Data Mapping
by Using
Business Form Copying Metaphor

Yongqiang Li

The University of Auckland, New Zealand

Acknowledgments

I would like to extend my heartfelt appreciation to John Grundy and Robert Amor for their insight and guidance throughout my master work. I am very grateful for the many hours that they spent discussing and critiquing my work.

I am especially grateful to John Grundy for his patient, help and understanding during my study.

Especially, I would like to express my gratitude to my family for their support and encouragement.

Abstract

Data mapping is a necessary process in the integration of applications, and for Web Services. The present approach to data mapping is either completely hand-coded or assisted by some mapping tools used by programmers. The development of a data mapping specification is error-prone, costly and time consuming. However business analysts have a better understanding of context of data mapping. A mapping tool, which can support the business analysts to specify the mapping and generate the mapping specification implementation, can help to avoid problems.

We designed and prototyped a mapping tool by using a business form copying metaphor, a spreadsheet-styled end-user programming environment, allowing the business analysts to define mapping specifications by visual and direct manipulation. In the mapping tool, a form-based metaphor gives a concrete representation for high-level abstracted source and target business data models for source and target data schemas; the users can get immediate feedback after the mapping specification is finished for each field or all fields; a code generator can generate different mapping specification implementations. An initial evaluation on the prototype shows that this tool have good support for an end user.

Table of Contents

Table of Figures

Table of Tables

Chapter 1 Introduction

This chapter gives the motivation and objectives for this research. This is followed by a summary of the approaches taken to satisfy these objectives. The final section provides an overview of this thesis.

1.1 Motivation

Data transformation has been widely used in building automated systems and integrating heterogeneous systems [Morgenthal 2001] [Swatman 1994] [Amor 1999]. These heterogeneous systems range from the newest systems using an object-oriented approach with different technologies, such as J2EE and .Net, CORBA and DCOM, to legacy systems using a data-oriented approach. In these systems, the data may be exchanged through various technologies, such as copy-and-paste, distributed object APIs (CORBA, DCOM, .Net), EDI, or web services technologies, such as SOAP, RPC-XML, etc. The exchanged data can be objects, serialized objects, EDI messages, XML messages, SOAP messages, and custom data formats. The target data may be in a different type or format from the source data. In order to make the data from the source system understood by the target system, transformation from the source data to the target data is needed. For example, in building a system by using web services technologies, a XML message from a source needs to be transformed to another XML message with different data structure [Capeclear.com 2001]; in a health industry, a patient treatment EDI message encoded in the UB92 protocol from a health provider must be translated to another treatment EDI message encoded in the 837a protocol which can be only accepted by a funder's system [Grundy 2001].

Current developments in data transformation system are programmer-centric. Traditionally the transformation system is directly hand-coded by a programmer using a general-purpose programming language, such as C++ or Java. In order to make the system be able to quickly react on changes of the system, which are caused by the

external environment of system, for example, changes of business partners, upgrading systems, many efforts have been made to improve the development of the data transformation system. These include: separating mapping specifications, which are defined based on the source and target schemas, from the processing unit (see Figure 1.1) to improve the flexibility and maintainability of transformation system; developing data mapping specifications by using domain specified script languages to avoid complexity of conventional programming languages and using visual mapping tools to give the programmer the power of direct-manipulation to define mapping specifications and generate mapping specification implementations. A combination of all these efforts greatly improves the development of the transformation system but these efforts are only aim at professional programmers.

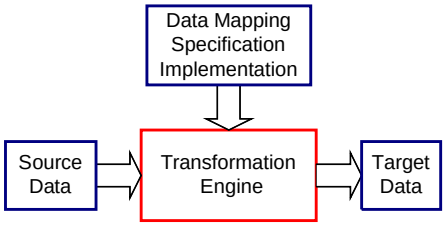


Figure 1.1 Architecture of Transformation system

In this research, we investigate a mapping tool to help a business analyst, a non-programmer, to develop the mapping specifications to further improve the development process of the mapping specifications. This is based on following reasons:

A business analyst is the best person to define the mapping specifications. The business analyst is a person who is responsible for understanding and developing business processes and procedures at the first stage of software development. They have the knowledge of what one business's data structure and semantics mean and how this can be mapped onto another business's data. In the beginning of current development of mapping specifications (see Figure 1.2), it is the business analyst that produce mapping specifications on a business data model (see Figure 1.2(1)) and describe the business data mapping to programmers (see Figure 1.2(2,3)) to make them produce the lower level data mapping specifications implementation, which involves complex computer data structures and programming knowledge.

A mapping tool supporting the business analyst to produce the mapping specification implementation can dramatically reduce cost of development of transformation system. In the mapping tool, the business analyst directly specifies mapping specifications through a friendly user interface (see Figure 1.3(1,2)), and then the tool takes the mapping specification to generate mapping specification implementations (see Figure 1.3(3)). In the current development of transformation system, after the business analyst produces the business data mapping specifications, the programmer either directly codify the data mapping specifications in an implementation language (see Figure 1.2(4)), or is aided by a mapping tool to textually or semi-visually to define the mapping specification in a domain-specified script mapping language (see Figure 1.2(5,6)) and then lets the tool generate the mapping specification implementation (see Figure 1.2(7)). In most of current mapping tools, data schemas are normally rendered in a graph [Ceri 1999], or tree format [Grundy 2001][Sonic 2003] [Capeclear 2001], or an UML model [Amor 1999] corresponding to the specific types of the data schema. This requires the users of the tools still to have detailed knowledge of data modeling and the mapping language to define the mapping specification. Although some transformation tools used a form-based presentation [Erwig 2002], they don't generate mapping specification implementations to support the data transformation system architecture we described in the previous section. So these mapping tools cannot totally avoid a need of programmer to let the business analyst to directly define the mapping specification. The further need to have a programmer involved rather than letting the business analyst to directly produce the data mapping specification implementations makes the cost high of the development of the mapping specification.

The proposed new tool can dramatically reduce errors of development of transformation system. During the current development process, errors in the low level mapping specification may be easily made by either poor communication between the business analyst and the programmer or mistakes made by the programmer himself. In our new development process, there is no gap between the business analyst and the programmer and all above errors are avoided.

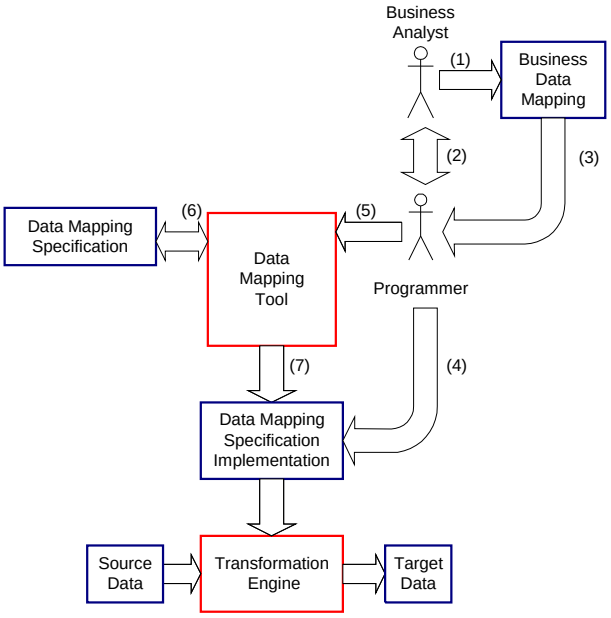


Figure 1.2 Mapping specification development process by using current mapping tools

The new tool can dramatically shorten the lifecycle of development of transformation system. During the current development process, it needs the whole software engineering process, which will last a long time, to cope with every change to the system environment. This includes business data mapping done by the business analyst, producing mapping specification implementations done by the programmer with a help of mapping tools, testing, finding bugs and fixing them. The new mapping tool can eliminate the most of the later stages and make the development much more efficient.

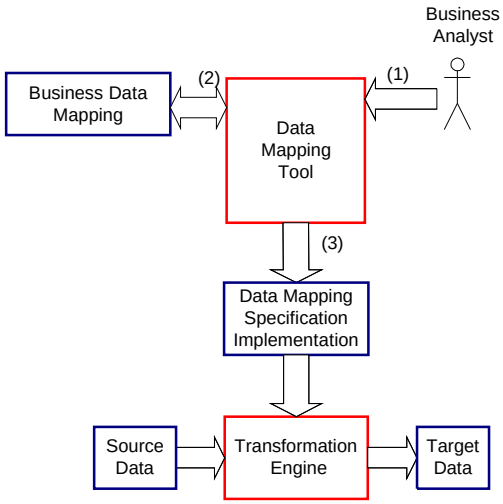


Figure 1.3 Mapping specification development process by using our mapping tool

Existed techniques and guidelines on end-user programming may enable the dream of such a mapping tool used by the business analyst come true. Current techniques and researches on end-user programming [Goodell 1998], such as visual programming [Burtnett 1999], programming by demonstration [Cypher 1993] and natural programming [Goodell 1998], utilize a visual approach to provide an end-user, a non-programmer, a concrete, direct-manipulated programming environment on their problem domain, to speak their own language, to make use of their existed knowledge, to fit to their cognitive models to enable them to write program without the professional programmer background, for example, a spreadsheet program [Nardi 1993].

1.2 Objectives of the Research

The objectives of our research are to develop a mapping tool with visualization of underlying data model using a high level of abstraction—concrete business forms, and a visual mapping specification environment, which can be easily learned by business analysts, non-programmers, to enable them to easily and correctly define their mapping specifications, The tool can then generate mapping specification implementations automatically for the transformation system.

In this research, we examine using a business form metaphor to layout complex source and target data schemas to meaningful business forms. In these forms, elements can be rearranged and sample data can be imported to make these forms look like true business forms and fit the user's mental model. Based on the forms, we examine using a business form copying metaphor to provide an end-user mapping specification environment, to which various techniques of end-user programming, such as the spreadsheet-styled, programming by demonstrations and nature programming, and a type system are applied to enable the user to mimic business form copying to define mapping specifications, and finally generate mapping specification implementations. Following these thoughts, we developed a prototype of the mapping tool in our research. Now it can take source and target XML DTDs and XML instances to visualize them to forms, enable an end user to specify mapping specifications visually and generate XSLT code.

1.3 Methodology

In order to achieve the above objectives, the prototyping process in software engineering [Floyd 1984] and the usability engineering model [Nielsen 1992] were combined and applied to our research. According to these methodologies, the following steps are conducted in this research:

1. Research features and characteristics of existing mapping tools, including examining functionality and user interfaces of these systems, and other related research fields, such as end- user programming and software usability, which can guide the development of our mapping tool.
2. Study main user requirements and possible architectures for the mapping tool, design end-user mapping specification environment with a business form copy metaphor, including an user interface design and an object-oriented design
3. Develop a prototype according to the above design
4. Carry out an initial usability evaluation on the prototype
5. Provide a set of suggestions for improving the usability of the prototype based on the initial evaluation.

1.4 Overview of the Thesis

The following is an overview of remaining chapters in this thesis, briefly summarizing the topics described in each chapter:

Chapter 2 describes the related works that have been done on the data mapping tool, end user programming, and software usability fields.

Chapter 3 starts from a business scenario of a manual system and an automatic system to give the motivation of using a business form copying metaphor for our mapping tool and then gives main requirements of our mapping tool.

Chapter 4 gives an object-oriented design and user interface design of our mapping tool according to the system architecture based on requirements of our mapping tool.

Chapter 5 gives details of java implementation prototype for XML-to-XML data mapping and XSLT code generation.

Chapter 6 discusses the usability of our prototype using a notational evaluation according to cognitive dimensions framework.

Chapter 7 concludes contributions of this research and gives further improvements of our mapping tool for future research.

Chapter 2 Related Work

This chapter introduces some related work to this research, and describes features and characteristics of some existing mapping tools, end-user visual programming environments and user interface design techniques.

2.1 Introduction

There are three main fields related to our research. They are data mapping/data transformation, end user programming, and software usability.

The data mapping/data transformation area gives what exactly have been done in our problem domain, what tools existed and how they work for assisting the users to define data mapping specification. So we can know the context of our research, find problems from them and improve them.

The users of our mapping tool are focused on business analysts who may have not any programming knowledge. Defining mapping specification by them actually falls into the end user programming field. Through investigating this field in a broad of problem domains, we get what end user programming languages, techniques and tools exist to make the programming more easily for professional programmers and end-users to learn and use, and how they achieve it. Then we can analyze them and apply suitable end user programming language techniques to designing our mapping specification environment to achieve high usability.

The software usability study provides us the guidelines for user interface design in design process of our tool, and techniques and methods for usability evaluation after implementation of user interface.

2.2 Mapping Tools

Current data mapping/data transformation approaches are programmer-centric. They include program-based data mapping, script-based data mapping and semi-visual and visual data mapping.

In the program-based data mapping, the programmers manually codify the mapping specifications using a conventional programming language, such as C, or C++, or Java. It takes considerable efforts of expert programmers on design, implementation and testing.

The introduction of script-based approach dramatically reduces the load of programming mapping specifications, because in this approach the programmers manually codify the mapping specifications using a domain-specific script language, which is much simpler than the conventional programming language, for example, the XSLT for XML transformation [W3C 1999 XSLT].

Many visual mapping/transformation tools were developed to release the programmer's burden in some specific domains. Because the above text-based script-based approach is still difficult to use. These tools visualize the data model or schema model to a graph-based presentation, such as XML-GL [Ceri 1993], or a tree-based presentation, such as Orion Symphonia System [Grundy 2001], Sonic Stylus Studio [Sonic 2003], or a table-liked presentation, such as Data Junction Integration Map Designer [Data 2003], or a form-based presentation, such as Xing [Erwig 2002], or UML class diagram, such as [Amor 1999], or other presentation, such as VXT [Emmanuel 2001] to give a clear and direct view of the data model, and allow user to direct manipulate the visual components and semi-visually or visually to define the mapping specifications. Some of these tools are described in the following sections.

2.2.1 Xing

Xing [Erwig 2002] is a visual language for querying and transforming XML data. The language is based on a visual document metaphor (see Figure 2.1(1)) and the notion of document and rules, and targets on the end users. Document patterns can be directly used as query patterns (see left-hand side of Figure 2.1(2)(3)). Document rules can be used to restructure query results (see Figure 2.1(4,5)). The language combines a

dynamic form-based interface for defining queries and transformation rules with pattern matching capabilities.

The form-based query interface gives it ability for the end-users to easily understand and use. But its XML transformation capability is limited to restructure query results, so it doesn't seem to support large and complex XML-to-XML transformation. And also it doesn't support using XML DTD to create mapping specification and generate the mapping specification implementation for the generic transformation model.

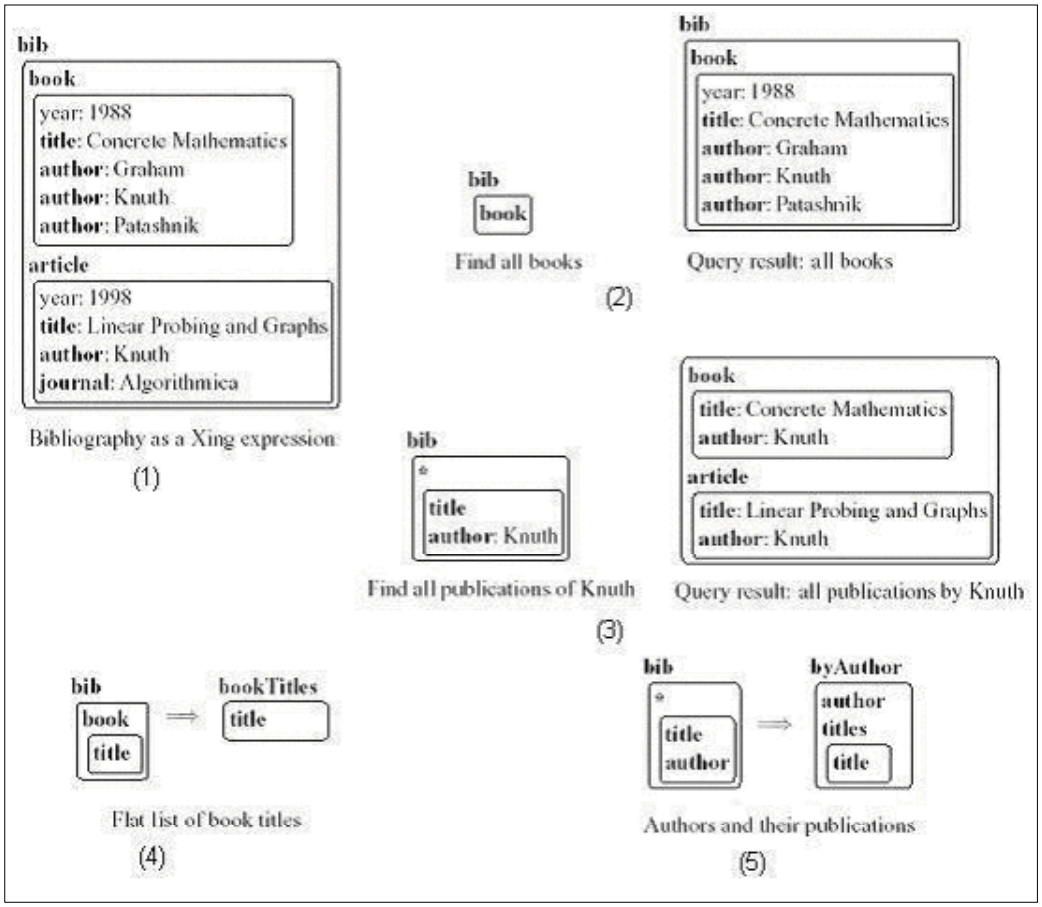


Figure 2.1 Xing—A visual language for XML query and reconstruction. Elements in this figure are extracted from [Erwig 2002]

2.2.2 VXT

This application provides a visual language/environment to *programmers* who want to specify XML transformations [Emmanuel 2001] (See Figure 2.2). It displays XML documents and/or their DTD in a treemap-like [Johnson 1991] presentation, from which it is possible to visually construct selection and extraction rules similar to templates in

XSLT. Mapping specifications can be defined visually and then XSLT codes can be generated according to the mapping specifications. To each rule is associated a constructor, also specified visually, that tells what should be output when the rule matches a node in the source document.

The visualization of XML DTD and visual mapping specification environment in VXT make the user of the tool to directly manipulate the visual element to define the simple and complex mapping specification. But the abstraction of the visual presentation for XML DTD and the underlying XSLT transformation model in VXT is in the same level as that of textual XML DTD and XSLT, i.e. visual notations almost one-to-one mapping to XML DTD and textual XSLT. So it requires the user have knowledge of underlying XML, XML DTD and XSLT.

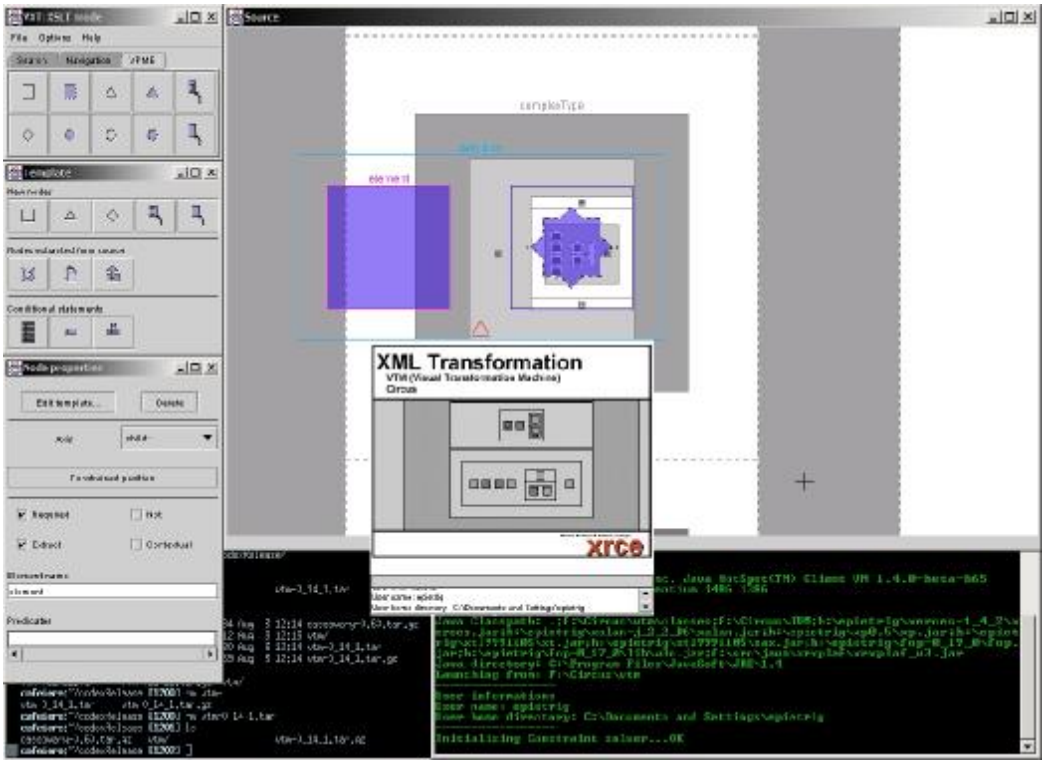


Figure 2.2 VXT – Visual language for XML transformation. This figure comes from http://www.xrce.xerox.com/competencies/contextual-computing/vtm/apps.html#vxtApp

2.2.4 Orion Symphonia System

The Orion Symphonia system is a commercial EDI and XML transformation system. A mapper inside the system separates transport-level information from the source and

target schema and renders them to a tree-like data structure (see Figure 2.3). The users can use drag-and-drop to wire the source and target segment, record and field to define the mapping specification. But formulae and functions need to be input in text in the text field at the bottom of the window.

The tree-based representation is not easy for the business analyst to understand the semantic mean of data because it's different from the business analyst's mental model of data. Textually building the formulae and functions makes it difficult for the end-user to use because it need the user to know the syntax of the text language.

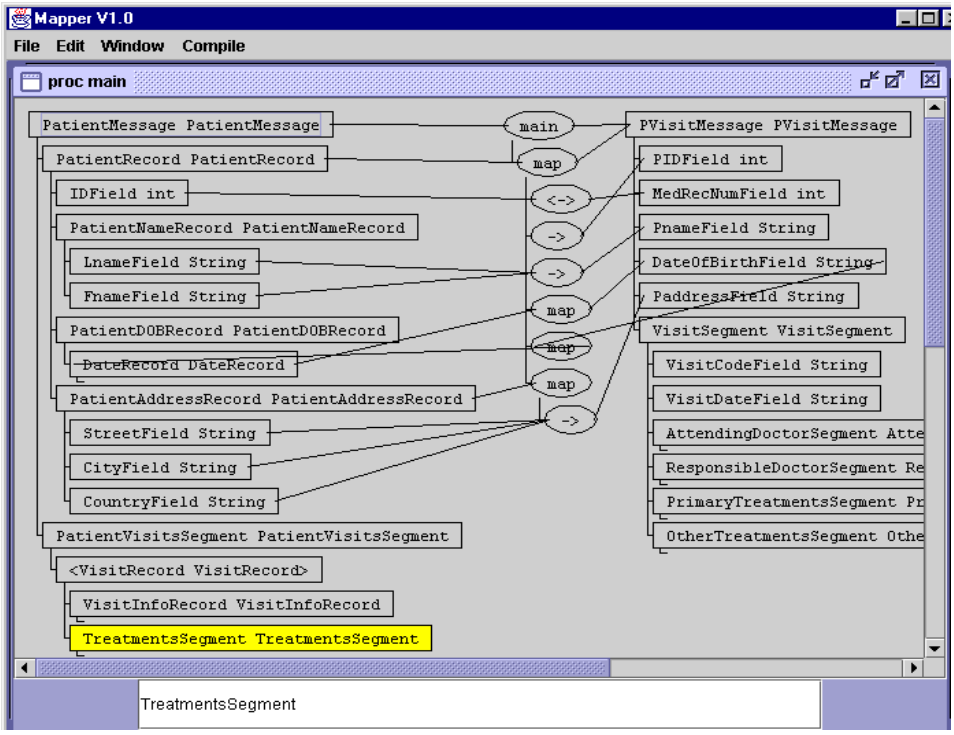


Figure 2.3 The Mapper for Orion Symphonia System. This figure originates from [Grundy 2001]

2.2.5 Sonic Stylus Studio

The Sonic Stylus Studio is a commercial XML transformation system. In its mapper, like Orion Symphonia system, the source and target data or schemas are rendered to a tree-like presentation. The user can use drag-and-drop to connect source and target fields to visually define simple copy relations. The user can visually insert XSLT build-

in functions and connect source and target fields to the functions arguments to define the merging and splitting operations (see Figure 2.4). But the user needs to textually define their own functions in java language and complex mappings need to be defined directly in textual XSLT code. This makes the tools also not suitable for the business analyst to use.

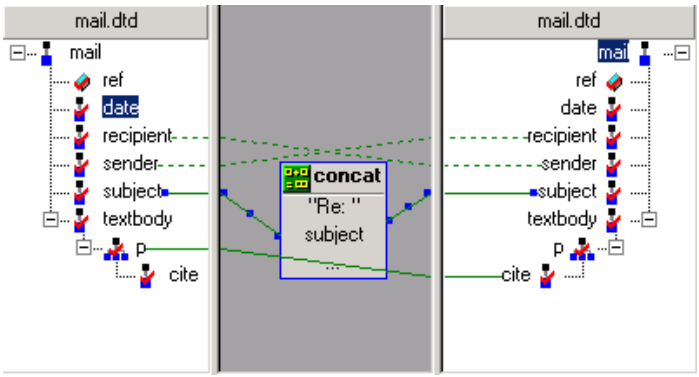


Figure 2.4 Sonic130 Stylus Studio

2.2.6 Data Junction

Data Junction supports a variety of data transformations, such as database, XML, etc. The source and target schemas are rendered to both a tree-like and table-based visualization (see Figure 2.5). The tree-like visualization represents the data relation and the table is for listing data element for the user to be ready for formula definition and browse. The drawback of the visualization is that it makes the user often switch between the two views to get the context of the data elements. Again the tree view is not easy for the end-user to understand the data context.

The mapping specification for a target field is expressed as a formula or a set of procedures, which contain the source field(s). The simple copy relation between the source and target field can be defined using drag-and-drop operation between cells of source and target table. Other mapping specifications can be built through an expression builder, a visual programming environment, by clicking on operator icons, statement icons and tree nodes to form text code, and sometimes inserting some text code in proper position among existed text code. Although this prevents some syntax errors caused when users just textually type the expressions, it still needs the user having knowledge of syntax of the programming language.

