices or devices that are invented in the future [Uiml04].
- UML:* The Unified Modeling Language is a non-proprietary, object modeling and specification language used in software engineering. UML includes a standardized graphical notation that may be used to create an abstract model of a system: the UML model. While UML was designed to specify, visualize, construct, and document software-intensive systems, UML is not restricted to modeling software. UML has its strengths at higher, more architectural levels and has been used for modeling hardware (engineering systems) and is commonly used for business process modeling, systems engineering modeling, and representing organizational structure. This language is the merging of the three methods that had the most influence in the nineties (OMT, Booch and OOSE).
- USIXML:* User interface description language developed at UCL. This XML-based language allows specifying context-sensitive UI description according to the four level of abstraction of the Cameleon reference framework (see chapter 6).
- VoiceXML:* Markup language designed for vocal interfaces.
- W3C:* The World Wide Web Consortium (W3C) is an international consortium where member organizations, a full-time staff, and the public, work together to develop standards for the World Wide Web. W3C's mission is: "To lead the World Wide Web to its full potential by developing protocols and guidelines that ensure long-term growth for the Web". W3C also engages in education and outreach, develops software, and serves as an open forum for discussion about the Web. The Consortium is headed by Tim Berners-Lee, the original creator of the World Wide Web and

Appendix L

primary author of the URL (Uniform Resource Locator), HTTP (HyperText Transfer Protocol) and HTML (HyperText Markup Language) specifications, the principal technologies that form the basis of the Web [Wiki].

- WAP:* Wireless Application Protocol. It is an open international standard for applications that use wireless communication. WAP was designed to provide services equivalent to a web browser with some mobile-specific additions.
- Widget:* Component of a graphical user interface.
- WML:* Wireless Markup Language. Markup language designed for mobile phones.
- XIML:* eXtensible user Interface Markup language developed at Redwhale software (<http://www.redwhale.com>)[Ximl]. This XML-based language allows defining the UI thanks to various models (see chapter 5), and has the particularity the designer can parameterize and extend the language (i.e. specify her own definitions of interaction objects, relations etc...).
- XML* The eXtensible Markup Language is a subset of SGML that has been designed for ease of implementation and interoperability with both SGML and HTML, allowing easy interchange of documents on the web. With XML, markup languages to define the content can be designed [Lope05].
- XSLT:* eXtensible Stylesheet Language Transformations. Language for transforming XML documents into other XML documents. XSLT is designed for use as part of XSL, which is a stylesheet language for XML. In addition to XSLT, XSL includes an XML vocabulary for specifying formatting. XSL specifies the styling of an XML document by using XSLT to describe how the document is transformed into another XML document that uses the formatting vocabulary [Xslt99].