[XML] All Source Fields

	Source Field	Source Record	Type	Size
1	[mail!textbody]p	[mail]textbody	Record	0
2	[mail]textbody	mail	Record	0
3	cite	[mail!textbody]p	Character	16
4	date	date	Character	16
5	date	mail	Record	0
6	p	[mail!textbody]p	Character	16
7	recipient	mail	Record	0
8	recipient	recipient	Character	16
9	ref	mail	Attribute	16
10	sender	mail	Record	0
11	sender	sender	Character	16
12	subject	mail	Record	0
13	subject	subject	Character	16

[XML] All Target Fields

	Target Field Name	Target Record	Target Field Expression
1	[mail!textbody]p	[mail]textbody	=
2	[mail]textbody	mail	=
3	cite	[mail!textbody]p	=For each index in Records("[mail]textbody")
4	date	date	=Date()
5	date	mail	=
6	p	[mail!textbody]p	=
7	recipient	mail	=
8	recipient	recipient	=Records("sender").Fields("sender")
9	ref	mail	=
10	sender	mail	=
11	sender	sender	=Records("mail").Fields("recipient")
12	subject	mail	=
13	subject	subject	="Re: " & Records("subject").Fields("subject

Figure 2.5 Mapping tables in Data Junction

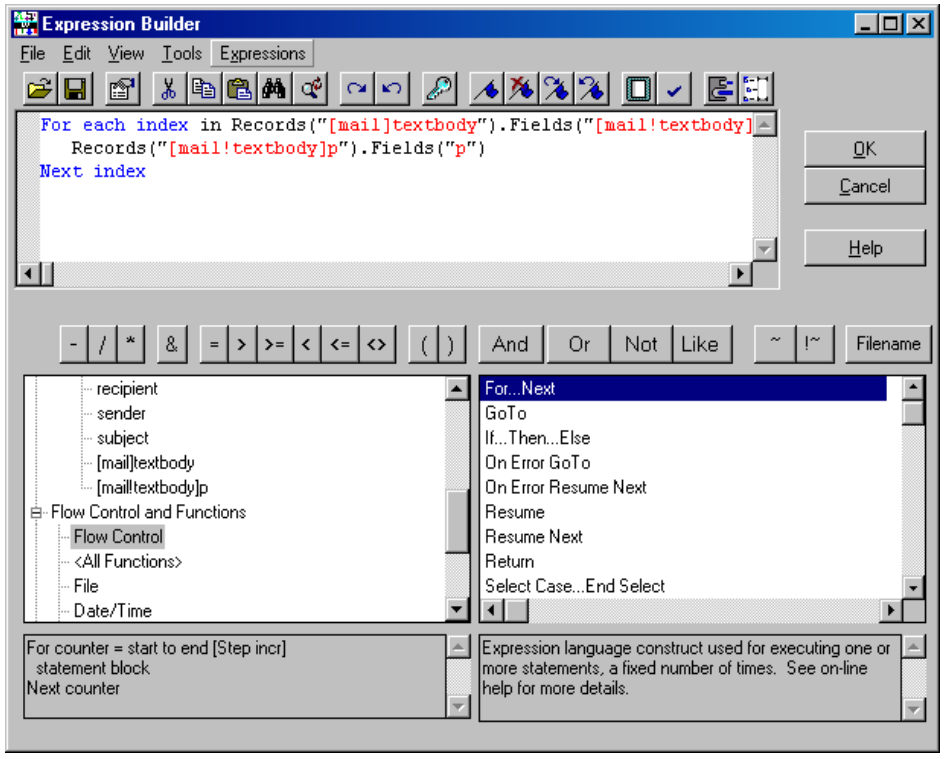


Figure 2.6 Expression Builder in Data Junction

2.3 End-user programming

“End-user Programming (EUP) is that when end-users, who have not necessarily been taught how to write code in conventional programming languages, write computer programs, for example, spreadsheet users who write formulas and macros” [Cypher 1993]. There are a wide variety of end-user programming techniques for different professions, tasks, and users. Many fall into the categories below [Goodell 1998].

- Application-specific Languages
- Programming By Demonstration
- Visual Programming
- Natural Programming

2.3.1 Application-specific Languages

An application-specific language is a script language, for example, JavaScript, VBScript, Unix Shell Script, which is a small, simple programming language whose vocabulary is specifically tailored to the objects and actions of a particular application domain and targets on more serious end users, such as Web page authors and network administrators [Goodell 1998]. The hope is that such a language will not be too difficult for end users to learn. “The basic failing of scripting is that it is still programming. That is, 1) users have to learn the arcane syntax and vocabulary conventions of the language, and 2) they have to learn the standard computer science concepts of variables, loops and conditionals.”[Cypher 1993]

2.3.2 Programming by Demonstration

Programming by demonstration is a technique for teaching the computer new behaviour by demonstrating actions on concrete examples. In this approach, normally a visual direct-manipulated metaphor is provided for a user to interact with. The system records the interactions and writes a program that corresponds to the user's actions and then generalizes the program to make it be able to work with other similar examples [Cypher 1993].

Programming by demonstration is largely used for *automating* repetitive activities. These activities include iterative activities, such as renumbering a long list when a new entry is inserted in the middle, and periodic activities, such as backing up recently changed files [Cypher 1993]. The key to success of programming by demonstration is using right example to make the correct inferences and generalize the program [Cypher 1993].

2.3.3 Visual Programming

Visual programming uses multi-dimension to convey semantics. It uses concrete instances, direct-manipulation, explicit notations and immediate visual feedback to make programming more accessible to some particular audience, and to improve the correctness and speed with which people perform programming tasks [Burtnett 1999].

Forms/3 [Hays 1995] is a general purpose, declarative, spreadsheet-based visual programming language. Its goal is to provide computational and expressive power in a language featuring a simple, concrete programming style with immediate feedback.

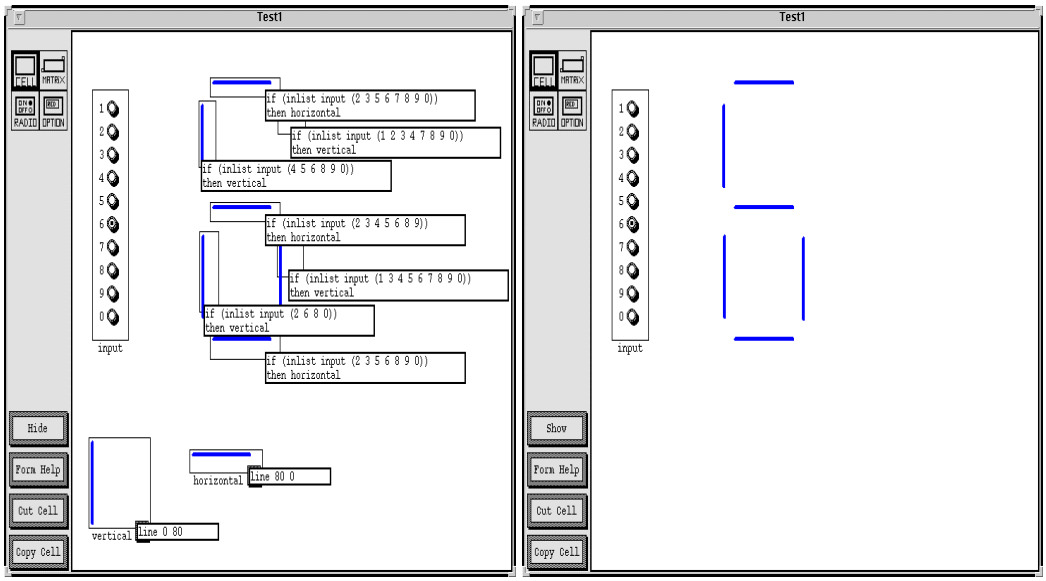


Figure 2.7 Form/3-a spreadsheet-based visual programming language. These two figures originates from [Wilcox]

Programming in Forms/3 follows the spreadsheet paradigm (see Figure 2.7). The programmer uses direct manipulation to place cells on forms, and then defines a formula for each cell. Such a formula may include constants, references to other cells,

or references to the cell's own value at a previous moment in time. Cells are referenced by clicking on them. A program's calculations are determined by these formulas.

The most successful end-user programming system to date is the spreadsheet, due in part to its familiar and effective metaphor of financial tables [Nardi 1993].

But it also has drawbacks. In the early spreadsheet program, there is no explicit connection between a cell and the cells its formula refers to. This causes a hidden-dependence problem. It's very risk to alter a spreadsheet cell. Modern spreadsheets have improved this by containing tools to analyse dependencies [Blackwell 2002]. Spreadsheets may often contain faults [Panko 1998]. The reason of the problem is spreadsheet programmers seem to have overconfidence in the correctness of their spreadsheets [Brown 1987] [Wilcox 1997]. "A possible cause of this overconfidence may be related to… that too much feedback and responsiveness, as featured in the immediate visual feedback of values in spreadsheet languages, can actually interfere with people's problem-solving ability in solving puzzles [Gilmore 1995] [Svendsen 1991], a task with much in common with programming." [Rothermel 2000] A "What You See Is What You Test"(WYSIWYT) methodology was introduced to tackle this problem [Rothermel1 998] and positive results were got [Rothermel 2000].

2.3.4 Natural Programming

Natural programming [Myers 1998] is a project leaded by Brad A. Myers, Human-Computer Interaction Institute, School of Computer Science, Carnegie Mellon University. The researchers have been trying to develop a more natural programming language, which is different from conventional programming languages, and provides the users much easier and more natural way to learn and use it to develop programs.

The following are some quoted results for guiding the design of a new programming system surveyed and observed by [Pane 1996] [Myer 1998]. They are very helpful for guiding our mapping specification environment development:

- One way to ease the entry into programming is to capitalize on the beginner's knowledge about the world. Many languages are based on a metaphor, which should be drawn from a concrete real-world system that is familiar to the user audience [Smith 1994].

- When they are stumped, beginners will attempt to transfer knowledge from other domains even if they are not appropriate [Hoc 1990]. This is a problem when the language uses words and symbols in ways that are different from English or math. For example, "AND" is often read to mean "THEN" as in: "We went to the store **and** bought milk," whereas in computers, AND is always used between two things that must both be true at the same time. People often use "AND" when a computer would require the use of "OR," as in: "All people whose names begin with 'A' **and** 'B' should be in the first line." Another problematic example is that "a = a+3" makes no sense if read as in mathematics. These kinds of features should be avoided in the new language.
- A very low-level language with many simple primitives requires the user to synthesize higher-level operations. This is one of the great difficulties in programming [Lewis 1987]. When there are many different choices, more planning is required, and this increases the likelihood of backtracking and revision, which slows the programmer [Gray 1987]. Therefore, the language should provide high-level operations.
- The object-oriented style seems to be harder to learn for novice programmers, and a full inheritance hierarchy has been shown to be too complex for novices, but a fixed two-level inheritance hierarchy is understandable [Pausch 1992].
- Much of the control was expressed in an "event language" (also called the "production language") style, with rules to control behaviours. This result is already reflected in some of today's end-user programming languages. The event-based style used by Visual Basic, Lingo for Director, and HyperTalk for HyperCard, is a form of rule-based style, since the code is of the form "if this event happens, then execute this code."
- The students preferred to express the general case first, and then later modify it with exceptions. For example, "When you encounter a ghost, the ghost should kill you. But if you get a little pill you can eat them." This is in contrast to conventional languages that generally require the conditional to be set up in advance using "ANDs," "NOTs" and "ORs," forcing the user to think about all the cases first, and resulting in a complicated Boolean expression.

- Iterations were usually expressed implicitly, by operating on sets of objects. For example, "When PacMan eats all of the yellow balls he goes to the next level." This is instead of using any form of iteration or explicit counting, as would be required in most programming languages.
- Participants did not construct complex data structures and traverse them, but instead performed content-based queries to obtain the necessary data when needed. For example, instead of maintaining a list of monsters and iterating through the list checking the color of each item, they would say “all of the blue monsters.”
- Participants often drew pictures to sketch out the layout of the program, but resorted to text to describe actions and behaviors.

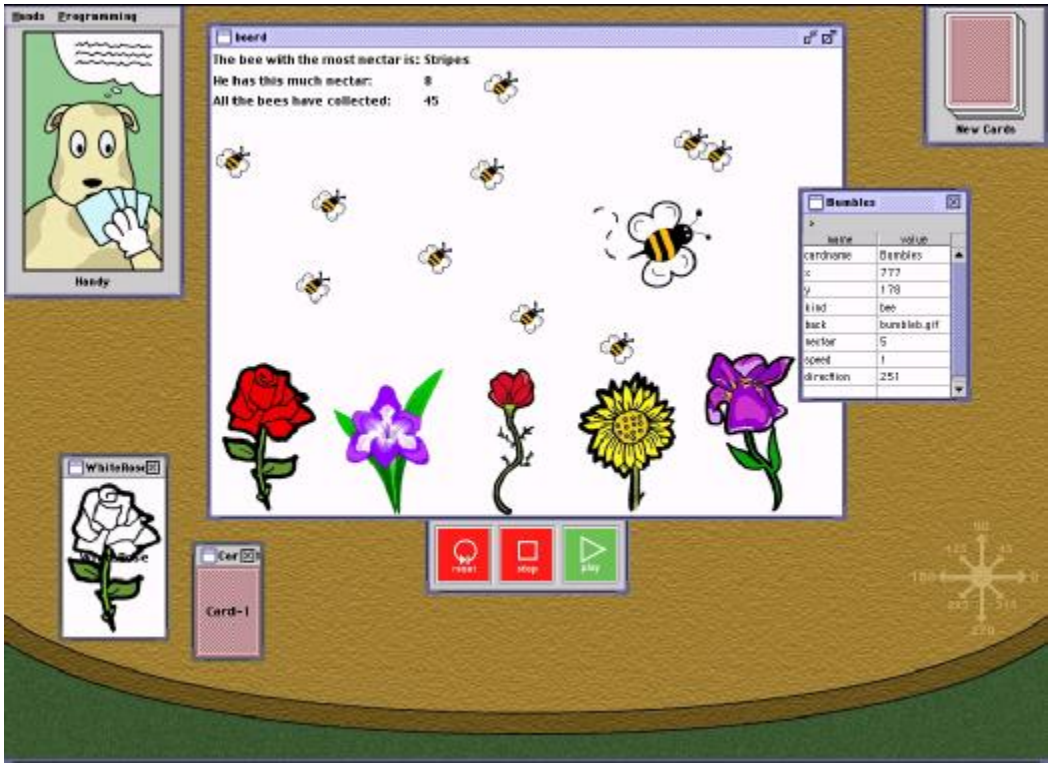


Figure 2.8 HANDS programming environment. This figure originates from [Pane 2002]

HANDS [Pane 2002] is a natural programming environment for *children* (see Figure 8) based on the above observations. It represents the computation as a metaphor in which a character sits at a table and manipulates cards that hold the program's data and are familiar, concrete, persistent, and visible. This familiar model avoids the need for

beginners to learn the traditional *von Neumann machine* model of computation. Cards can expand to accommodate any size of data, storage is always initialized, and types are enforced only when necessary, such as when performing arithmetic. It uses an event-based style of programming, and provides queries and aggregate operators to allow more concise high-level expressions for tasks that require the assembly of many primitives in other languages. HANDS directly supports queries for content-based data retrieval. HANDS uniformly permits all operations that can be performed on single objects to also be performed on lists of objects, including the lists returned by queries. Study shows that features of these have a significant positive effect on usability [Pane 2002].

2.4 Software Usability

There are some general principles and heuristics [Nielsen 1994] in the field of Human Computer Interaction. They can be applied to programming system design. [Pane 2002a] gives these terms very good explanation shown as following:

- simple and natural dialog – user interfaces should be simplified, and should match the user's task in as natural a way as possible, such that the mapping between computer concepts and user concepts becomes straightforward.
- speak the user's language – the terminology in user interfaces should be based on the user's language, instead of using system-oriented terms or attaching non-standard meanings to familiar words.
- minimize user memory load – the system should not force the users to memorize too many things.
- consistency – the same command or action should always have the same effect.
- feedback – the system should continuously inform the user about what it is doing and how it is interpreting the user's input.
- clearly marked exits – the system should offer the user an easy way out of as many situations as possible, including ways to undo.
- shortcuts – the system should make it possible for experienced users to perform frequently used operations quickly.

- good error messages – the system should report errors politely in clear language, avoid obscure codes, use precise rather than vague or general explanations, and include constructive help for solving the problem.
- prevent errors – where possible, the user interface should be structured to avoid error situations.
- help and documentation – the help system and documentation should provide a quick way for users to find task-specific information when they are having a problem.

Cognitive Dimensions of Notations framework [Green 1996] gives useful evaluation criteria when we design and evaluate programming systems. [Pane 2002a] gives these dimensions very good explanation shown as following:

- viscosity – the system should not resist change; it should not require many user actions to accomplish one small goal.
- visibility – the information needed by the programmer at any particular time should be visible or very easy to access.
- premature commitment – the system should not force the user to go about the job in a particular order, or make a decision before the needed information is available.
- hidden dependencies – important links between entities should be visible.
- role expressiveness – the purpose of an entity should be readily apparent.
- error proneness – the notation should protect against slips and errors.
- closeness of mapping – the system's operations should closely match the way users think about problem solutions.
- secondary notation – the system should allow the programmer to communicate additional information with comments, typography, layout, etc.
- progressive evaluation – the system should permit users to test partial programs.
- diffuseness – small goals should not require extraordinarily long solutions or large amounts of screen space.

- provisionality – the system should allow the user to sketch out uncertain parts of their solution.
- hard mental operations – none of the system's operations should require great mental effort to use.
- consistency – similar notations should mean similar things, and vice versa.
- abstraction management – the system should provide a way to define new facilities or terms that allow the user to express ideas more clearly or succinctly, but it should not force users to use this capability right from the start.

These factors are sometimes in conflict, so improving the system along one dimension can result in reduced performance on another. Tradeoffs are necessary, and in making these tradeoffs it is useful to consider cognitive models and observations from empirical studies.

2.5 Summary

Most existing mapping/transformation tools have poor data schema visualization and poor mapping specification environment to support a business analysis. The low–level graph-, tree-, table-like and UML class diagram presentations, and textually or semi-visually defining mapping specifications require that users of these tools must have a data modeling and programming background. In order to provide the end user support, our tool needs to overcome these problems by making use of existing techniques on end user programming, and findings on end user problem-solving behaviors, and applying usability design principles and heuristics throughout the development of our tool. It's important to provide the user a concrete, direct-manipulated environment which can make use of the user's previous knowledge and match to their cognitive problem-solving model.

Chapter 3 System Requirements Analysis

This chapter starts from a real life scenario to describe data transformation in a manual and an automatic system. Then a motivation of using form-based business copying metaphor for our system is discussed. Finally requirements of our system are described.

3.1 A Scenario

Let's consider the following business scenario of data transformation:

Comobile Solutions (CS) Ltd is a retailer for selling PDAs and their accessories. *TotalPDAs (TP)*, *AllHandhelds (AH)* are wholesalers for distributing PDAs and their accessories. In the beginning, *CS* orders goods from *TP*. But later *CS* shifts to *AH* because *AH* provides better service and technical support. *CS* has its own order generation system to generate orders, which contain information of supplier and order items, when its inventory is below a certain amount. Because orders, which contain information of purchaser and order items, used in *AH* and *TP* are different from *CS*, *CS* has to transform its orders from its own format to one of its supplier's before sending these orders to its suppliers.

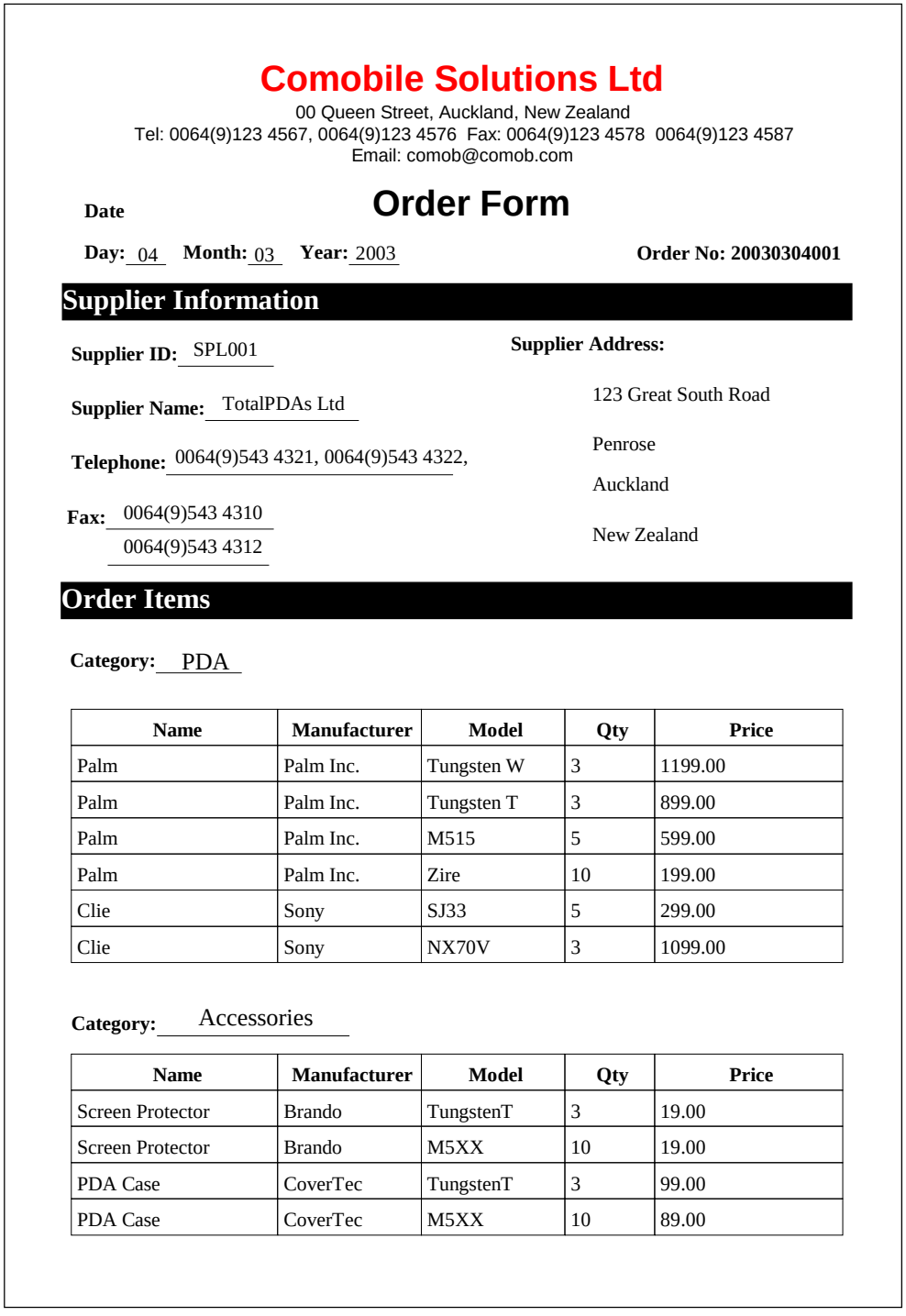
In following section, we first describe the scenario, in which *CS* orders goods from *TP* in a manual and automatic system respectively, and then the scenario, in which *CS* changes its supplier from *TP* to *AH* in the manual and automatic system respectively.

CS orders goods from TP

In the manual system, the order is represented in a physical form format, such as paper-based form, or electronic form, e.g. Access form or HTML form, which can be shown on computer screen. There is a data entry person, who could be in either side of source and target and is in charge of the data transformation. In our case, we suppose that the person in *CS* interpreters the meaning of fields in source and target form and finds the

context of them, then manually copies the data from source to target. Then the order is manually sent to *TP* from *CS* through mail, fax, email etc. Figure 3.1 shows a paper based order form of *CS*. The *TP* order form, which is different from the *CS* order, is shown on Figure 3.2. Figure 3.3 shows how the data in the order form of are manually mapped and copied to the order form of *TP*. It includes following business data mapping:

- One-to-one direct-copy: *ThisCompany TCName* in order of *CS* directly copied to *Customer Name* field in target form (see Figure 3.3(1)).
- One-to-many splitting: *Address* of *CS* is splitted to three parts to *Street, Suburb, City, State, Zipcode* and *Country* fields in target form (see Figure 3.3(2)).
- Many-to-one combining: *Year, Month* and *Day* fields in *CS* are combined to *Date* field in target form (see Figure 3.3(4)).
- One-to-many: Telephone numbers in one Telephone field in *CS* are splitted to a group of individual telephone numbers which are copied to telephone fields in target form, but formats of number are changed (see Figure 3.3(3)).
- Many-to-one: Individual fax number in a group of fax number is combined and then it is copied to a fax field in target (see Figure 3.3(5)).
- Many-to-Many conditional: *OrderItems* in target form are recategourized by manufacturer (see Figure 3.3(7)). For each *OrderItem*, if the manufacturer name is same, copy the record to the same category (see Figure 3.3(6)).



Comobile Solutions Ltd

00 Queen Street, Auckland, New Zealand
Tel: 0064(9)123 4567, 0064(9)123 4576 Fax: 0064(9)123 4578 0064(9)123 4587
Email: comob@comob.com

Date

Order Form

Day: 04 **Month:** 03 **Year:** 2003 **Order No: 20030304001**

Supplier Information

Supplier ID: SPL001

Supplier Name: TotalPDAs Ltd

Telephone: 0064(9)543 4321, 0064(9)543 4322,

Fax: 0064(9)543 4310
0064(9)543 4312

Supplier Address:

123 Great South Road
Penrose
Auckland
New Zealand

Order Items

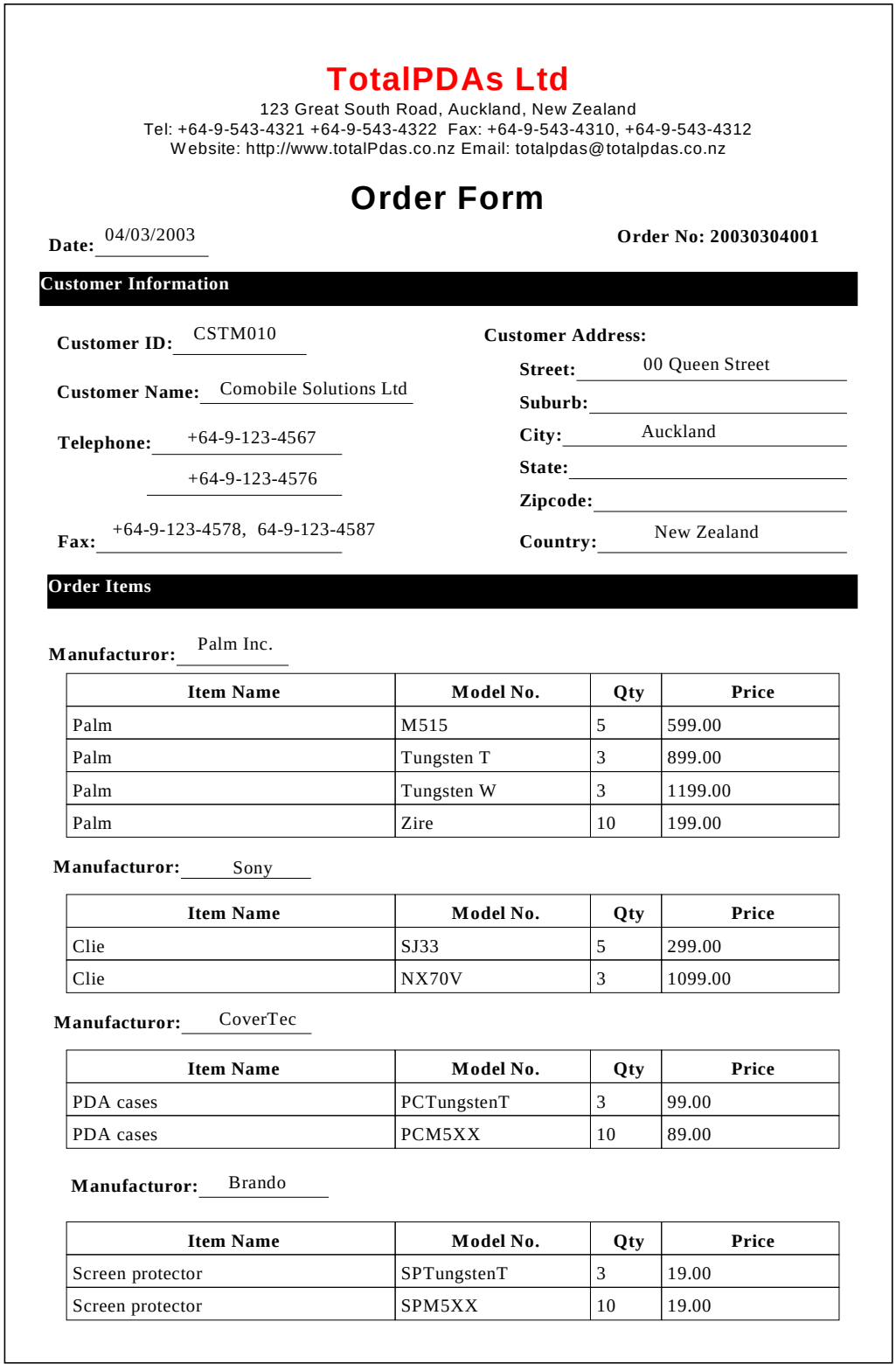
Category: PDA

Name	Manufacturer	Model	Qty	Price
Palm	Palm Inc.	Tungsten W	3	1199.00
Palm	Palm Inc.	Tungsten T	3	899.00
Palm	Palm Inc.	M515	5	599.00
Palm	Palm Inc.	Zire	10	199.00
Clie	Sony	SJ33	5	299.00
Clie	Sony	NX70V	3	1099.00

Category: Accessories

Name	Manufacturer	Model	Qty	Price
Screen Protector	Brando	TungstenT	3	19.00
Screen Protector	Brando	M5XX	10	19.00
PDA Case	CoverTec	TungstenT	3	99.00
PDA Case	CoverTec	M5XX	10	89.00

Figure 3.1 A paper-based *CS* order



TotalPDAs Ltd

123 Great South Road, Auckland, New Zealand
Tel: +64-9-543-4321 +64-9-543-4322 Fax: +64-9-543-4310, +64-9-543-4312
Website: http://www.totalPdas.co.nz Email: totalpdas@totalpdas.co.nz

Order Form

Date: 04/03/2003 **Order No: 20030304001**

Customer Information

Customer ID: CSTM010

Customer Name: Comobile Solutions Ltd

Telephone: +64-9-123-4567
+64-9-123-4576

Fax: +64-9-123-4578, 64-9-123-4587

Customer Address:

Street: 00 Queen Street
Suburb:
City: Auckland
State:
Zipcode:
Country: New Zealand

Order Items

Manufacturor: Palm Inc.

Item Name	Model No.	Qty	Price
Palm	M515	5	599.00
Palm	Tungsten T	3	899.00
Palm	Tungsten W	3	1199.00
Palm	Zire	10	199.00

Manufacturor: Sony

Item Name	Model No.	Qty	Price
Clie	SJ33	5	299.00
Clie	NX70V	3	1099.00

Manufacturor: CoverTec

Item Name	Model No.	Qty	Price
PDA cases	PCTungstenT	3	99.00
PDA cases	PCM5XX	10	89.00

Manufacturor: Brando

Item Name	Model No.	Qty	Price
Screen protector	SPTungstenT	3	19.00
Screen protector	SPM5XX	10	19.00

Figure 3.2 A paper-based *TP* order

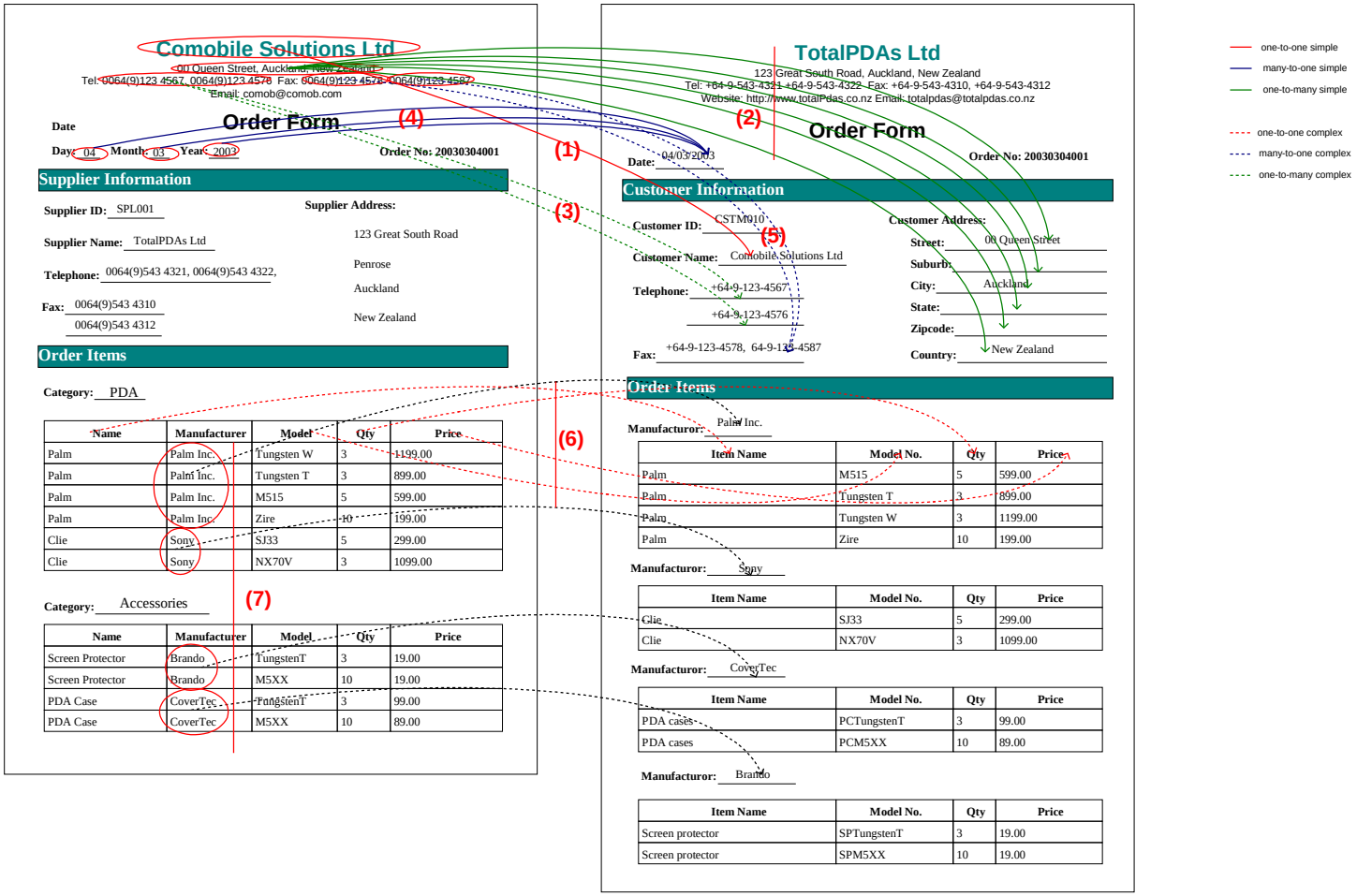


Figure 3.3 Data mapping between paper-based *CS* order and *TP* order

In the automatic system, the orders are represented in an electronic format. The order can be a set of related objects, or an XML message, or an EDI message. Figure 3.5 shows the orders of the CS and TP which are represented in an XML format. There is a computerized transformation system (see red boxes on Figure 3.4), which takes the order data from the *CS* as an input and mapping specification implementation between the CS and TP, and transforms them to conform the data format required by the *TP*. The data mapping specification implementation needs to be defined in the development stage. At beginning of the current development, a business analyst gives the business data mapping which just like the figure shown on Figure 3.3. Then a data modeler and programmer will design and implement the mapping based on the source and target data schemas. Figure 3.6 and Figure 3.7 show a low-level demo view of data mapping from the data modeler and the programmer perspective. Figure 3.6 shows the mapping between the source and target data schemas that are expressed in UML, and underlying data are objects. Figure 3.7 shows the mapping between the source and target data schemas that are expressed in XML DTD, and underlying data are XML files. From the figures we can see that different types of data message have different types of data schemas. The data schemas are complex and the mapping between them is far more complex. The definition of the mappings specification has to be implemented by a programmer who has data modeling and programming knowledge, even with help from a data mapping tool.

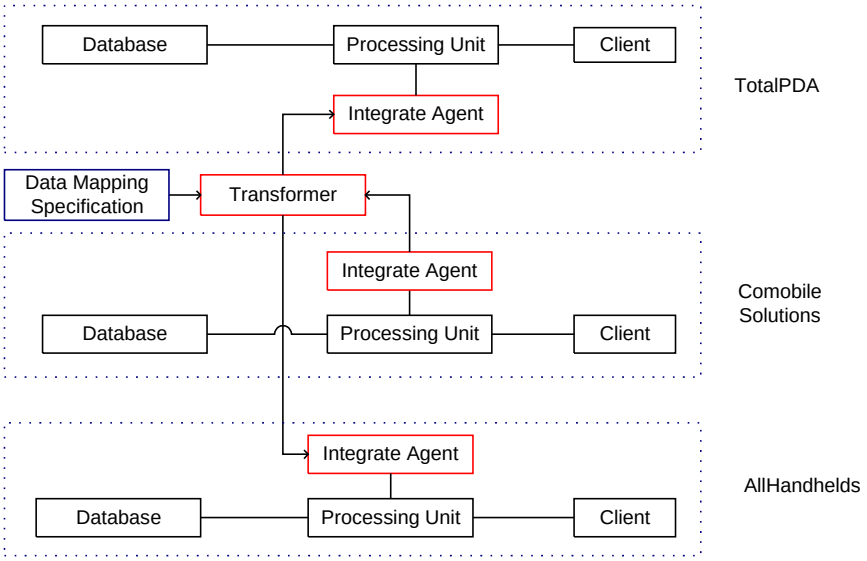


Figure 3.4 An integration model for *CS*, *TP* and *AH*

```
<?xml version="1.0"?>
<!DOCTYPE ComobileOrder SYSTEM "ComobileOrder.dtd">
<ComobileOrder OrderNo = "20030304001">
        <ThisCompany>
                <TCName>Comobile Solutions Ltd</TCName>
                <TCAddress>00 Queen Street, Auckland, New Zealand</TCAddress>
                <TCTel>0064(9)123 4567, 0064(9)123 4576</TCTel>
                <TCFax>0064(9)123 4578</TCFax>
                <TCFax>0064(9)123 4587</TCFax>
        </ThisCompany>

        <Date>
                <Day>04</Day>
                <Month>03</Month>
                <Year>2003</Year>
        </Date>

        <Supplier SupplierID="SPL001">
                <Name>TotalPDAs Ltd</Name>
                <Address>
                        <Street>123 Great South Road</Street>
                        <Suburb>Penrose</Suburb>
                        <City>Auckland</City>
                        <Country>New Zealand</Country>
                </Address>
                <Tel>0064(9)543 4321, 0064(9)543 4322 </Tel>
                <Fax>0064(9)543 4310 </Fax>
                <Fax>0064(9)543 4312 </Fax>
        </Supplier>

        <OrderItems>
                <Category>
                        <CategoryName>PDA</CategoryName>
                        <OrderItem>
                                <PartName >Palm</PartName>
                                <Manufacturer>Palm Inc.</Manufacturer>
                                <Model>Tungsten W</Model>
                                <QTY>3</QTY>
                                <Price>1199.00</Price>
                        </OrderItem>
                        <OrderItem>
                                 <PartName >Palm</PartName>
                                <Manufacturer>Palm Inc.</Manufacturer>
                                <Model>Tungsten T</Model>
                                <QTY>3</QTY>
                                <Price>899.00</Price>
                        </OrderItem>
                        <OrderItem>
                                 <PartName >Palm</PartName>
                                <Manufacturer>Palm Inc.</Manufacturer>
                                <Model>M515</Model>
                                <QTY>5</QTY>
                                <Price>599.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <PartName >Palm</PartName>
                                <Manufacturer>Palm Inc.</Manufacturer>
                                <Model>Zire</Model>
                                <QTY>10</QTY>
                                <Price>199.00</Price>
                        </OrderItem>
                        <OrderItem>
                                 <PartName >Clie</PartName>
                                <Manufacturer>Sony</Manufacturer>
                                <Model>SJ33</Model>
                                <QTY>5</QTY>
                                <Price>299.00</Price>
                        </OrderItem>
                        <OrderItem>
                                 <PartName >Clie</PartName>
                                <Manufacturer>Sony</Manufacturer>
                                <Model>NX70V</Model>
                                <QTY>3</QTY>
                                <Price>1099.00</Price>
                        </OrderItem>
                </Category>
                <Category>
                        <CategoryName>Accessories</CategoryName>
                        <OrderItem>
                                <PartName >Screen Protector</PartName>
                                <Manufacturer>Brando</Manufacturer>
                                <Model>TungstenT</Model>
                                <QTY>3</QTY>
                                <Price>19.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <PartName >Screen Protector</PartName>
                                <Manufacturer>Brando</Manufacturer>
                                <Model>M5XX</Model>
                                <QTY>10</QTY>
                                <Price>19.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <PartName >PDA Case</PartName>
                                <Manufacturer>CoverTec</Manufacturer>
                                <Model>TungstenT</Model>
                                <QTY>3</QTY>
                                <Price>99.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <PartName >PDA Case</PartName>
                                <Manufacturer>CoverTec</Manufacturer>
                                <Model>M5XX</Model>
                                <QTY>10</QTY>
                                <Price>89.00</Price>
                        </OrderItem>
                </Category>
        </OrderItems>
</ComobileOrder>
```

```
<?xml version="1.0"?>
<!DOCTYPE TotalPDAsOrder SYSTEM "TotalPdasOrder.dtd">
<TotalPDAsOrder OrderNo = "20030304001">
        <ThisCompany>
                <TCName>TotalPDAs Ltd</TCName>
                <TCAddress>
                        <TCStreet>123 Great South Road</TCStreet>
                        <TCSuburb>Penrose</TCSuburb>
                        <TCCity>Auckland</TCCity>
                        <TCState></TCState>
                        <TCZipcode></TCZipcode>
                        <TCCountry>New Zealand</TCCountry>
                </TCAddress>
                <TCTel>+64-9-543-4321 </TCTel>
                <TCTel>+64-9-543-4322 </TCTel>
                <TCFax>+64-9-543-4310, +64-9-543-4312</TCFax>
        </ThisCompany>

        <Date>04/03/2003</Date>

        <Customer CustomerID="CSTM010">
                <Name>Comobile Solution Ltd</Name>
                <Address>
                        <Street>00 Queen St</Street>
                        <Suburb></Suburb>
                        <City>Auckland</City>
                        <State></State>
                        <Zipcode></Zipcode>
                        <Country>New Zealand</Country>
                </Address>
                <Tel>+64-9-123-4567</Tel>
                <Tel>+64-9-123-4576</Tel>
                <Fax>+64-9-123-4578,  64-9-123-4587</Fax>
                <Fax>0064(9)543 4312 </Fax>
        </Customer>

        <OrderItems>
                <Manufacturer>
                        <ManufacturerName>Palm Inc</ManufacturerName>
                        <OrderItem>
                                <ItemName>Palm</ItemName>
                                <ModelNumber>Tungsten W</ModelNumber>
                                <QTY>3</QTY>
                                <Price>1199.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <ItemName>Palm</ItemName>
                                <ModelNumber>Tungsten T</ModelNumber>
                                <QTY>3</QTY>
                                <Price>899.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <ItemName>Palm</ItemName>
                                <ModelNumber>TungstenW</ModelNumber>
                                <QTY>3</QTY>
                                <Price>1199.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <ItemName>Palm</ItemName>
                                <ModelNumber>Zire</ModelNumber>
                                <QTY>10</QTY>
                                <Price>199.00</Price>
                        </OrderItem>
                </Manufacturer>
                <Manufacturer>
                        <ManufacturerName>Sony</ManufacturerName>
                        <OrderItem>
                                <ItemName>Clie</ItemName>
                                <ModelNumber>SJ33</ModelNumber>
                                <QTY>5</QTY>
                                <Price>299.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <ItemName>Clie</ItemName>
                                <ModelNumber>NX70V</ModelNumber>
                                <QTY>3</QTY>
                                <Price>1099.00</Price>
                        </OrderItem>
                </Manufacturer>
                <Manufacturer>
                        <ManufacturerName>Brando</ManufacturerName>
                        <OrderItem>
                                <ItemName>Screen Protector</ItemName>
                                <ModelNumber>TungstenT</ModelNumber>
                                <QTY>3</QTY>
                                <Price>19.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <ItemName>Screen Protector</ItemName>
                                <ModelNumber>M5XX</ModelNumber>
                                <QTY>10</QTY>
                                <Price>19.00</Price>
                        </OrderItem>
                </Manufacturer>
                <Manufacturer>
                        <ManufacturerName>CoverTec</ManufacturerName>
                        <OrderItem>
                                <ItemName>PDA Case</ItemName>
                                <ModelNumber>TungstenT</ModelNumber>
                                <QTY>3</QTY>
                                <Price>99.00</Price>
                        </OrderItem>
                        <OrderItem>
                                <ItemName>PDA Case</ItemName>
                                <ModelNumber>M5XX</ModelNumber>
                                <QTY>10</QTY>
                                <Price>89.00</Price>
                        </OrderItem>
                </Manufacturer>
        </OrderItems>
</TotalPDAsOrder>
```

Figure 3.5 Orders of the *CS* and *TP* which are represented in an XML format

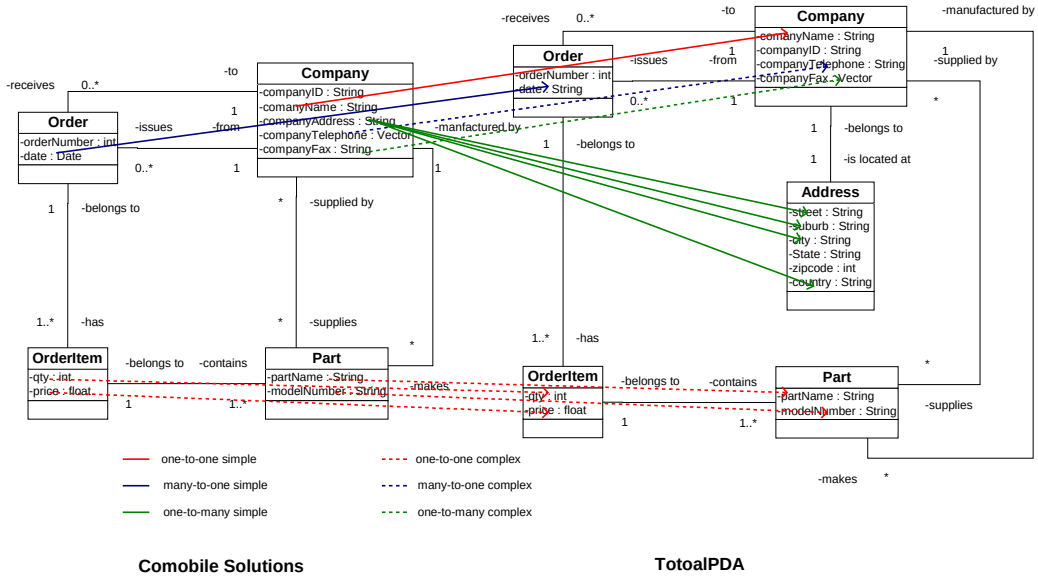


Figure 3.6 A demo data mapping between objects of *CS* order and *TP* order

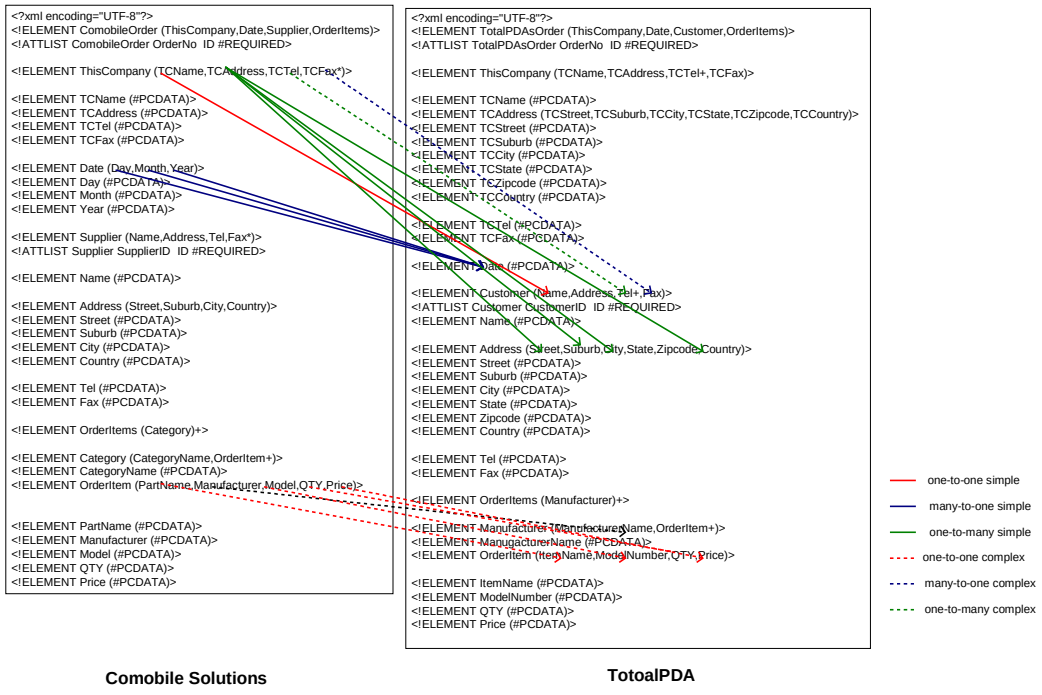


Figure 3.7 A demo data mapping between XML DTDs of *CS* order and *TP* order

CS changed its supplier from TP to AH

Due to differences between format orders of *AH* and *TP*, a mapping of business data from *CS* to *AH* is also different from the one from *CS* to *TP*.

In the manual system, the data entry person needs to remap the fields in order of *CS* to fields in order of *AH* and copy the data from source to target in the same when he/she did for order forms from *CS* to *TP*. Normally there is almost no time consumed and extra cost on the change.

In the automatic system, a renewed data mapping specification implementation needs to be developed according to the changed mapping of business data and fed into the transformation system. Development of renewed data mapping specification implementation follows the same software engineering process as that from the *CS* to *TP*. It needs a lot amount of work to complete the mapping specifications with great possibilities of errors and cost of money and time as we described in introduction chapter.

3.2 Our Approach

We wished to show that it is possible to let the business analyst define the data mapping specifications by developing a mapping tool to mimic the form copying process in the manual system,

From above scenario, we can see that meaning of structure and semantics of field between concrete order forms are easy for the clerk to understand. This makes copying data from one field in the source form to one in the target form to be direct and explicit because the clerk has knowledge of business process. That's reason why there is almost no time consumed and cost when business environment changes in the manual system.

From the previous chapter, we know that "one way to ease the entry into programming is to capitalize on the beginner's knowledge about the world. Many languages are based on a metaphor, which should be drawn from a concrete real-world system that is familiar to the user audience [Smith 1994]". According to this, in our mapping tool, we use a concrete order presentation similar to one in the manual system to visualize complex computer data schemas and import the data instance in the form to make it more concrete. This concrete and high-level abstract presentation hides the complexity

of the data schemas, and falls into a business analyst's problem domain which is easy for he/she to understand. In the mean while, based on the form-based metaphor, a spreadsheet-styled visual programming language and other end user programming techniques are used to let the users to define the most of mapping specifications just like copying form data in the manual system and defining formula in a spreadsheet without having a professional programming background. The spreadsheet is the most successful end-user programming system to date [Nardi 1993] and it also falls into the business analyst's cognitive model of problem solving. After the mapping specifications are defined, the tool will generate a mapping specification implementation. See Figure 3.8.

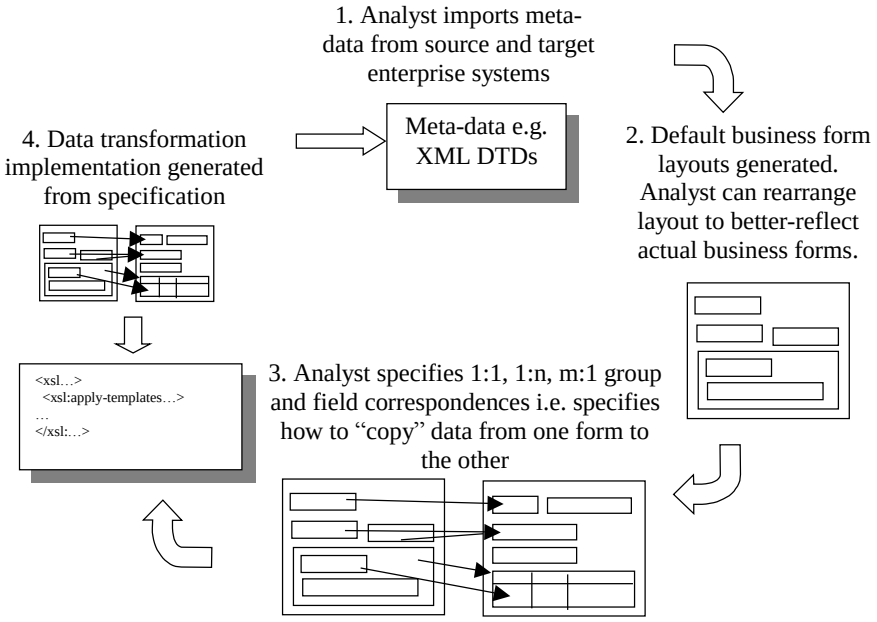


Figure 3.8 A high level data mapping process in our mapping tool. This figure originates from [Li 2002]

Our approach here is using the business form copying metaphor in our mapping tool to allow business analysts directly define the data mapping specifications and generate of data mapping specifications implementations.

3.3 Requirements of Our System

According to the above approach, the architecture of transformation system mentioned in the first chapter, and considering the end user problem-solving behaviors, general

principles and heuristics on usability and cognitive dimensions for visual programming described in the previous chapter, main requirements of our mapping tool is described as the following.

Need to support form visualization for multiple data schemas

The system should be able to automatically convert different data schemas, such as XML DTD, XML schema, EDI message, UML for object model, ER model, which are stored at anywhere on a local area network and the Internet, to a form-based presentation (see Figure 3.9 (1)). The form is one of the most common artifacts used in real world and is most familiar to the business analyst. The concreteness, directness and explicitness of form make the business analyst understand the data context at his best without knowing underlying technologies. It is also needed that our system should be architected flexible enough to process the different data schemas so that different processing units for converting data schemas to form-based presentation can be plugged in. When importing a data schema, the users select the kind of the data schema, and the system then choose a correspondent processing unit for it.

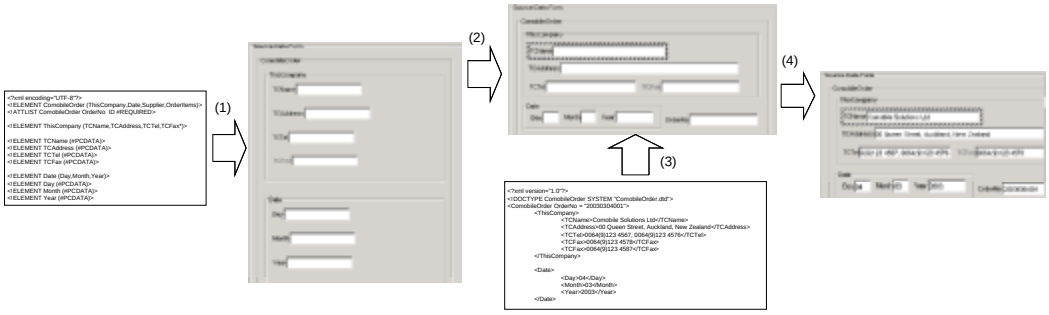


Figure 3.9 Form rendering process

Business form presentation can be customizable

The user can further modify layout of auto-generated form to make it close to real form to fit more his/her own taste, i.e. user can re-layout the form elements to make it more understandable for himself/herself (see Figure 3.9 (2)).

Be able to mapping by sample data

The system can import sample data to the business form presentation (see Figure 3.9 (3,4)). The sample data not only give the user a better understanding of the data type and format of form fields, but also enable the user to utilize the programming by demonstration technique to operate on the concrete data to define program, and furthermore give the user immediate feedback for debugging.

Need a visual business form copying environment

It should be able to be used by a business analyst and give him/her a concrete, direct-manipulated, explicit visual environment to mimic business form copying to define mapping specifications and get immediate feedback. The environment should satisfy most usability requirements we described in the previous chapter. The environment should cover all respects of business data form copying, i.e. following relations in field-, section-, collection-level with or without condition we will detail later:

- One-to-one, e.g. company name in order of *CS* directly copied to *Customer* Name field in target form.
- One-to-many, e.g. *TCAddress* of *CS* is splitted to three parts to *Street*, *Suburb*, *City*, *State*, *Zipcode* and *Country* fields in target form; *Telephone* numbers in one telephone field in *CS* are splitted to a group of individual telephone numbers which are copied to telephone fields in target form, but formats of number are changed.
- Many-to-one, e.g. *Year*, *Month* and *Day* fields in *CS* are combined to *Date* field in target form; individual fax number in a group of fax number is combined and then it is copied to a fax field in target.
- Many-to-many, e.g. *OrderItems* in target form are recategourized by manufacturer. For each *OrderItem*, if the manufacturer name is same, copy the record to the same category.

Need to support code generation

The system should have capability to generate different mapping specification implementation according to user mapping specification. The user can get either XSLT, or java or other language mapping specification implementation depending on what the user's need. It also requires our system should be architected flexible enough so that different code generation module can be plugged in.

Need testing and debugging support

The system should be able to take sample source data to produce the target data. This includes two levels:

- The individual field in the target form. In order to test and debug the mapping specification on the fly, after the user defining mapping specification on one field in target form, the system can take the sample data in source fields, and transform them to target data by using a transformation engine. For example,
- The whole target data. After the user completing mapping specification on the whole target form, the system can take the sample source file, and produce a target file by using a transformation engine.

Through above produced target data, the user is able to know if the defined mapping specification is correct and if not, the user can analyze the result to find where the problem is. Sample source data process and output data process units should be architected flexible enough to deal with various data formats

3.4 Main Modules of Our Mapping Tool

Figure 3.10 shows the main modules of our system according to the above requirements. The main functions of each module are described as following:

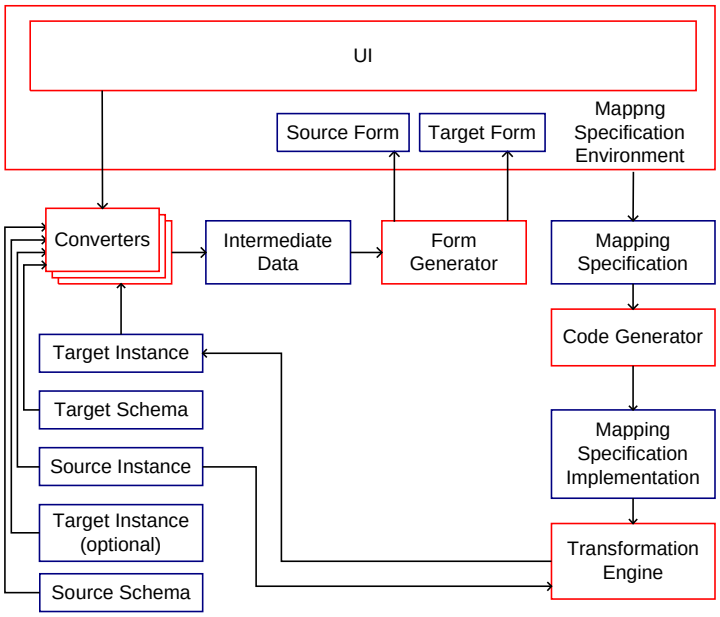


Figure 3.10 Main modules of our mapping tool

UI

The UI module is responsible for interacting with the users. It accepts information of source and target data schema and instance, and commands of file, editing, generating code, etc, from the users and invokes actions. It presents form presentations of the source and target schemas and instances generated from a form generator. It enables the user to re-arrange the form layout. Based on the generated source and target forms a concrete, direct-manipulated mapping specification environment is provided for the user to interact with it to define the mapping specifications by mouse clicking, drag-and-drop, and finally presents mapping specification results to the users.

Converters

It accepts commands from the UI, imports source and target data schemas and their instances, and then converts them to an intermediate data structure for form generator.

Form generator

It takes intermediate data from the converter, automatically generates forms and imports sample data to the forms. The user can rearrange elements in the forms.

Code generator

It accepts the mapping specifications defined by the user and generates a required mapping specification implementation.

Transformation engine

It accepts the generated mapping specification implementation from the code generator module and source data instance, and produces target data instance. Data in the target instance will be shown on the target data form for debugging and testing purpose.

3.5 Summary

Letting the business analyst to define data mapping specifications requires that our mapping tool should be end-user-oriented, i.e. this tool should render the complex underlying data schemas to a meaningful presentation to end-users, provide them a powerful mapping specification environment to define the complex mapping specifications without knowing programming, and generate the mapping specification implementations. In order to provide a user-friendly interface to make use of the user's domain knowledge, we decided to investigate using a business form metaphor to represent the underlying data schemas, and provide business form copying metaphor—a spreadsheet-styled end-user programming environment—for the business analysts to define mapping specifications and debugging them. Based on these requirements, the main modules of our tool are identified. All of these will guide our later development.

Chapter 4 System Design

In this chapter, we first select the architecture of our system, then give the user interface design for form rendering and mapping specifications, finally the static and dynamic specifications of object-oriented design of our system are described.

4.1 Architecture of the Tool

Software architectures have been identified as a critical design concern when bridging the gap between system requirements and implementation, particularly in large, complex software system [Kramer 1997]. Software architecture is the structure of the components of a program or system, their interrelationships, and principles and guidelines governing their design and evolution over time. It provides a clear and well-defined level at which to describe, understand, and analyze system designs.

4.1.1 Possible System Architectures of Our Data Mapping Tool

According to the main requirements we described in the previous chapter and uncertainty of requirements on budget of project, number of users, actual environment the mapping tool will run on, etc., we first consider our mapping tool as a standalone system and as a distributed system in a general way and then discuss them later.

4.1.1.1 Standalone Architecture

The standalone architecture of our mapping tool, which is actually 2-tiered, is shown on Figure 4.1. In this architecture, all the modules of the system we described in previous chapter are in a single application, which is the first tier. In the second tier, it’s the storage of input files and output files. The schema files and the instance data can be loaded from the local storage or anywhere on the Internet; the output files for mapping

specification implementation and target instance can also be placed to the local storage or anywhere on the Internet.

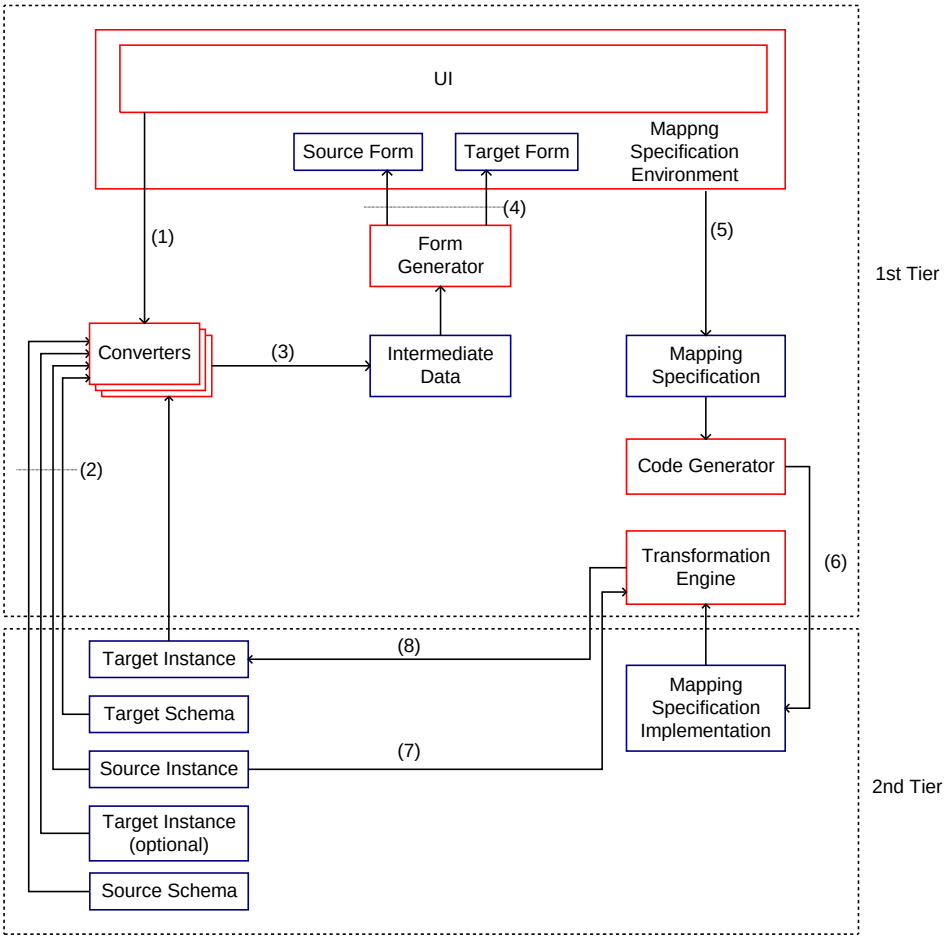


Figure 4.1 Mapping tool with standalone architecture

The processes for mapping specification in the standalone architecture are described as following:

The UI module is responsible for accepting commands, such as new, open, save, etc, and inputs about types and locations of schema files and instance data from the users, and sending them to corresponding converter modules (see Figure 4.1 (1)). The converter module accepts the information from the UI module, loads the schema file or instance data (see Figure 4.1 (2)), and then converts it to a unique intermediate data (see Figure 4.1 (3)). The form generator module accepts schema objects or/and instance objects with the intermediate data structures (see Figure 4.1 (4)), processes them and generates the forms or/and fills the instance data into the generated form. Then mapping

specification environment, which consists of the UI and the generated forms, accepts the users' instructions to build mapping specification (see Figure 4.1 (5)). When the mapping specification for one target field is finished, it will be sent to a code generator module to produce the mapping specification implementation (see Figure 4.1 (6)). The implementation and source instance are then fed to transformation engine module (see Figure 4.1 (7)) to output the target instance (see Figure 4.1 (8)). The target instance will be loaded to the converter module, then through the form generator module to show the instance data in the target form to the users who can determine the correctness of the mapping specification through the feedback. After the mapping specification for all target fields is finished, through the same process, the mapping specification implementation for the whole source will be produced and the target instance will be output. All of these results will be sent to and shown on mapping specification environment

4.1.1.2 Distributed Architecture

The distribute architecture uses a client-server model in which client sends a request to server and server gets the request, processes the request and then sends back a result to client. For our mapping system, the distributed system can be a 3-tierd or a 4-tiered architecture that are described in following sections.

4-tiered architecture

Figure 4.2 shows the 4-tiered architecture. The first tier is the client application, which is mainly for interacting with the users and sends the user requests to a server in next tier. It consists of a UI, a form generator and a mapping specification environment, which is based on the UI and the generated source and target forms. The second tier is a distribute server, which accepts requests from the client application and interprets them and directs or distributes them to correspondent applications in next tier, and then gets the replies from next tier, sends them back to the client application. The third tier consists of various converters for transferring different schema and data instance to the intermediate data structure, code generators for generating different mapping specification implementation from abstract mapping specification, and transformation engines for transferring source instance to target instance according to the mapping

specification implementation. These converters, code generators, and transformation engines can be run on different machines. The fourth tier is the storage of the source and target schema files, source and target data instance, and mapping specification implementation.

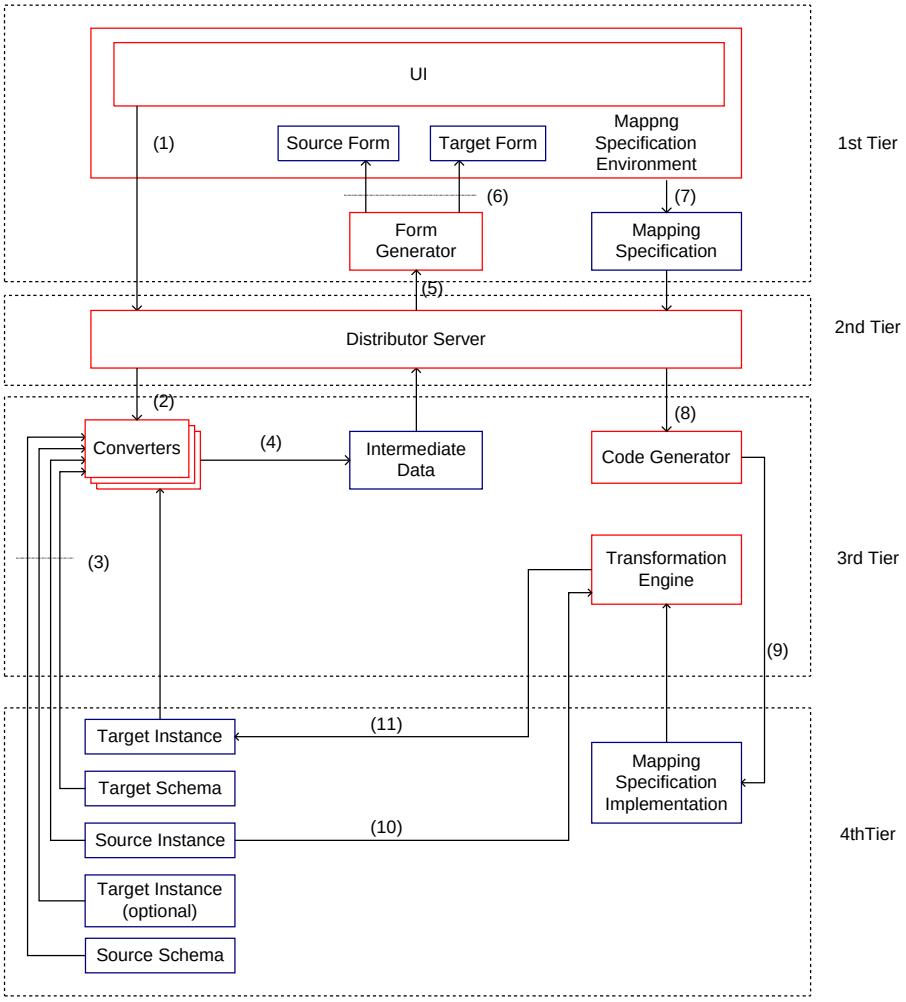


Figure 4.2 Mapping tool with a 4-tiered distributed architecture

The data processing in the 4-tiered architecture is described as following.

The client application accepts the new project command and inputs about types and locations of schema files and instance data from the users, and sending them to loader server in second tier (see Figure 4.2 (1)).

The distribute server accepts the information from the loader client, and distributes them to correspondent converters in the third tier (See Figure 4.2 (2)). It acts as a bridge between the client application and different processing units in the next tier.

Each converter gets input from the loader server and interprets them, then loads the schema file or data instance (see Figure 4.2 (3)) and converts it to the intermediate data structure, and sent it to back to the distribute server (see Figure 4.2 (4)), which will get all the messages from the converters, combine them together and send them back to the client application (see Figure 4.2 (5)).

The form generator module in the client application takes the intermediate data for the schemas as inputs to generate forms, and takes the intermediate data for the instances as inputs to fill the instance data into the forms (see Figure 4.2 (6)).

The users interact with the mapping specification environment, which consists of the UI and the generated forms, to specify the mapping specification for each target field. Once the mapping specification for one target field is finished, the environment sends the message of the mapping specification to the distribute server (see Figure 4.2 (7)). The distribute server then redirects it to a correspondent code generator for generating mapping specification implementation (see Figure 4.2 (8)).

The code generator accepts its input from the distribute server and generates the required mapping specification implementation which then is sent to the transformation engine (see Figure 4.2 (9)).

The transformation engine accepts the mapping specification implementation and source instance, and output the target instance (see Figure 4.2 (10~11)). Then the target instance is loaded to the converter to produce the intermediate data, and then the intermediate data is sent back to the client application through the distribute server for feedback to the users.

After the mapping specification for the all target fields is completed, the mapping specification implementation and the transformed target instance will be sent back to the client application the same way as above.

3-tiered architecture

In the 3-tiered architecture, the first tier is a combination of the first and the second tiers in the 4-tiered (see Figure 4.3). In order to make each client to know the latest address

of processing unit in the second tier, a database server or file is needed to serve at the third tier for the client to request or load. The data processing in the 3-tiered is almost the same as that in the 4-tiered except the client needs to request for the address information.

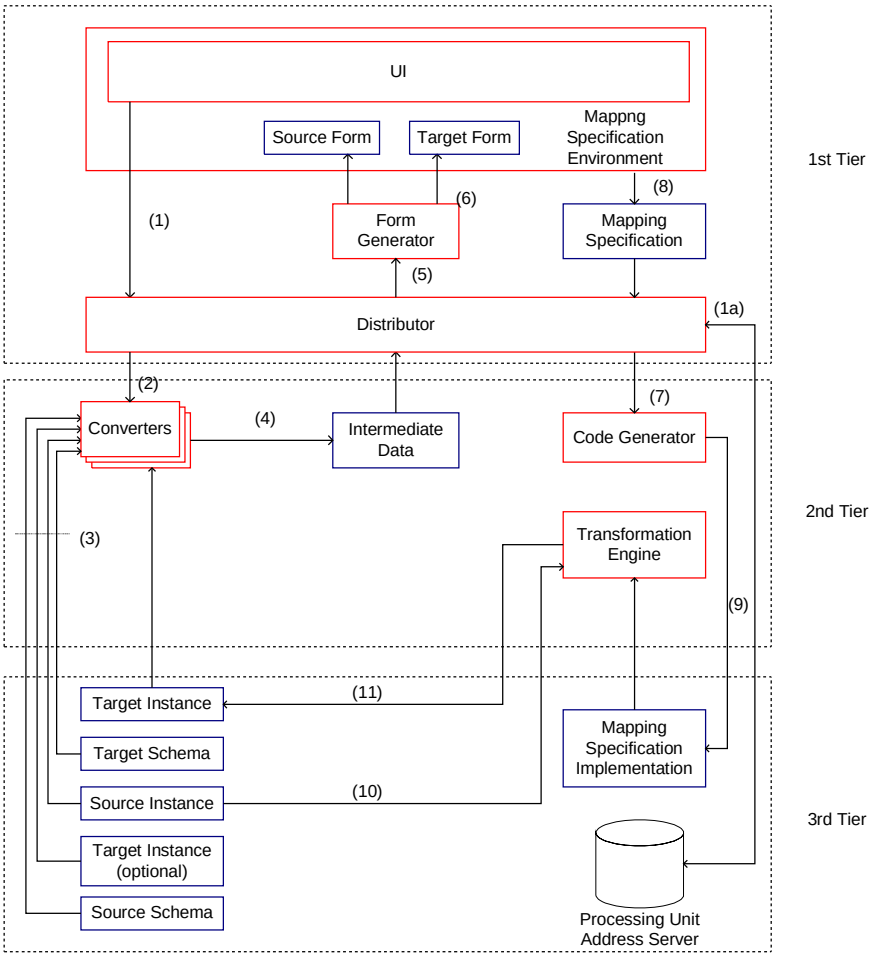


Figure 4.3 Mapping tool with a 3-tiered distributed architecture

4.1.2 The System Architecture We Choose

For the standalone system, the users need to install the whole application, which may be in a removable disk or CD, or downloaded from the Internet, to the users' local machine and run it. The application also can be a Java applet running in the Internet Browser. The users specify locations of the source and target schemas and instances, which can be on local machine or network, or somewhere in the Internet. After the data is loaded, the whole mapping specification processing will happen on the local application. The application only serves one user at one time.

For the distributed system, the users only need to install a lightweight client application, which may be in a removable disk or CD, or downloaded from the Internet, to the users' local machine and run it. The client application also can be a Java applet running in the Internet Browser. The users specify locations of the source and target schemas and instances, which can be on local machine or network, or somewhere in the Internet. After the data is loaded, the most of the mapping specification processing, such as converting schemas and instances to intermediate data, code generation, transformation from the source instance to the target result, will happen on the distributed processing units, which are scattered on the network or the Internet. Many clients can share the distributed processing units at the same time.

In the following, we compare the standalone architecture, 4-tiered and 3-tiered distributed architectures each other in terms of performance, complexity, reliability, scalability, flexibility extensibility and maintainability.

Performance

The standalone architecture makes all modules of the program in a single application. All modules are so tightly coupled each other that make the whole processing from inputting data to getting the result very fast. For the same type of source and target schema /instance, the correspondent converter has to processing them one by one.

For distributed architecture, there are three main factors downgrading its performance. The first is the communication between the client and server. The communication in our system involves the client application with distribute server, distribute server with processing units, code generator and transformation engine in the 4-tiered, and the client application with processing units, code generator and transformation engine in the 3-tiered. The second is the message parsing and producing. Each application in the system need to build messages from objects and send them back and forth, and parse messages to objects for further processing within application. The third is sharing the processing units. Each processing unit may process data from many clients.

On another hand, the distribute system can convert each schema or each instance in a separated parallel process by adding more distributed server and processing units, i.e. the schemas and instances can be processed simultaneously. This may make the

converting schema/instance to the intermediate data in distributed system faster than that in the standalone.

Complexity

The system with distributed architecture involves communication protocols for message transfer between the client application and the distributed server, the distributed server and processing units. Also inside these applications, the messages need to be parsed to objects for further processing, and built from objects for data exchange. Other issues like loading balance among processing units, and different operation environments. All these will make the system with distributed architecture more complex than the standalone architecture, although using XML technologies will simplify the process of integration of these applications.

The 4-tiered architecture is more complex than the 3-tiered, because the distributor in the 3-tiered is separated from the client application as an independent application—Distribute Server—in the 4-tiered. The distribute server needs to deal with communication and parse messages from both 1st and 3rd tiers.

Reliability

The system with distributed architecture needs the network to make their applications communicate each other. The problem with the main network path will cause poor reliability of the system. On the other hand, the distribution of the process units could make the system still work with failures of connection to some units, or breakdown of some units. For the standalone system, although there is no network problem, but malfunction with only one module in the runtime will cause the whole system crashes. With the improvement of the reliability of network, the reliability of the distribute system will get better reliability than the standalone system.

Extensibility

Both architectures can support extension of processing additional data schemas and instances or using different converters to processing the same data schema and instance, if the modules and patterns are properly designed. But with the distributed architecture,

extension can be just happened in the individual application, but with the standalone system, the extension has to be made on the whole application, even just one of the modules inside the application is extended.

Scalability

In the distributed architecture, the distribute server in 4-tiered can accept all client requests, and optimize the utilization of the processing units in next tier according to the load of next tier processing unit. But the distribute server may cause the bottleneck when too many clients connect to it. Additional parallel distribute server could be a solution for the bottleneck. In the 3-tiered, a bottleneck may happen on the processing units because of the unbalanced loading from the unorganized client applications.

In the standalone system, all the users just run the program on their local machine, and there is no bottleneck problem when the number of user increases.

Flexibility

The system with distributed architecture is more flexible than one with standalone architecture. In distributed architecture, each application can be replaceable, upgraded without influencing other applications in the development time, even in the runtime when there are multiple servers for the same function. With the standalone system, the whole system has to be replaced or upgraded by a new system.

Maintainability

In the standalone system, any changes of the module in the system cause the whole application to be upgraded. It needs to maintain the upgrade for a large number of the users. It is troublesome for both the developer and the user.

But in the distributed system, changes on the processing units will not affect the client application. Upgrading for the very limited number of units is very easy to be maintained. In the 4-tired, the changes on distribution functionality only affect the distribute server, not like that the changes of distributor causes that the entire client

applications need to be changed and every individual client application need to be upgraded in the 3-tiered.

From above analysis, each system has its advantages and disadvantages in terms of in terms of performance, complexity, reliability, scalability, flexibility extensibility and maintainability. The choice of architecture needs to consider the specific requirements on these aspects and other non-functional requirements of the mapping tool.

The standalone rather than the distributed architecture is chosen for later implementation based on following considerations:

- The main focus of our research is to investigate if we can develop a data mapping system that can be used by a non-programmer. The important part is the user interface that presents mapping data to a form and provides the way in which the user defines the mapping specification. The standalone architecture provides enough of this ability for me to investigate. So using the standalone application makes us focus on the main issue of our research and avoid the complexity of distributed system.
- Limitation of research time forces author to choose the simpler architecture so that it can be developed as quick as possible.

4.2 Form Visualization Design

To make the users understand the data and their relations in the data schemas without knowing the detail of complex technical terms of data schema, and achieve the best form visualization, we first decide to remove the data, which are the technical terms in the data schemas, for example, the namespace, element and attribute tags in XML DTD, from the form visualization, and only extract the data with business meaning and its relations from the data schemas and then present them on the visual form metaphor; second, we provide a capability for the users to rearrange the automatically generated form to make it more like a real business form; third, the users can import the schema instance to fill the sample data into the form fields to make the form more meaningful to them. We describe all the details in the following sections.

4.2.1 Form rendering

An intermediate data model is used as an input of the form generator. In order to make the form generator independent of various data schemas, such as UML for objects, XML DTD, EDI schema, these schemas need to be parsed and converted to an intermediate data model before they are taken into the forma generator.

In this intermediate data model, the technical terms are removed and the semantic of elements with business meaning and their relations are retained as those in their original schema but represented in a unified format. The intermediate data model uses a tree-liked hierarchical data structure represented by XML document objects, because the XML document objects can be easily processed by using current XML technologies, and make the system flexible for future extensions. In order to get the intermediate model, we first convert these data schemas to labeled graphs introduced in [Milo 1998]. But these graphs still have a lot of technical terms such as object reference nodes, data type nodes, inheritance relations, and different presentations for cardinality of node. It is very hard for the end user to understand these terms on a form-based presentation. These graphs may contain cyclic structure, which is difficult to be visualized on the form presentation. So we further build a tree structure to eliminate these object reference nodes, data type nodes; unify the different presentation of cardinality by using a relation node labeled "*zeroOrMore*", or "*oneOrMore*", or "*oneOrZero*", or "*or*" for choice; reorganize nodes with a inheritance relation by removing the super class node and moving its child nodes as child nodes of its sub-class node; present recursive relation by adding a child node with a label like "continue with <recursive node name> here" to the node, which refers to a recursive node. There is a map that stores the nodes in the tree structure and their correspondent nodes in the labeled graphs model.

The *CS* order schemas in XML DTD (see Figure3.7) and UML class diagram (see Figure 3.6) can be converted to the unified intermediate data model which structure is shown Figure 4.4. In the figure, there is a root node with edges and its child nodes. For XML messages, the root node corresponds to the root element of XML DTD. For objects schema represented by UML, the root node needs to be selected by the user. The red non-leaf nodes stand for the cardinality of its child node(s), it and other non-leaf nodes and leaf nodes represent data elements. The form generator takes the intermediate data model as its input and generates the forms.

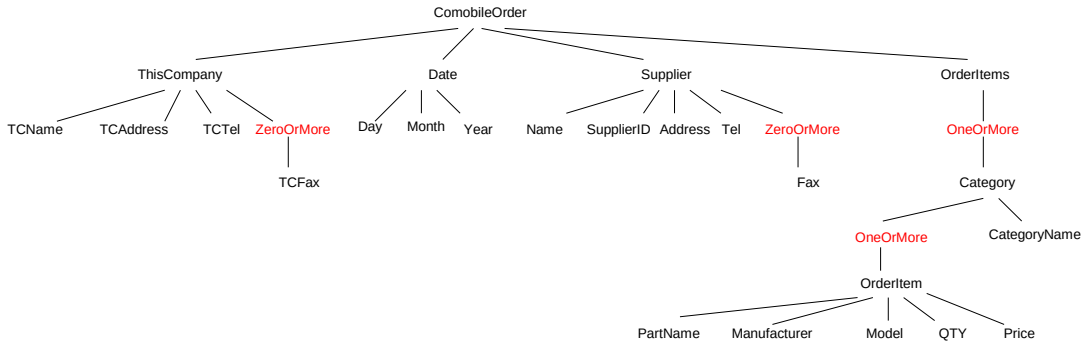


Figure 4.4 An intermediate data model of schema of CS order

The automatically generated form from the above intermediate data model is shown as Figure 4.5. The notation in the visual form for the above intermediate data mode is described as following:

A non-leaf node

A panel is used to represent a non-leaf node, not including the relation node. The name of the node-leaf node is shown on the border of panel. See Figure 4.5 (1) for *ThisCompany* node. The children of the node are rendered as visual components inside the panel. The panel is called a *section* or *group* in the form.

A leaf node

A labeled text field is used to visualize the leaf node. In the labeled text field, the label is used to represent the leaf node name and the text field to represent its value, which will be imported from an instance of the schema. See Figure 4.5 (2) for *TCAddress* of *ThisCompany*. The labeled text field is called a *field* in the form.

A relation node

Relation node is a non-leaf node which name is *zeroOrOne*, or *zeroOrMore*, or *oneOrMore*, or *or*. A black bold font is used for the name of a node with an *oneOrMore* relation (see Figure 4.5 (4)); a gray bold font for the name of a node with a *zeroOrMore* relation (see Figure 4.5 (3)); a gray font for the name of a node with a *zeroOrOne*

relation. A group of radio buttons with the names of node are used to represent the "*or*" relation. The field and section with *zeroOrMore* or *zeroOrMore* are a *collection* field and a *collection* section respectively.

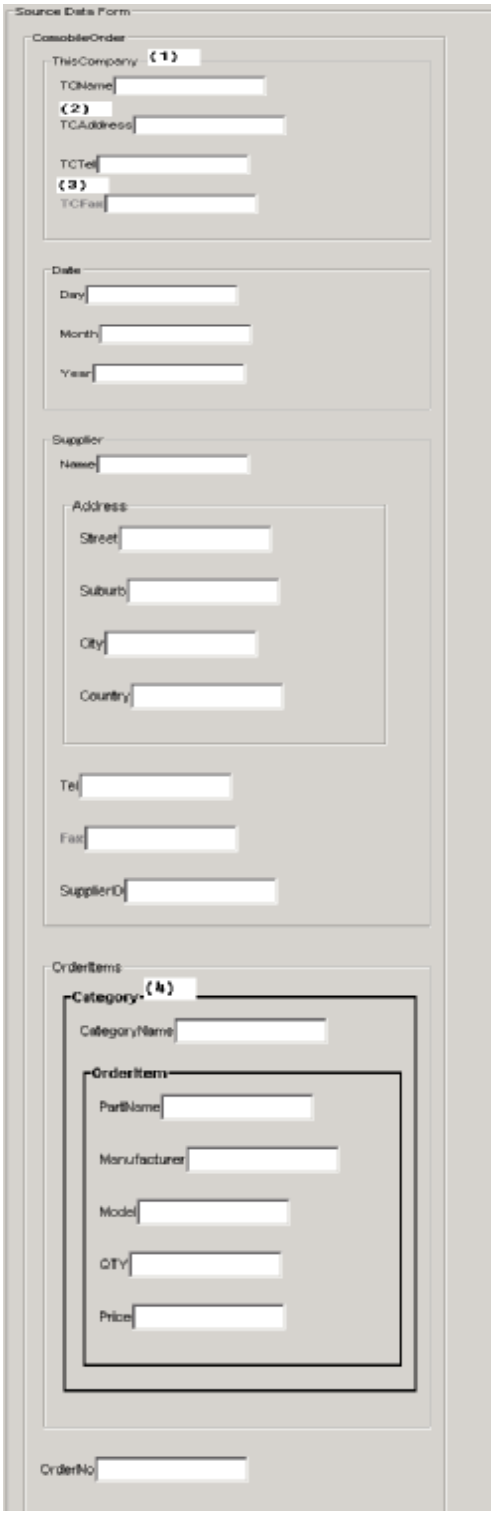
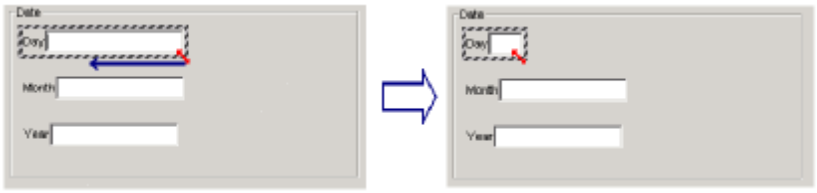


Figure 4.5 The automatically generated form for above tree structure of CS XML DTD

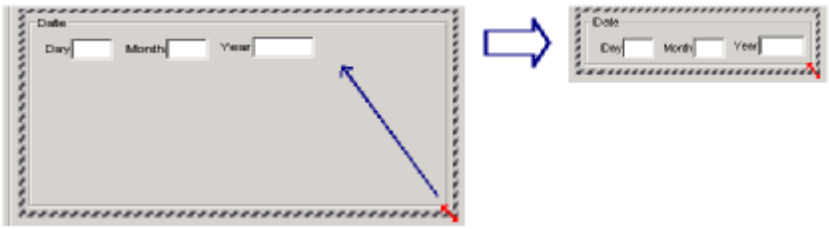
4.2.2 Reformatting Form

From above automatically generated form, we can see that the form looks very naive. It's better for the user to rearrange the layout of form to make it more intuitive or like an actual business form without changing the structure of underlying data. The system provides the capability for the user to select any fields or sections and move them to a proper position within their parent's section panel, or resize the fields or sections within their parent's section, to re-layout the form. Figure 4.6(1) (3) show how to resize a field and a section respectively. The user first selects the *Day* field or *Date* section by clicking a mouse button on it, then move the mouse to the corner of the selection box until a resize curser (it's red in figure) shows up, then drag the mouse to proper position (see blue arrow line) then release the mouse button to finish resizing. Figure 4.6(2) shows how to move *Month* field. The user first selects the *Month* field by clicking a mouse button on it, then move the mouse to the border of selection box until a move curser (it's red in figure) shows up, then drag the mouse to proper position (see blue arrow line) then release the mouse button to finish moving. Figure 4.7 shows the *CS* order form after the automatically generated *CS* order form in Figure 4.5 is rearranged.



(1) Resize the *Day* field

(2) Move the *Month* field



(3) Resize the *Date* Section

Figure 4.6 Rearrange form layout

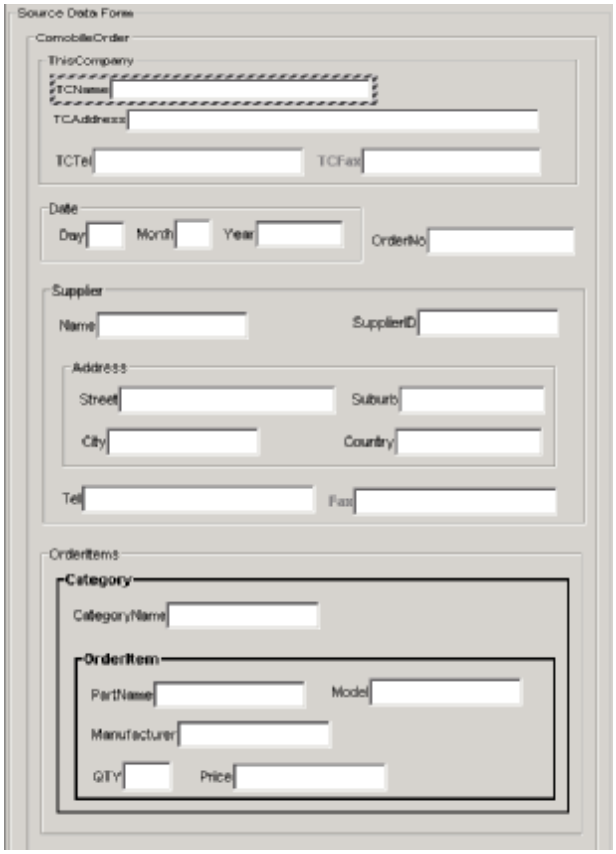


Figure 4.7 A rearranged CS order form

4.2.3 Importing Sample Data

Sample data from an instance of schema can be imported into the source or target form. The sample data can make the user understand not only meaning and semantics of form field but also the data format and type. Furthermore, the sample data also can be used for programming by demonstration and debugging purpose later on. In order to show sample data, the user can just click on a show-source-data-button or show-target-data-button (see Figure 4.8) to show the data directly in text fields in the source or target form if the user has already imported the schema with its instance. Otherwise, after the user clicks on the button, a file chooser will pop up for opening the instance file. After the instance file is opened, the data will show on the form. Figure 4.9 shows *CS* order form and *TP* order form after show-source-data-button and show-target-data-button are pressed.

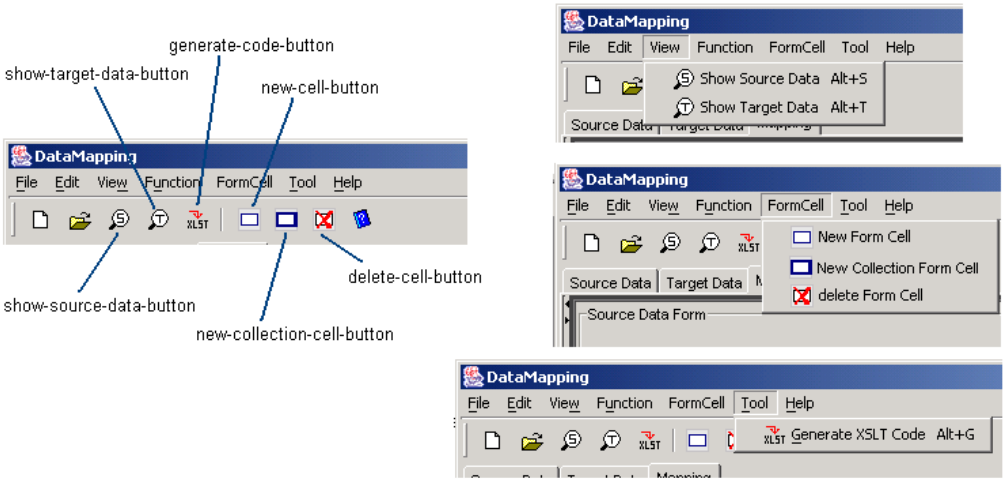


Figure 4.8 Tool buttons and menu

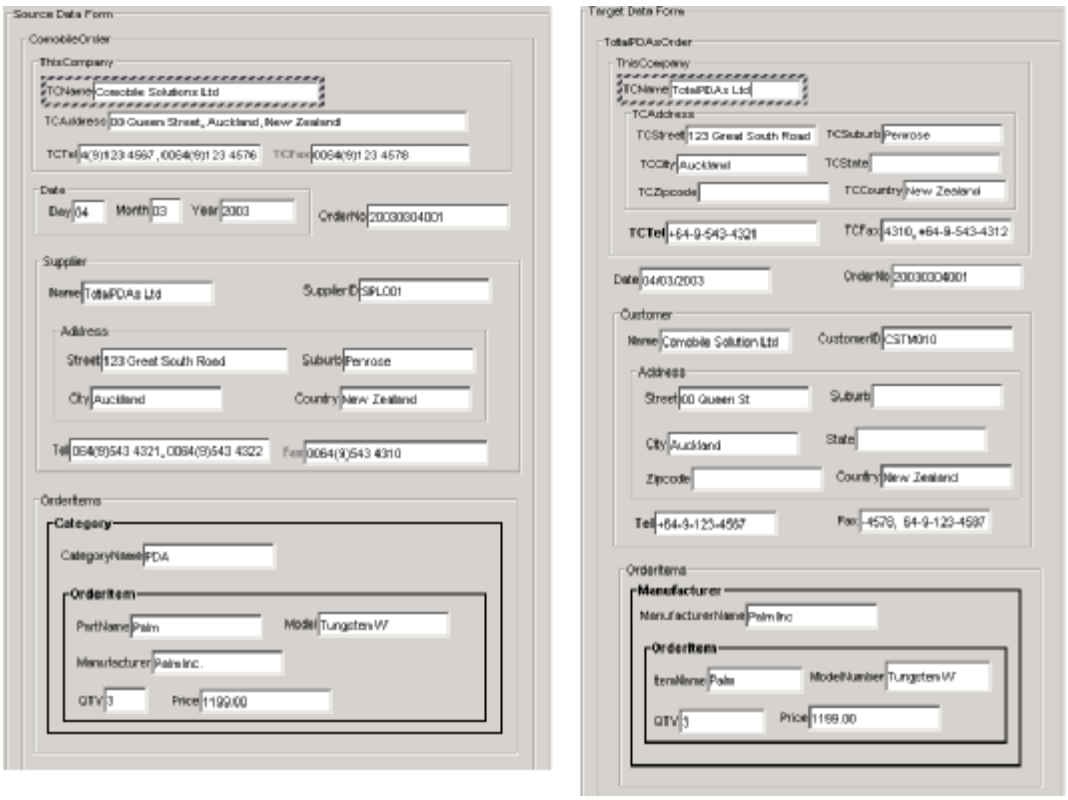


Figure 4.9 CS and TP order forms after the sample data is imported

4.3 Visual Mapping Specification Environment Design

The visual mapping specification environment should provide a user a concrete, direct-manipulated environment which can make use of the user's previous knowledge and

match their cognitive model of problem solving to make. In the following sections, we make use of existing end user programming techniques, findings on end user problem-solving behaviors from researches on end user programming, and usability principles on user interface design, and apply them to our mapping specification environment design.

4.3.1 Outlook of Mapping Specification Environment

Our visual mapping specification environment is a spreadsheet-styled programming environment, which is shown on Figure 4.10. The spreadsheet-styled environment is actually form-based user interface, so it is consistent with our business form presentation for underlying data schemas. It consists of a source and target form area (see Figure 4.10 (1)), an intermediate form area (see Figure 4.10 (2)), an operation selection area (see Figure 4.10 (3)), and a formula display area (see Figure 4.10 (4)).

Source and target forms area

This area consists of both source and target forms. All the sections and fields in the form are selectable by clicking mouse on them. Each selectable field in both source and target forms is treated like a cell of spreadsheet. The user can define the mapping specifications by either drag-and-drop, i.e. first selecting a source field and then dragging it to a target field, or type-and-select, i.e. first selecting a target field and then building formula or procedure by first entering equal symbol and then selecting the source field from the source form or intermediate form area, or selecting an operation from the operation area. We will give the samples later.

The intermediate form area

When defining some complex mapping specifications, sometimes we need to use some intermediate results. We can do it by creating an intermediate field, which also a cell of spreadsheet, to get one of the intermediate results, and then refer to the field from a target field or another intermediate field. This intermediate field is not a part of the source and target forms. It is used separately as a variable for defining the mapping specifications for its target field. Now the environment supports the intermediate field

and an intermediate collection of field. This is the methodology of divide-and-conquer, in which a complex problem is divided into several small problems. We will give examples later.

The operation selection area

In this area, there are a lot of common used operations for mapping specifications listed by using a tree structure to categorize these operations. The operations including operations on strings, such as ***substring_before***, ***concatenate***, etc, numbers, such as ***round***, ***ceil*** etc, the fields in the forms, such as ***sum***, ***position*** etc, and logical statements, such as ***if, while*** etc. These operations are pre-defined and can be reused by just clicking on the desired operation nodes.

The formula display area

This area provides a way for defining, displaying, editing, and deleting mapping specification for current selected cell in target or intermediate form area.

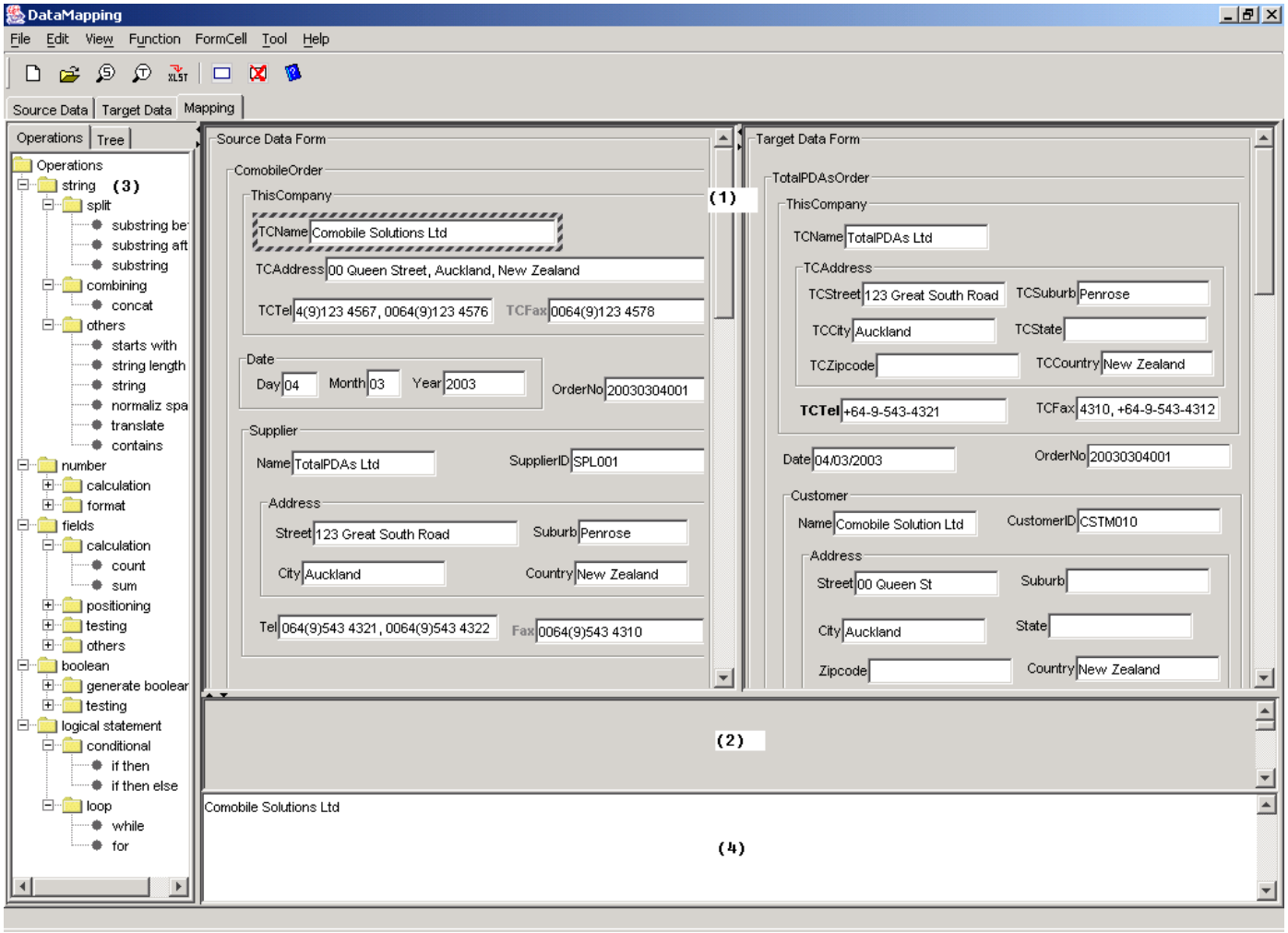


Figure 4.10 Visual mapping specification environment of our tool

4.3.2 User Interfacing and Notations for Mapping Specifications

As we described above, the environment of mapping specification of our tool is a spreadsheet-styled end user programming environment. We design the environment to make it as a business form copying metaphor through following ways:

By using the spreadsheet-styled programming environment makes the user use it like the way he/she uses Microsoft Excel [Microsoft 2003 Excel] or Forms/3 [Hays 1995], i.e. selecting the target field and defining a formula for it. A spreadsheet is the most successful end-user programming environment in the business world [Nardi 1993]. It can make use of the business analyst previous domain knowledge and problem-solving skill.

Built-in high-level operations, which are quite frequently reused in mapping specifications, are provided to avoid the user to synthesize them from many simple primitives [Lewis 1987]. These operations are listed on a panel and the user can select them for use by just clicking on them.

According to that an end user prefers to express the general case first, and then later modify it with exceptions [Pane 1996] [Myer 1998], in our mapping tool, when the user defines mapping specification by using drag-and-drop, a default copying operation without conditions is applied. The user can add conditions for the default operation at later time.

Based on the spreadsheet-styled programming environment, a type system is introduced, and the user can apply a type to form fields and sections to make most of format conversions, splitting and combining mapping specifications to be defined just through a drag-and-drop operation. Applying a type to form fields and sections is not compulsory.

In the following, we detail how the mapping specifications are defined in our mapping tool.

4.3.2.1 The Type System

The type system is used to define and apply type and format on the value of the form field, and the section in the form. Some common used types, such as date, time, person

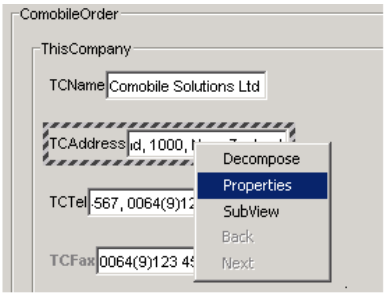
name, address, number, telephone and their formats are built-in the system. The users can also define their own types and add to the system by using a programming by demonstration [Cypher 1993] technique. But the users are not forced to apply the types to the form fields and sections; they do it at their convenient.

Apply a type to a form field

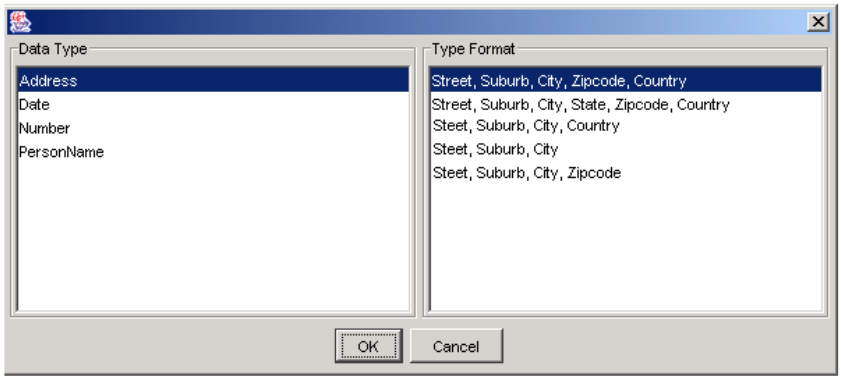
Figure 4.11 shows processes for applying the type to ***ThisCompany.TCAddress*** field in CS order form. To apply the type to the form field, the users first select the field, right click the mouse to show the popup menu, select the properties (see Figure 4.11(1)); then a type dialog box shows up; the users select proper type and format for the form field, then click OK (see Figure 4.11(2)). After the type is applied to the form field, if there is more than one attribute in the type, there is a red plus sign showing on the right of the form field (see Figure 4.11(3)). When the red plus is clicked, there is a pop-up sub-form, which contains the attributes of the type and sample data, and the plus sign changes to minus sign (see Figure 4.11(4)). If the users click on the minus sign, the pop-up sub-form disappears and the minus sign will change to plus sign.

Apply a type to a form field

Figure 4.12 shows processes for applying the type to ***Customer.Address*** section in TP order form. To apply the type to the form section, in the beginning, the procedures are the same as applying the type to the form field. The users first select the field, right click the mouse to show the popup menu, select the properties (see Figure 4.12(1)); then a type dialog box shows up; the users select proper type for the form field, then click OK (see Figure 4.12(2)). Then the following steps are different from applying the type to the form field. If one of names of the attribute in the type doesn't match any name of child of the section, a dialog box will show up to let the users map the attribute to a field in the section. If the name of the attribute in the type is the same as the label of the field in the section, the mapping will automatically be done by the system (see Figure 4.12(3)). After finishing the mapping, the users click on the OK button. A red star shows on the right corner of the section to indicate a type being applied to the section (see Figure 4.12(4)).

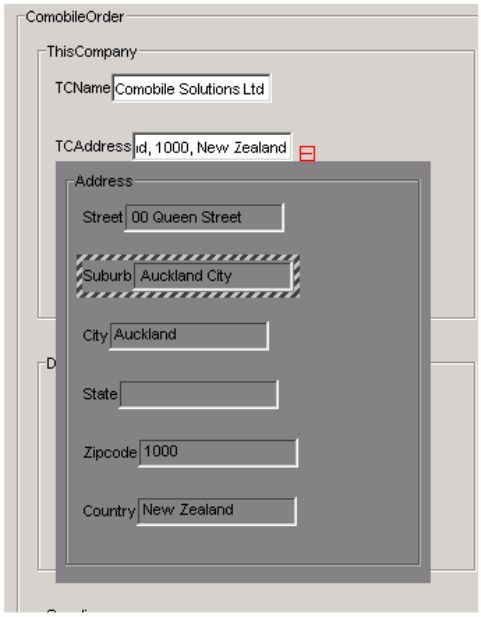


(1) Select TCAddress field, right click mouse, choose properties from the pop-up menu.



(2) Type selection dialog box shows up, choose data type and format, click on OK button

(3) Red plus sign shows on the right side of TCAddress field, click on the plus sign.



(4) A sub-form shows up, and the plus sign changes to the minus sign. Click on the minus sign, it returns to the (3) state.

Figure 4.11 Apply a type to a form field

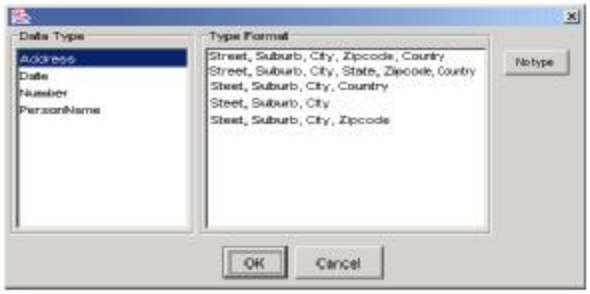
All the types applied to the form field and sections can be modified and removed.

Applying type to the fields and sections in the form can make the mapping specification for the fields and sections much easier than no typing at all. We will see it later.

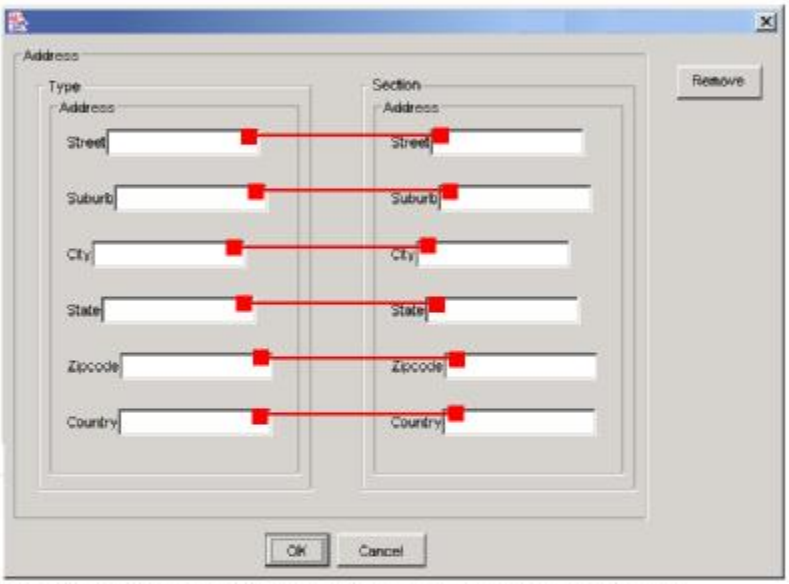
Define a type

If a type that is frequently used but are not built in the system, the user can define it by demonstrating on a concrete sample data and then system will generate a new type which can be used by the user at later time. Now the type definition only applies to a type with attributes which type is primitive type. Figure 4.13 shows procedures to define a new type by using programming by demonstration technique. The user just need to create new sub-elements and select text from the field string and drag and drop it to the correspondent cell of sub-element (see Figure 4.13 (2~6)). After you push the Add Type button, the system will generalize what you did, form a type and add to the system for later use. We can see from Figure 4.13(8), after TCAddress is assigned to Address type, generalized program is applied to the sample data of the TCAddress field and a sub-form with sample data is generated. We can check the type manager to see if the Address type is there ready for use at later time (see Figure 4.13 (10)).

(1) Select Address field, right click mouse, choose properties from the pop-up menu.



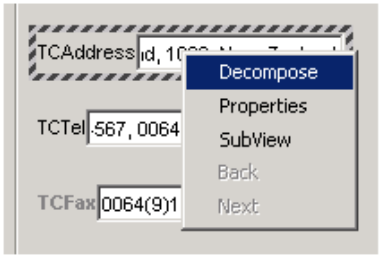
(2) Type selection dialog box shows up, choose data type, click on OK button



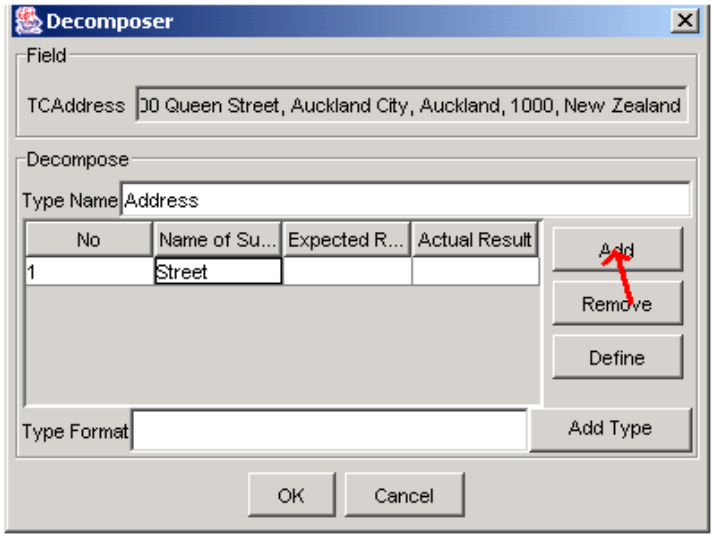
(3) A dialog box for the attributes of the type matching children of the section shows up. If there is a child name in the section the same as the name of attribute in type, the program will automatically match them up. Otherwise, a **manual mapping** is needed. Click OK button to finish the matching.

(4) A star sign shows on the right side of the Address section to indicate a type applied to it.

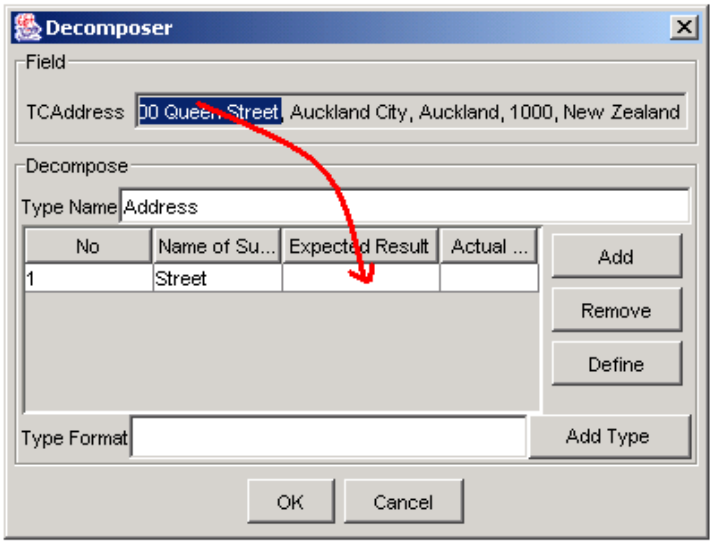
Figure 4.12 Apply a type to a form section



(1)Select TCAddress Field and right click mouse, shortcut menu shows up, select Decomposer.

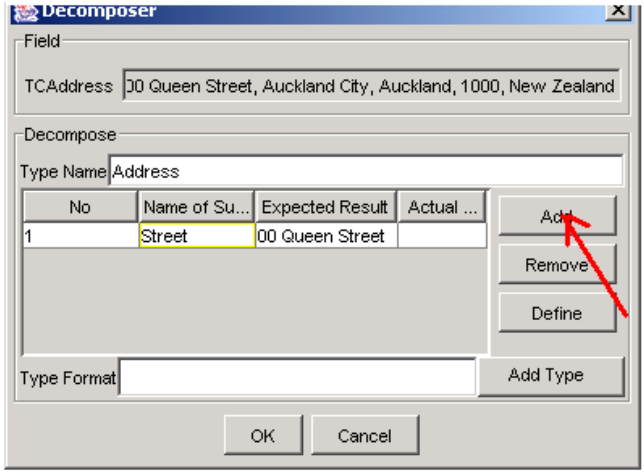


(2)Decomposer dialog box shows up. Input type name in Type Name field. Add click on add button, and input Street in the first line and the second column

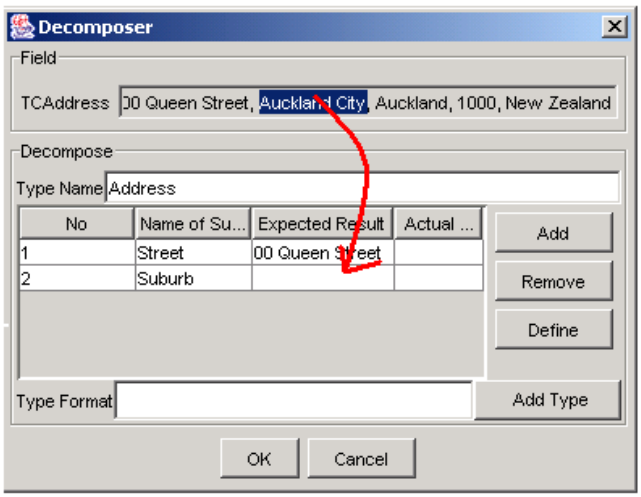


(3) Select "00 Queen Street" and drag and drop to the first row and the third column.

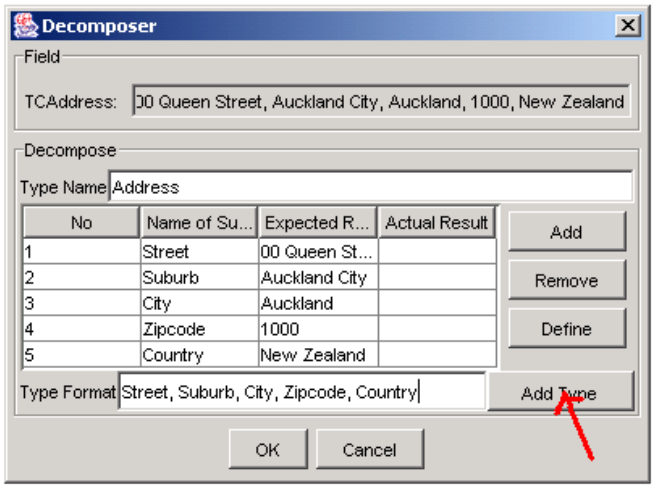
Figure 4.13 Define a new type by using programming by demonstration technique



(4)Now we finish the Street definition. Click on Add button again to add more Subelement.

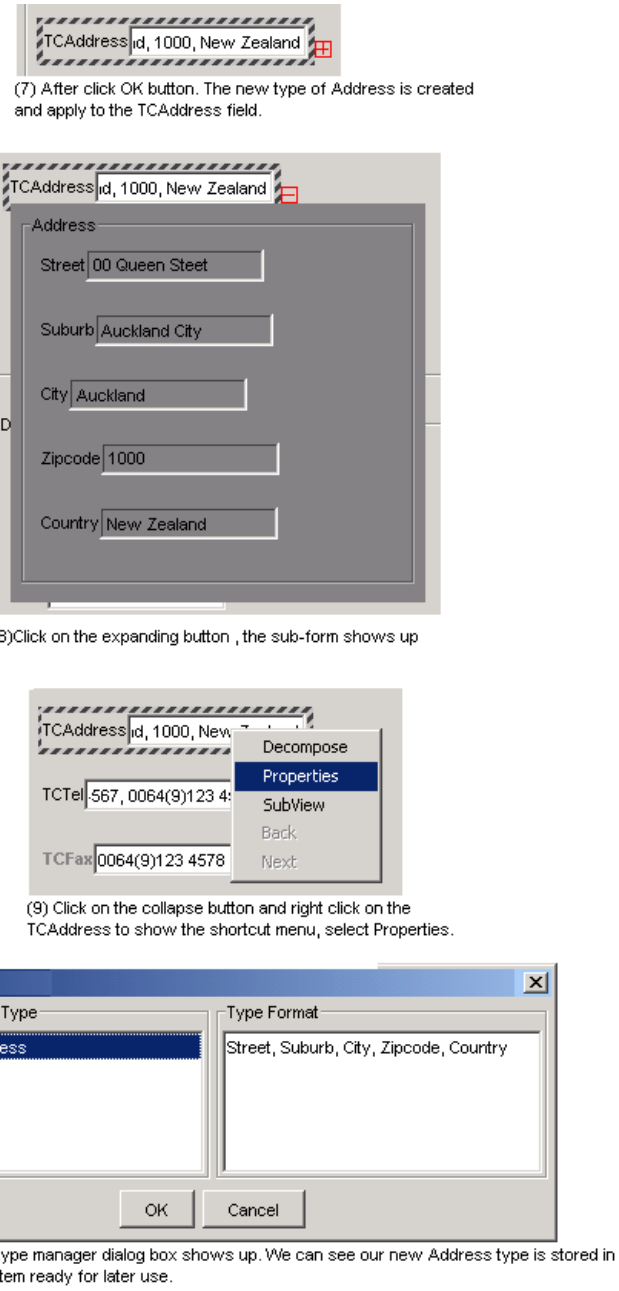


(5) Input "Suburb" in the second row and the second column. And select "Auckland City" from the TCAddress field, add drag and drop it to the cell below "00 Queen Street"



(6) Repeat the above procedures to define the City, Zipcode and Country. And Then click Add Type button to add the type and click on OK button.

Figure 4.13 (Continued)



(7) After click OK button. The new type of Address is created and apply to the TCAddress field.

(8)Click on the expanding button , the sub-form shows up

(9) Click on the collapse button and right click on the TCAddress to show the shortcut menu, select Properties.

(10) A type manager dialog box shows up. We can see our new Address type is stored in the system ready for later use.

Figure 4.13 (Continued)

4.3.2.2 Mapping Specifications

In order to make clear of mapping specification on the business form copying metaphor, in following sections, we will use some symbols to present the mapping relations. These symbols are described as following:

Figure 4.14 (1) shows the symbol of form field. The square stands for the form field. Inside the squares, there is a triangle or circle(s). The different shape of these little widgets stands for different value in the field. The different outline color of the triangle or circle stands for a difference of the label of field. The blue color of widget (Figure 4.14 (1)d) stands for the type of the value in the field being defined. The multiple widgets in the field (Figure 4.14 (1)e) stand for the data in the field consisting of multiple data elements.

Figure 4.14 (2) shows the symbol of form section, which contains many form fields and/or section(s). The blue square (Figure 4.14 (2)c) stands for the type of the section being defined.

Figure 4.14 (3) shows the symbol of collections, which can be collection of field or collection of section.

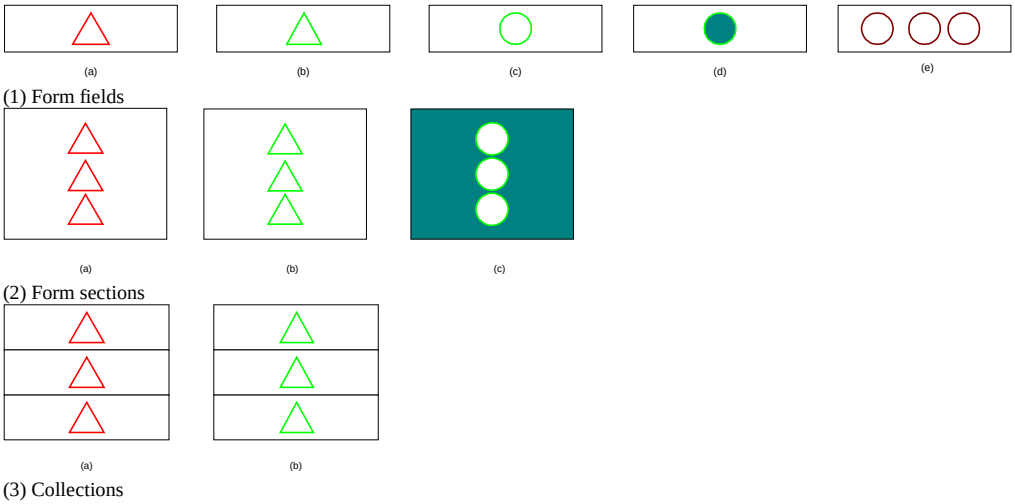


Figure 4.14 Symbols used for illustrating mapping specification

4.3.2.2.1 Simple mapping specifications

In the simple mapping specifications, there are no collection fields and conditions involved in the mapping specification formula for the target field. They include the following situations:

- One-to-one

It only involves one target field and one source field. Both fields are not collections. The formula in the target field only contains only one source field reference.

- Direct copy

This is the simplest mapping specification. The value of the target field is exactly the same as the value of the source field (see Figure 4.15). We just directly copy the value in the source field to the target field. The formula in the target field is only the reference of the mapped source field. For example, the value of *Customer.Name* field in TotalPDAs order is a direct copy of the value of the *ThisCompany.Name* field in CS order. To specify the mapping, the user can use drag-and-drop to select the source field and then directly drag and drop it to the target field (see Figure 4.16), or use type-and-select to select the target field first and enter an equal symbol in the field, then select the source field and enter return key (see Figure 4.17). When finishing the mapping specification, we can see that a line between the source field and target field indicates the direct copy mapping specification.

A special case for the mapping is to assign a constant value to a field in the target form. It can be simple done by selecting the target field and type the value in the target text field.

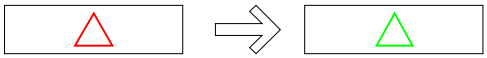


Figure 4.15 One-to-one direct copy

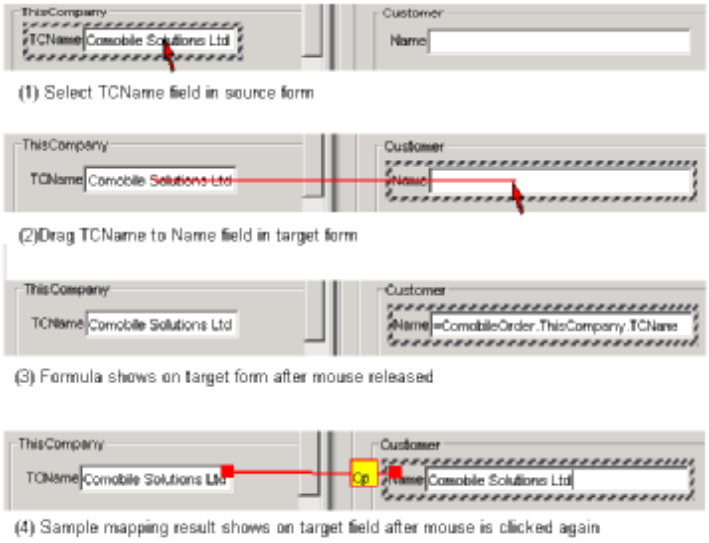


Figure 4.16 One-to-one copy by drag-and-drop

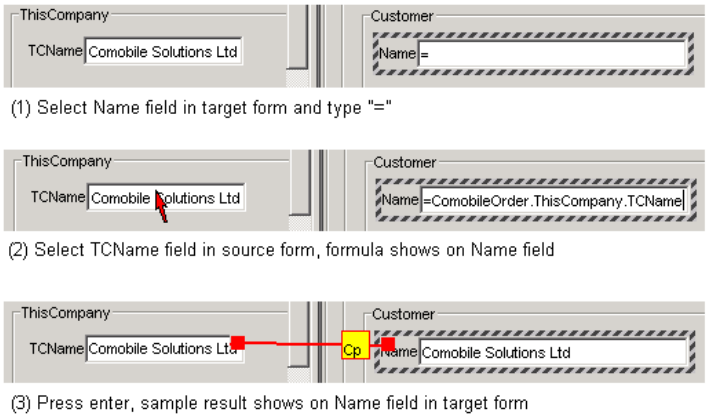


Figure 4.17 One-to-one copy by type-and-select from CS order to TP order

- Formula

In this mapping specification, the value or format of the value in the source field is different form one in the target field (see Figure 4.18). The formula for the target field needs to be defined to transfer the source value to the target value. This is normally for the mathematical calculation, string manipulation and operations for fields, sections. For example, the value in *Street* field in *Address* Section in *TP* order is a substring of the value of *TCAddress* field in *ThisCompany* Section in the *CS* order.

There are two ways to define the mapping specification. The one is directly defining the formula using type-and-select when there is no type applied for the source and target

fields (see Figure 4.18(1)). For above example, the user first selects the target field and enters equal symbol (see Figure 4.19(1)), and then enters the formula. When there is a reference to a source field in the formula, the user just click on the source field in source form. When needing a function, the user can use mouse to browse and select a desired operation in the operation section (see Figure 4.19(2)), and then following the instruction on operation dialog box. In our example, a ***string_before*** operation is used to get the string before the first "," in the address in *TCAddress* field. Form the dialog box (see Figure 4.19(3)), we first click on the top *Select* button to select *TCAddress* field from the source form (see Figure 4.19(4)), and then the dialog box is back again (see Figure 4.19(6)). Click on the combox button, choose "," from the pop-down list, and then press *OK* button (see Figure 4.19(6)). We can see the formula is shown on the target field (see Figure 4.19(7)). Enter ***Return***, the result of the formula is shown on the target field, it is exactly the street value! See Figure 4.19(8)). A line between the source field and the target field indicates the formula mapping specification.

(1) Non-typed one-to-one (2) Typed one-to-one

Figure 4.18 One-to-one formula

The other way to define the formula is through first applying type to both the source field and the target field and then drag-and-drop (See Figure 4.18(2)). Here let's take a simple example, the source Date field (format dd/mm/yy) mapping to the target Date field (format mm/dd/yy). Figure 4.20 shows the procedures of the typed mapping specification. The users first assign the type to the source form field and the target form field. See Figure 4.20(1)~(5). Then the users use the drag-and-drop to select source field, drag the mouse and drop the mouse upon the target form field. See Figure 4.20 (6)~(8).

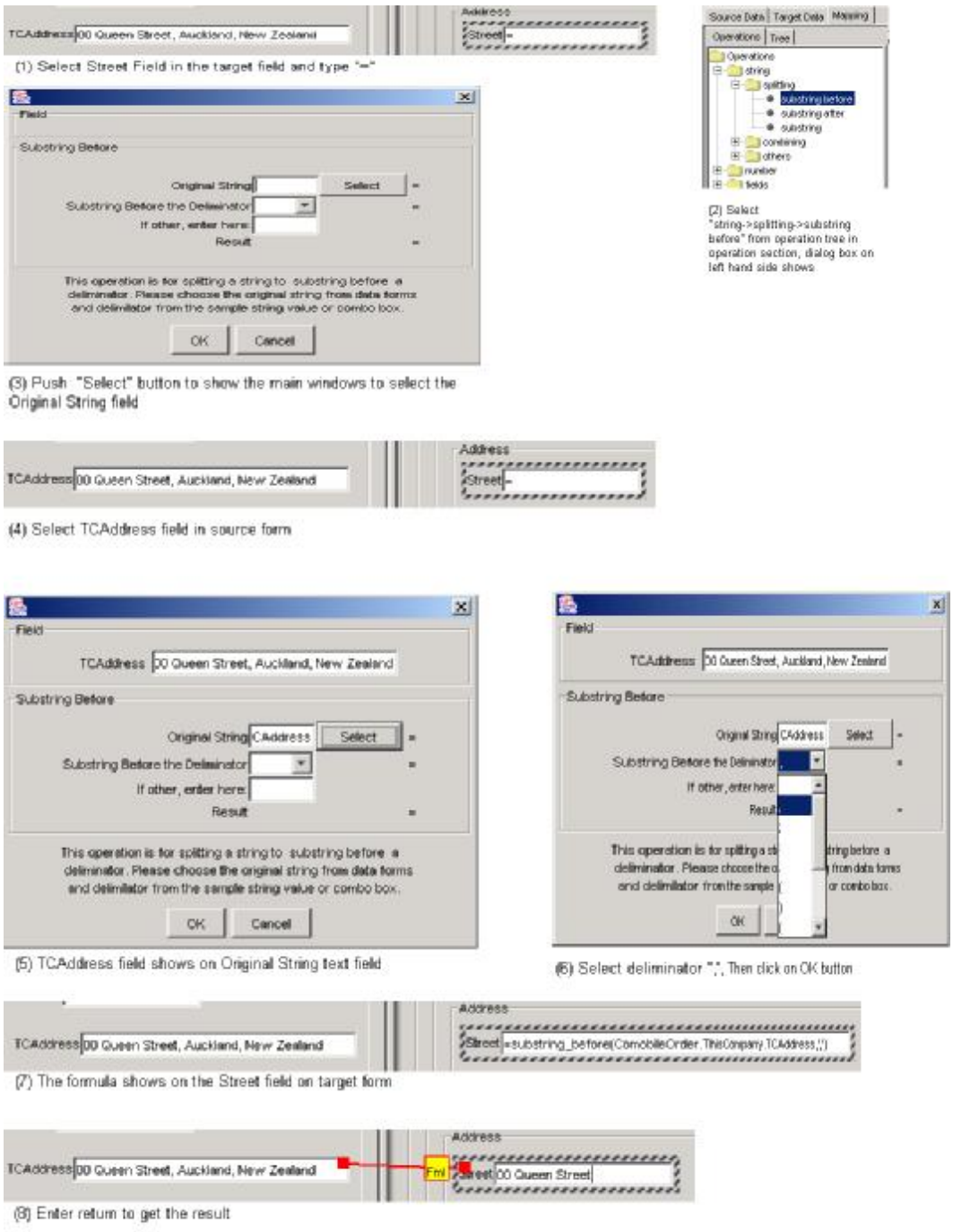


Figure 4.19 One-to-one formula non-typed mapping specification from CS order to TP order

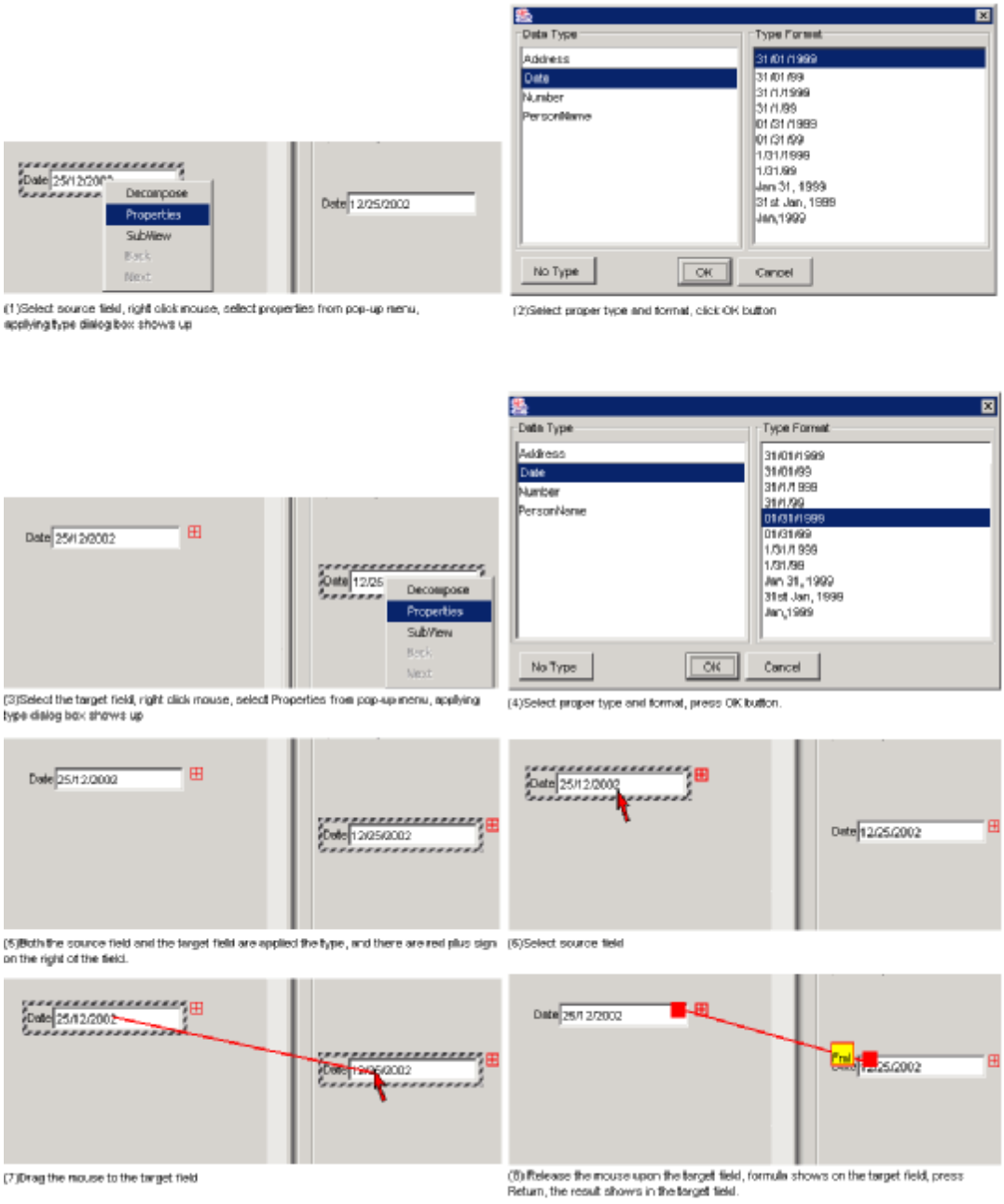
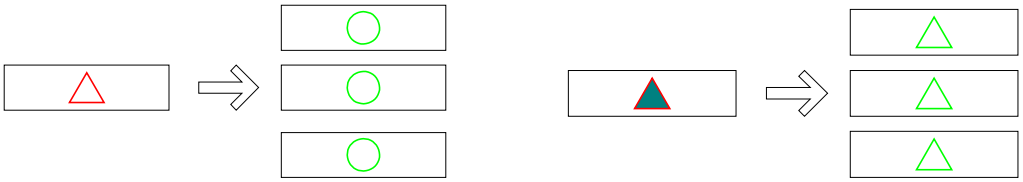


Figure 4.20 One-to-one formula typed mapping specification from CS order to TP order

- One-to-many

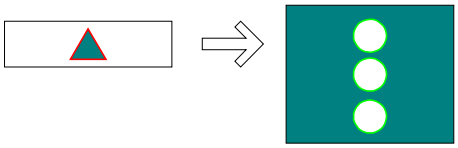
This one-to-many simple specification refers to a splitting operation, in which the value in one source field is splitted to several parts to target fields. An example of it is that the

TCAddress of *CS* order is splitted to three parts to *Street*, *Suburb*, *City*, *State*, *Zipcode* and *Country* fields in TP order form.



(1) Non-typed source and target fields

(2) Source -field-typed and target-field-non-typed



(3) Source-field-typed and target-section-typed

Figure 4.21 One-to-many simple mapping specification

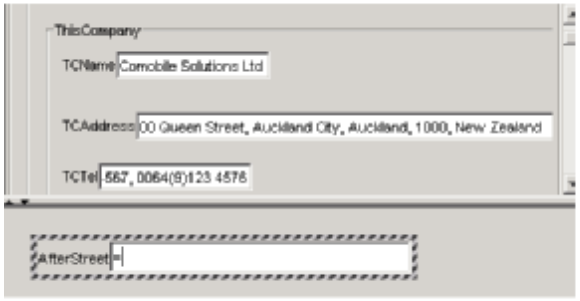
There are three ways to define the mapping specification. The first is no typed field and no typed section involved (see Figure 4.21(1)). The users just directly define the mapping specification between the form fields.

Figure 4.22 shows the procedures of defining the mapping specification for the above example. From the procedures illustrated in the figure, for each target field, the mapping define process is similar to one-to-one formula simple mapping.

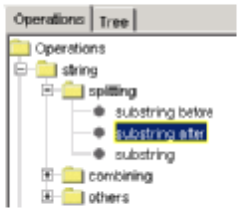
The second way is applying the type to the source field and then using the sub-form of the source field mapping to target field through one-to-one direct copy mapping. Figure 4.23 shows procedures of defining mapping specification in this way on the same example as the first way. We can see through applying the type on the source field, the mapping process becomes much easier.

The third way is applying the type to both the source field and the target section if all the target fields are within one form section and then mapping the typed source field to the typed target section through one-to-one formula mapping by drag-and-drop. Figure 4.24 shows procedures of defining mapping specification in this way on the same example as the former two ways. Again we can see through applying the type on both the source field and the target section, the mapping process becomes much easier.

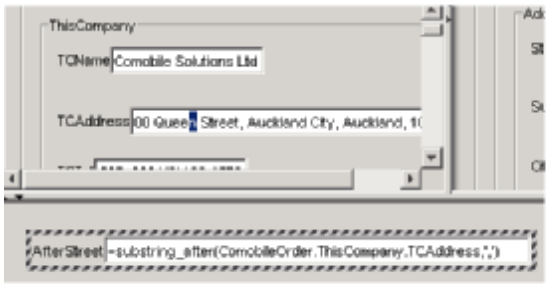
(1) Create a new cell by clicking the new-cell-button on the toolbar



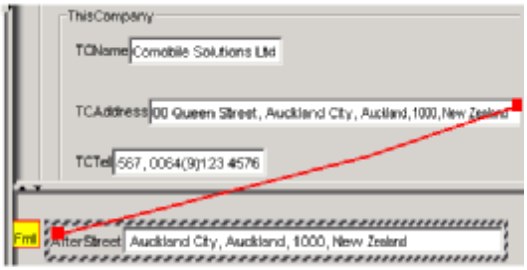
(2) Rename the cell to AfterStreet and input "=" in the text field of the cell



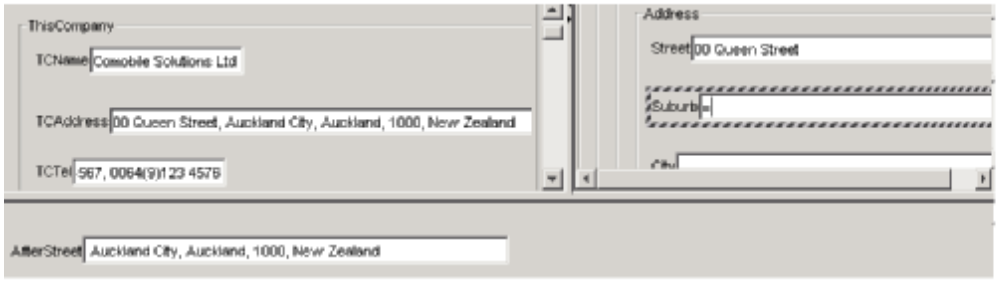
(3) select "substring after" operation from the operation tree



(4) After mapping specification in "substring after" dialog box is finished, the formular is shown on the AfterStreet text field



(5) Press return, the result of mapping specification is shown on the AfterStreet text field



(6)Define mapping specification of Suburb field by using intermediate result of AfterStreet

(7) After mapping specification of "substring_before" dialog box is finished, the maping specification is shown on Suburb text Field

Figure 4.22 One-to-many splitting for non-typed from CS order to TP order

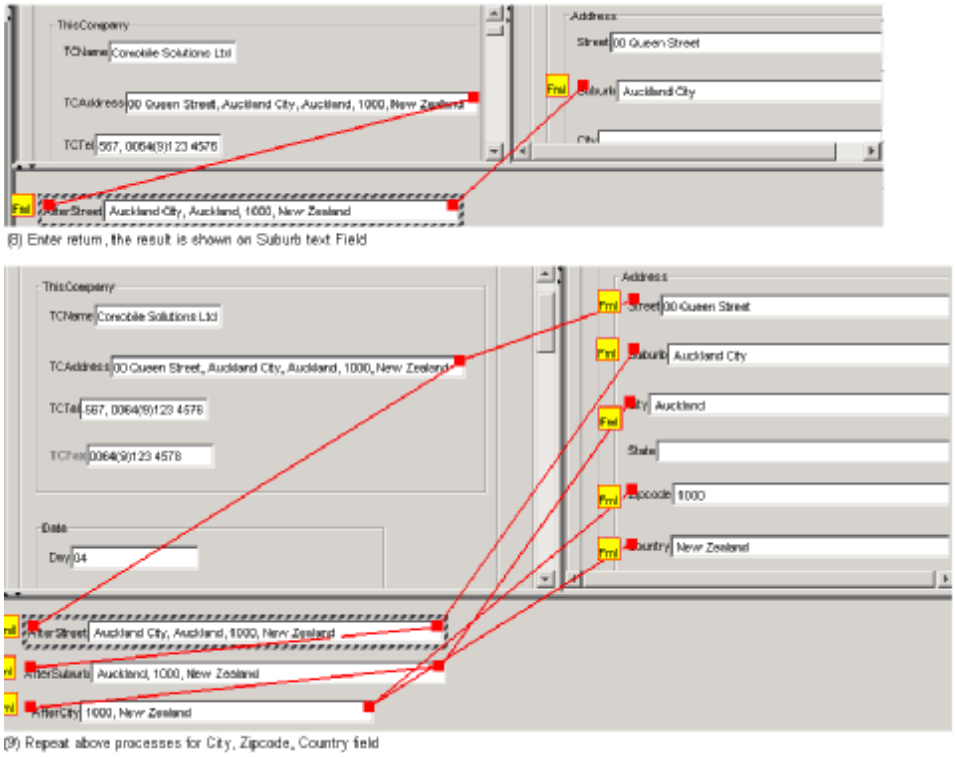


Figure 4.22 (continued)

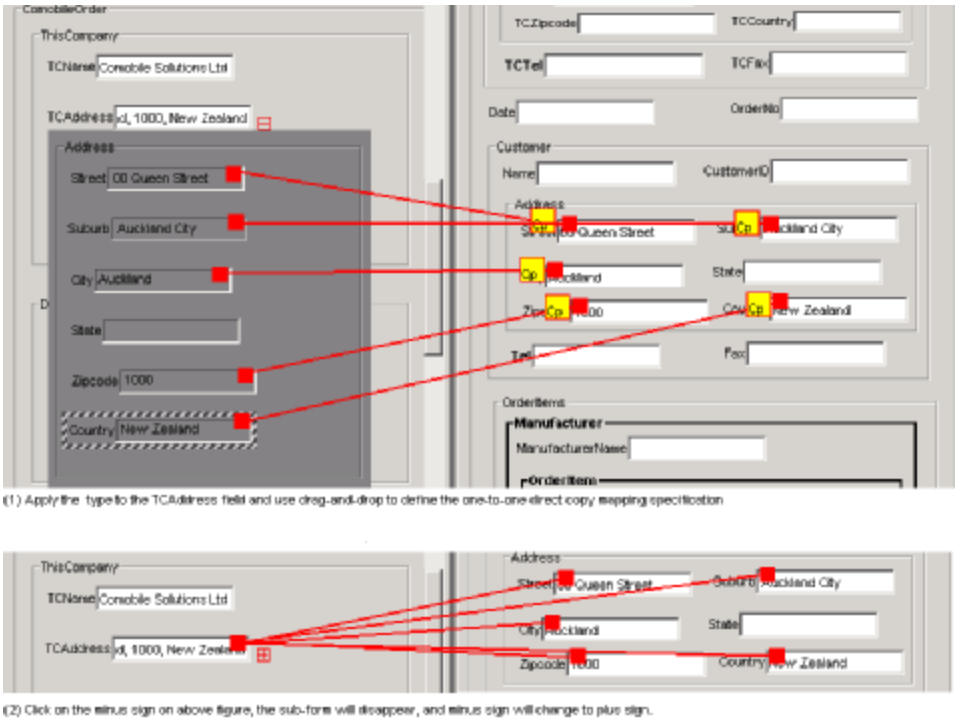


Figure 4.23 One-to-many splitting for the source-typed from CS order to TP order

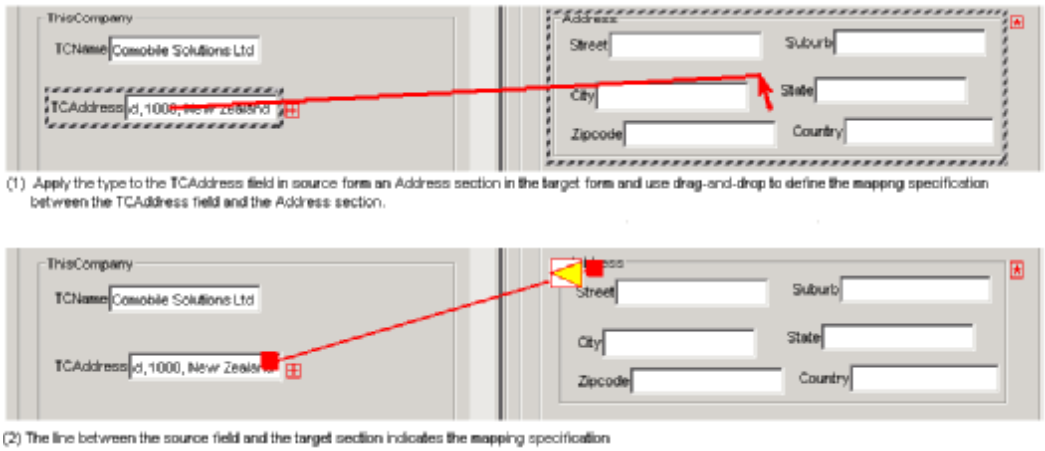
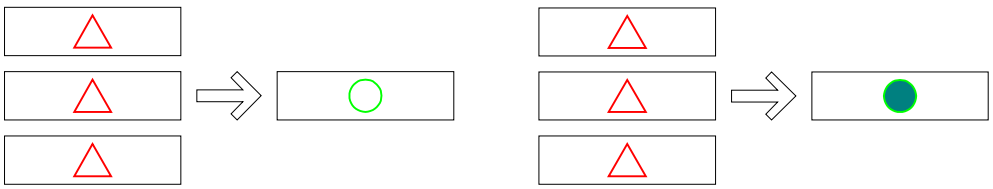


Figure 4.24 One-to-many splitting for the source-typed and target-typed from CS order to TP order

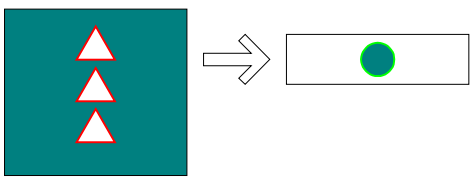
- Many-to-one:

This simple specification can be a combining operation, in which several fields are combined to one target field, or one target field with a formula, which involves several source fields. The formula can be the mathematic calculation, or string manipulation, or format conversion. An example of it is that *Year*, *Month* and *Day* fields in CS order are combined to *Date* field in target form.



(1) Source-field-non-typed and target-field-non-typed

(2) Source-field-non-typed and target-field-typed



(3) Source-section-typed and target-field-typed

Figure 4.25 Many-to-one simple mapping specification

There are three ways to define the mapping specification corresponding to one-to-many simple mapping specification, see Figure 4.25. The first is no typed field and no typed section involved (see Figure 4.25(1)). The users just directly define the mapping specification between the form fields. Figure 4.26 shows the procedures of defining the mapping specification for the above example.

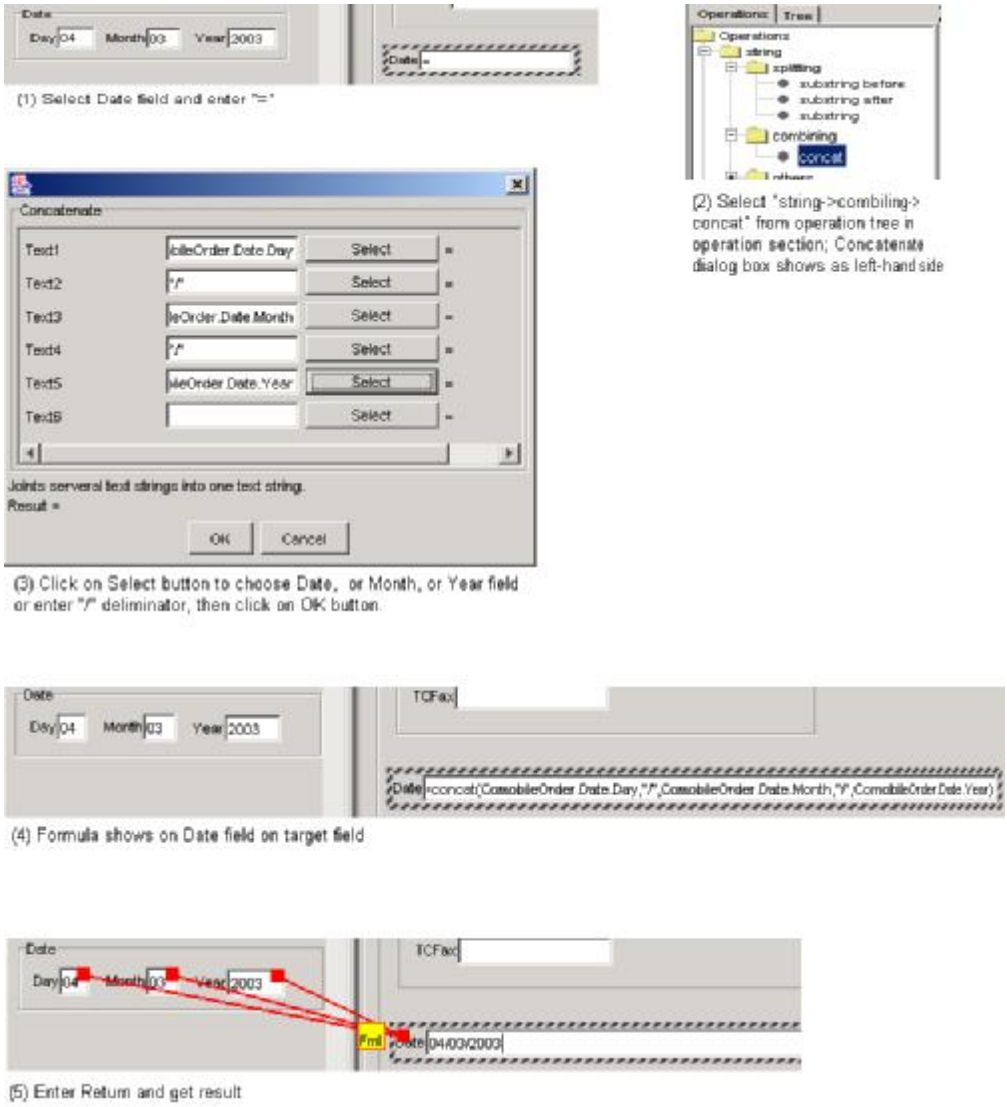
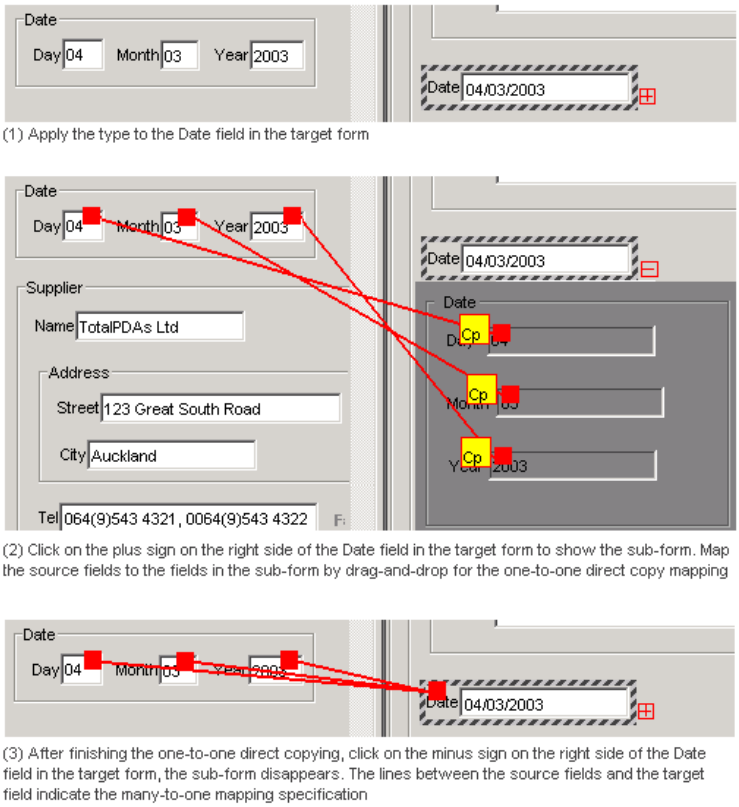


Figure 4.26 Many-to-one combination, non-typed source section and non-typed target field from CS order to TP order

The second way is applying the type to the target field and then using the sub-form of the source field mapping to target field through one-to-one direct copy mapping (see

Figure 4.25(1)). Figure 4.27 shows procedures of defining mapping specification in this way on the same example as the first way. We can see that through applying the type on the target field, the mapping process becomes much easier.



(1) Apply the type to the Date field in the target form

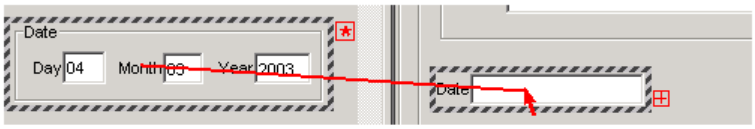
(2) Click on the plus sign on the right side of the Date field in the target form to show the sub-form. Map the source fields to the fields in the sub-form by drag-and-drop for the one-to-one direct copy mapping

(3) After finishing the one-to-one direct copying, click on the minus sign on the right side of the Date field in the target form, the sub-form disappears. The lines between the source fields and the target field indicate the many-to-one mapping specification

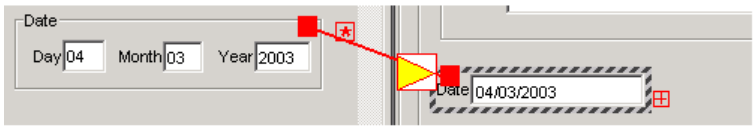
Figure 4.27 Many-to-one combination, non-typed source section and typed target field from CS order to TP order

The third way is applying the type to both the source section and the target field if all the source fields are within one form section and then mapping the typed source section to the typed target field through one-to-one formula mapping by drag-and-drop. Figure 4.28 shows procedures of defining mapping specification in this way on the same example as the former two ways. Again we can see through applying the type on both the source section and the target field, the mapping process becomes much easier.

(1) Apply the type to the Date section in the source form and the Date field in the target form



(2) Select the Date section in the source form and drag the mouse to the Date field in the target form and drop the mouse.



(3) The line between the Date section in the source form and the Date field in the target form indicates the mapping specification.

Figure 4.28 Many-to-one combination, typed source section and typed target field from CS order to TP order

- Many-to-many

Multiple source fields map to multiple target fields. This is normally for the source section, which contains multiple fields, map to the target section, which also contains multiple fields. Amount of the fields in the source section can be the same as, or different from the amount of the fields in the target section. The source section and the target section can be typed or non-typed (see Figure 4.29). All of these mapping specifications can be specified by drag-and-drop from the source section to the target section no matter that the source and target sections are typed or non-typed. The difference between the typed and non-typed sections is that the correspondent mapping fields are ordered in the non-typed (see Figure 4.29(1)(3)(5)(7)(9)(11)), while the correspondent mapping fields can be non-ordered in the typed (see Figure 4.29(2)(4)(6)(8)(10)(12)). If the fields in the source and target sections are non-typed, the mapping for the correspondent fields in source and target source is only one-to-one direct copy (see Figure 4.29(1~6)). If the fields in the source and target sections are typed, the mapping for the correspondent fields in source and target source can be one-to-one formula (see Figure 4.29(7~12)). For non-typed section mapping, the users must rearrange the source or target fields inside its section to make sure the order of the fields correct.

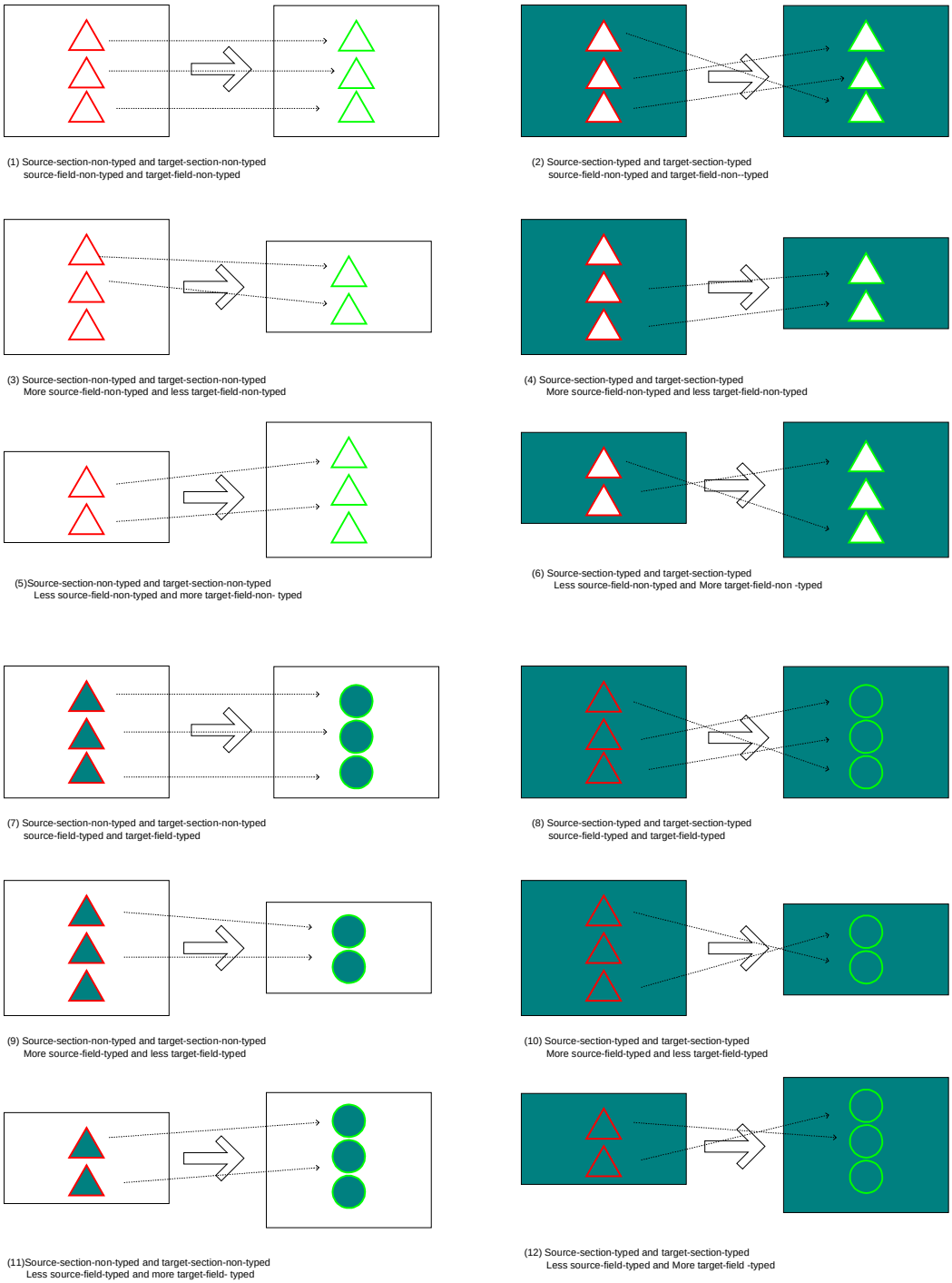


Figure 4.29 Many-to-many mapping specification

4.3.2.2.2 Complex mapping specifications

A complex mapping specifications involves one or more *collection* fields or sections in the source form or target form, or a mapping specification with a condition. The

collection field and section are a form field and a section with zero-*or-more*, *one-or-more* cardinality.

- One-to-many

One field in source is splitted to a collection field in target form. For example, value of *TCTel field* of *ThisCompany* in *CS* order needs to be splitted to multiple records of *Tel* field of *Customer* section in *TP* order form. When mapping the relation, the users first need to know how to split the non-collection source field. Normally there are deliminator to separate the value in the source field to many chunks. So the users need to identify the deliminator to separate the value to collection of many chunks. Then the users need to ask if all the chunks are needed to compose the collection of the target field. If not, the users need to specify the condition to filter chunks. Then the users need to transform the chuck to the required element of the target collection and finally sort them in the correct order (see Figure 4.30). The later three steps involve the many-to-many mapping specification we will discuss later.

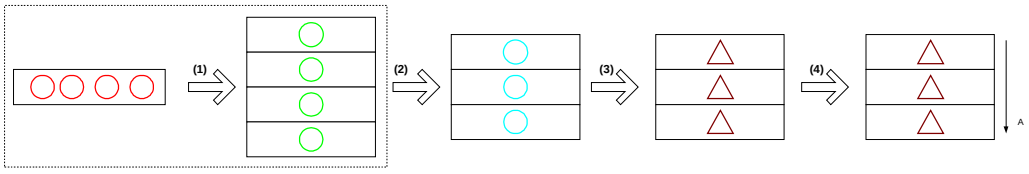


Figure 4.30 One-to-many complex mapping specification process

Figure 4.31 shows the procedures to directly split the value in the *TCTel* field of *ThisCompany* in *CS* order form to multiple records of *the Tel* field of *Customer* section in *TP* order form. In this mapping, there is no chunk filtering and other transformation required.

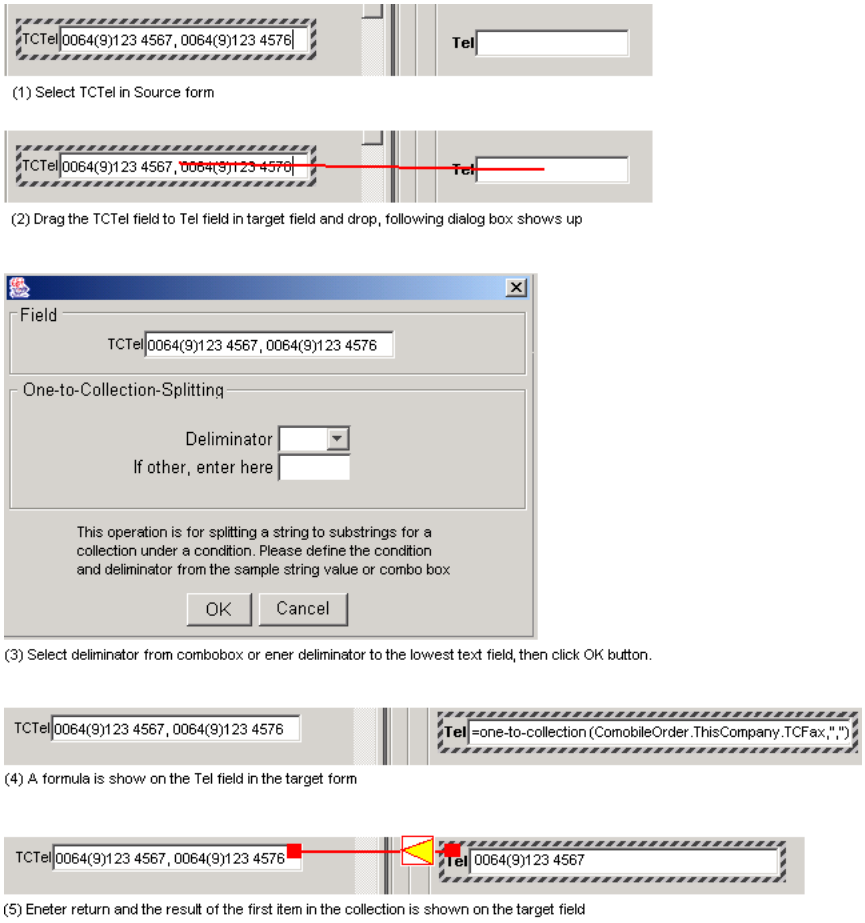


Figure 4.31 One-to-many complex data mapping from CS order to TP order

- Many-to-one:

In the many-to-one mapping specification, values in the collection field in source form are combined to one value of the field in target form, or one record in a specific position is extracted from the collection and mapped to one field in target form. For example, all the records of *TCFax* field in *ThisCompany* section in *CS* order form need to be combined to one record and mapped to the *Fax* field of *Customer* in *TP* order form. For the first case in this mapping specification, normally the users need to first filter the records of the source collection fields, and then convert each of the value to required format, and then sort them, finally combine them together to the field of target form (see Figure 4.32). The former three steps involve the many-to-many specification we will describe later.

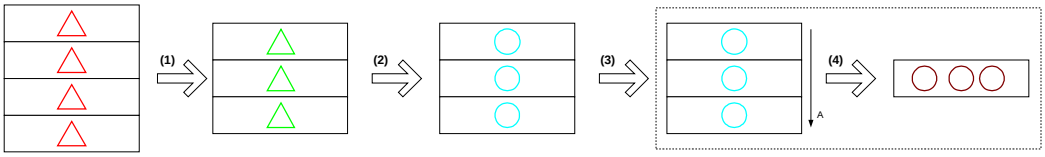


Figure 4.32 Many-to-one complex mapping specification process

Figure 4.33 shows procedures of all the records of *TCFax* field in *ThisCompany* section in *CS* order form being combined to one record, and mapped to the *Fax* field of *Customer* in *TP* order form. There are no records filtering and records transformation involved in this mapping specification.

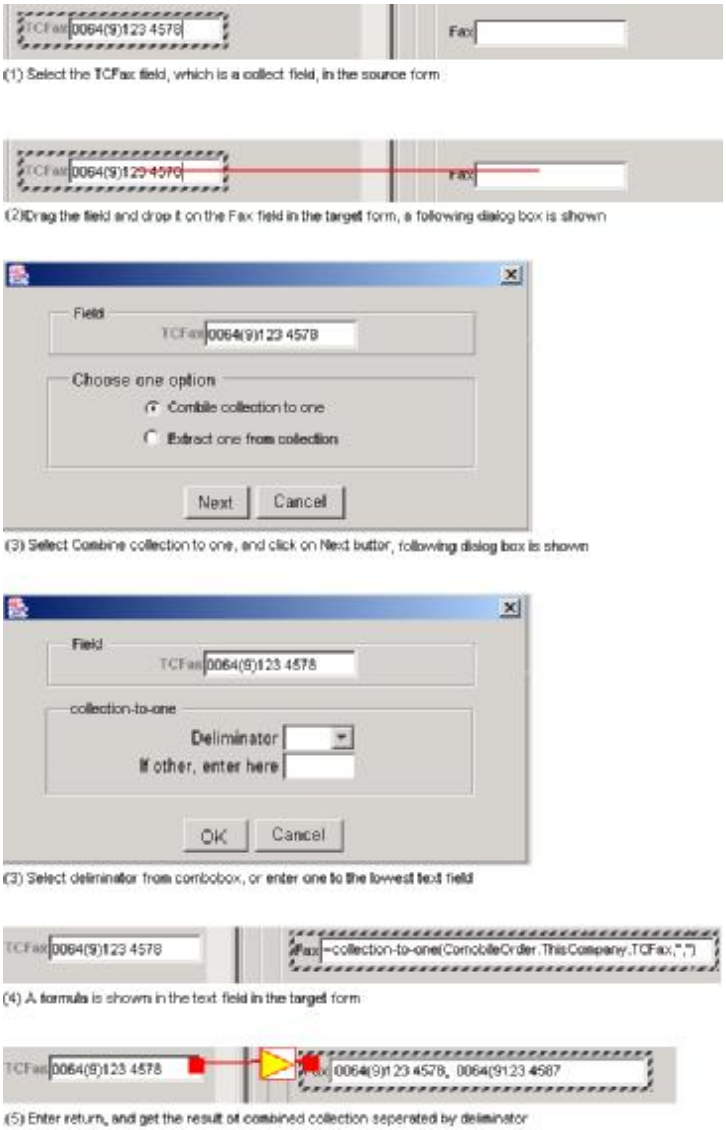


Figure 4.33 Many-to-one complex data mapping from CS order to TP order

- Many-to-many:

The many-to-many complex mapping specification refers to a collection field in the source form mapping to a collection field in the target form. In this mapping specification, normally the users need to first decide the scope of iteration of the source collection (see Figure 4.34(1)), and then decide if the source collection need to be filtered to the target collection and how they to be filtered (see Figure 4.34(2)), and then specify how the data in each of the records in the source collection to be mapped to that of the target collection (see Figure 4.34(3)), and finally the result of the target collection may be sorted (see Figure 4.34(4)).The third step is relevant to the individual record mapping, i.e. simple mapping we have already discussed. In the following, the collection-level-related first step, second step and final step are discussed.

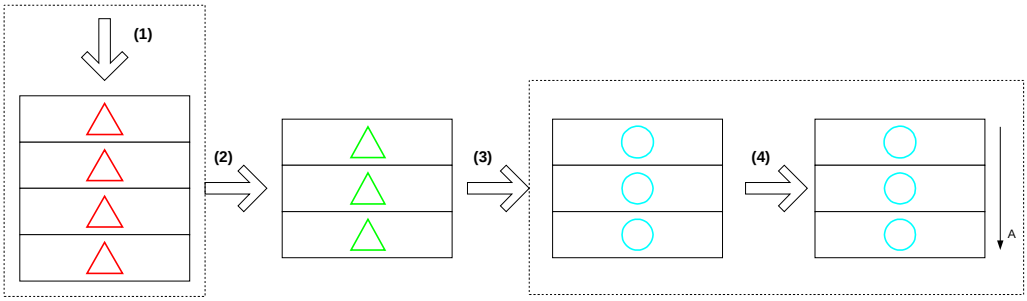


Figure 4.34 Many-to-many complex mapping specification

The many-to-many mapping specification can be defined by both drag-and-drop and type-and-selected. The scope of iteration of source collection tells where we should start the collection iteration when there are nested collections in the source form. By default, when the users drag and drop from a field in a source collection to a field in the target collection, or from a collection field or a collection section in the source form to a collection field or a collection section in the target form, or type in a target field in a collection and select a field in the source collection, if the collection-level mapping has not be defined yet, the system will automatically define the collection-level mapping as setting the scope of iteration of source collection to iterating all elements from the root collection to the leaf collection, no condition or an operation for filtering, and no sorting. If the default collection-level mapping does not satisfy the users' need, the users can further modify the collection-level mapping properties on the collection-level

mapping properties dialog box on which the nested collections, filters, operations and sorting for the each collection are listed.

Let take the example of the mapping specification for the *ManufacturerName* field in the *Manufacturer* collection in the TP order form (see Figure 4.36). From the context, the target collection comes from iterating all the *OrderItem* in all the categories, and then finding all not repeated manufacturers and sorting by the manufacturer name. The *Manufacturer* field in the *OrderItem* collection in the CS order form has a grandparent *Category* collection. To define the mapping specification, we first start to create a temporary collection cell named AllManufacurer to collect all the manufacturers in one collection and then we map the *Manufacturer* field in the *OrderItem* collection in the CS order form to it by drag-and-drop from the later field to former field (see Figure 4.36(1)). The default collection-level mapping and field-level direct copy are defined. The blue line and red line show the two level mapping (see Figure 4.36(2)). That's all we want for this mapping.

Then we create another new collection cell named *NoRepeatedManufacturer* to collect no repeated manufacturer names. Apply drag-and-drop again between *AllManufacturer* field and the new created field. Default settings are applied to the mapping (see Figure 4.36(2)). But for collection level mapping, we want to apply a collection operation "Normalize" to delete all the repeated manufactures in the source collection. So double click on the blue line and edit the dialog box (see Figure 4.36(3)). We uncheck the default setting, check the Collection Operation check box, select Normalize operation from the list of the combo box, and click OK button.

Finally we apply drag-and-drop again between the *NoRepeatedManufacuturer* collection field and *ManufacturerName* field in the target form. Default settings are applied to the mapping (see Figure 4.36(4)). But for collection-level mapping, now we want to sort the NoRepeatedManufacturer in ascending order. So double click on the blue line and edit the dialog box (see Figure 4.36(5)). We uncheck the default setting, check the Sort by check box, select field we want to sort, click on the ascending radio box and click OK button. So far, we have finished the mapping specification between the *Manufacturer* field in the *OrderItem* collection in the source form and the ManufacturerName field in the Manufacturer collection in the target form (see Figure 4.36(6)).

The top combo box on the collection-level mapping dialog box(see Figure 4.36(3)) lists all the nested source collections for the mapping specification. If the users want to modify the default scope of the iteration, the users can choose the collection from the combo box, and then edit the rest of content on the dialog box.

- Other many-to-many

It refers to combining two or more collections (see Figure 4.35).

Figure 4.35 shows procedures for combining two collections each of which only contain a field. The first filtering step, the third converting step and the fourth sorting step have been described in previous sections. The second step is combination of filter collections. It can be performed by selecting all the filtered collection fields, and then dragging and dropping them to a target collection field.

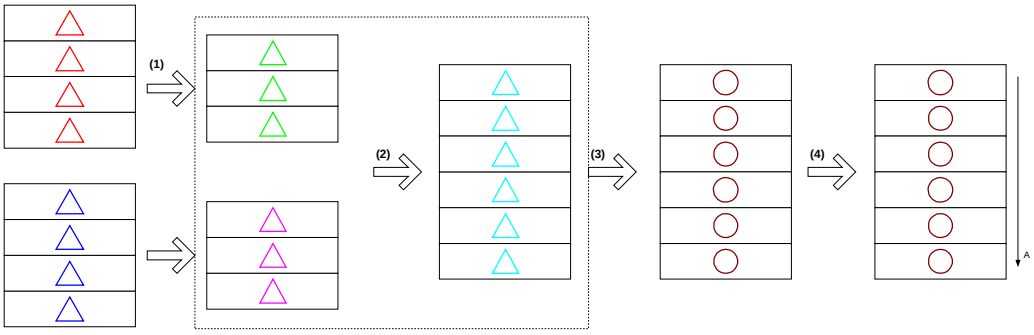
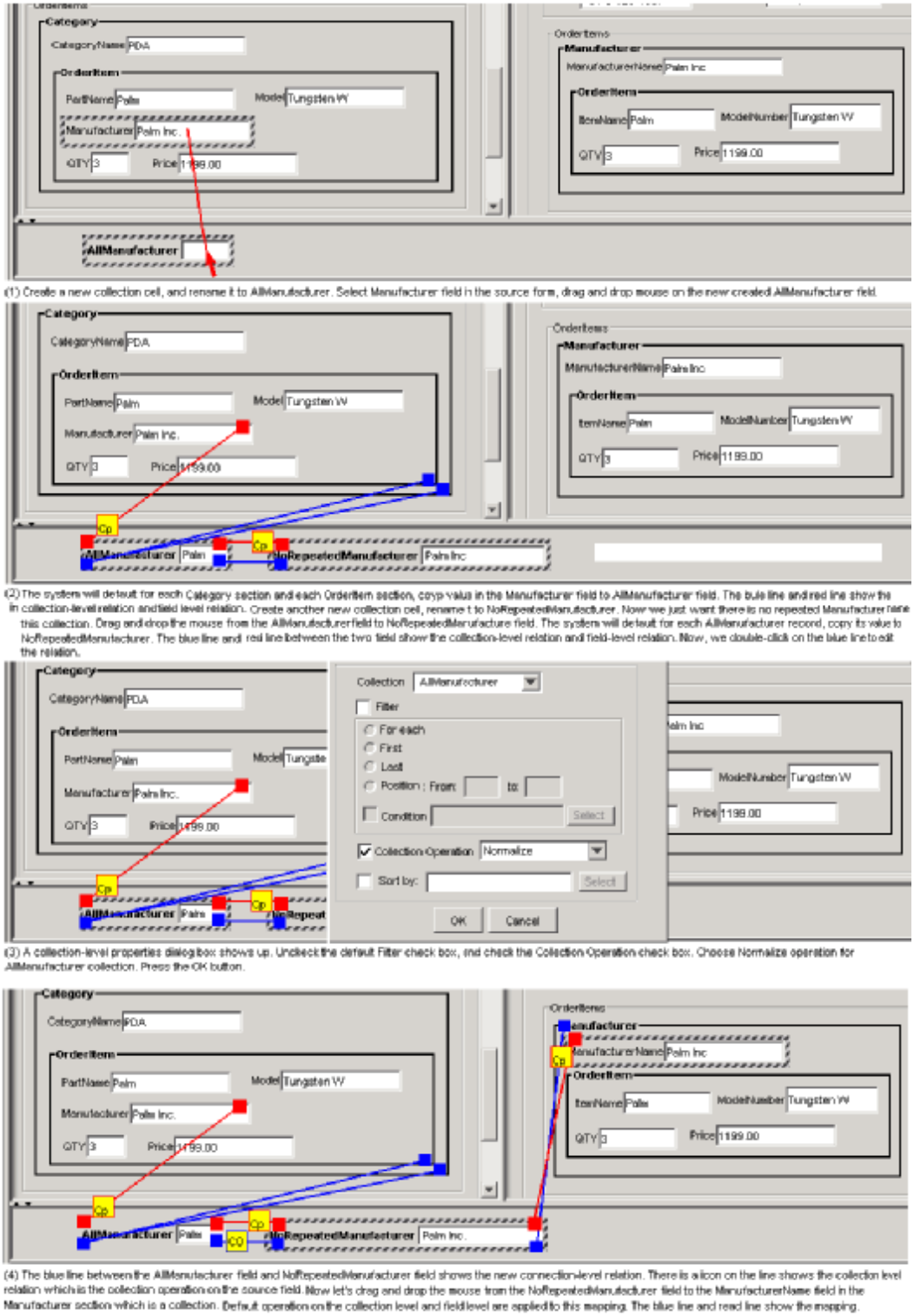


Figure 4.35 Combine two collections



(1) Create a new collection cell, and rename it to AllManufacturer. Select Manufacturer field in the source form, drag and drop mouse on the new created AllManufacturer field.

(2) The system will default for each Category section and each OrderItem section, copy value in the Manufacturer field to AllManufacturer field. The blue line and red line show the in collection-level relation and field level relation. Create another new collection cell, rename it to NoRepeatedManufacturer. Now we just want there is no repeated Manufacturer name this collection. Drag and drop the mouse from the AllManufacturer field to NoRepeatedManufacture field. The system will default for each AllManufacturer record, copy its value to NoRepeatedManufacturer. The blue line and red line between the two field show the collection-level relation and field-level relation. Now, we double-click on the blue line to edit the relation.

(3) A collection-level properties dialog box shows up. Uncheck the default Filter check box, and check the Collection Operation check box. Choose Normalize operation for AllManufacturer collection. Press the OK button.

(4) The blue line between the AllManufacturer field and NoRepeatedManufacturer field shows the new connection-level relation. There is a icon on the line shows the collection level relation which is the collection operation on the source field. Now let's drag and drop the mouse from the NoRepeatedManufacturer field to the ManufacturerName field in the Manufacturer section which is a collection. Default operation on the collection level and field level are applied to this mapping. The blue line and red line show the mapping.

Figure 4.36 Many-to-many complex data mapping from CS order to TP order

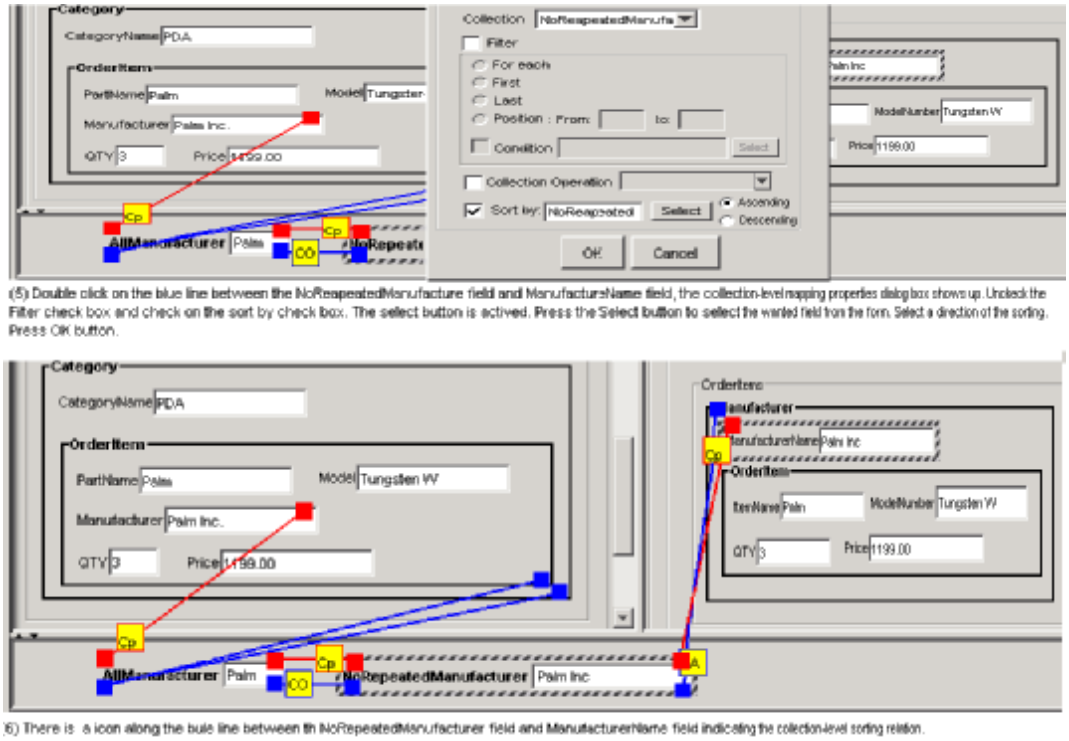


Figure 4.37 (continued)

- Conditional mapping

For the above simple and complex mapping specifications, they may be mapped conditionally. The conditions we provide are *If-Then*, *If-Then-Else* and *If-ElseIf-…-Else*.

If one field or section has different formula definition when different conditions are satisfied, the conditional mapping needs to be defined by type-and-select. That is, choose the target field and type "=" sign, and then select the proper conditional operation from the operation area. Then the condition dialog box shows up. Figure 4.37 shows a dialog box for *If-Then-Else* mapping definition. Now we start build the condition expression and formulae when the condition is true and false by type-and-select. To do this, the users type numbers, or mathematical symbols, or some logical symbols. When needing the operations or a field reference, the users press *Select* button to select the field or operations from the mapping specification environment. When definition of the expression or the formulae is finished, the result of the expression or the formulae will show following the "=" sign beside the *Select* button.

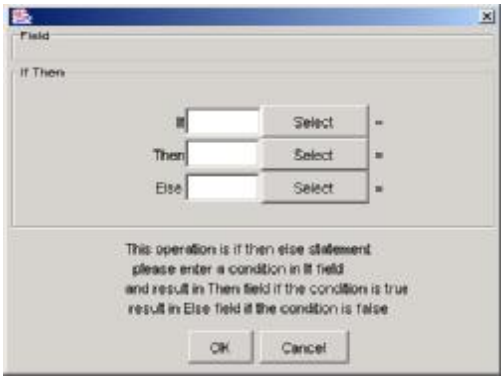


Figure 4.37 Conditional mapping

Sometimes there are cases that we have already know the context and want to add a condition to them, for example, when the formula of field and section has already be defined, we want to add a condition to the formula, or in the case of the many-to-many filtering, we want to add a condition to filter the source collection to the target collection.

4.4 Object-oriented Design

According to the requirements of our mapping tool, we have identified the main components of our system, which are UI, converters, a form generator, code generator and transformation engines, and have selected 2-teried architecture for our system. In the following sections, a documentation of the object-oriented design for the standalone system is presented. It includes the main class diagrams of the main modules, and sequence diagrams for main operations.

4.4.1 User Interfacing

The user-interfacing module is responsible for

- Providing the interface to the users;
- Accepting the users' commands, such as creating a new project, save project, creating new operation, generating implementation code etc, and invoking other correspondent modules etc;
- Getting information about source and target schemas and instance from the users' input;
- Letting the user customize the form
- Letting the user define the mapping specification

Its class diagram is shown on Figure 4.38.

The type and description of the classes of client application are summarized in Table 4.1.

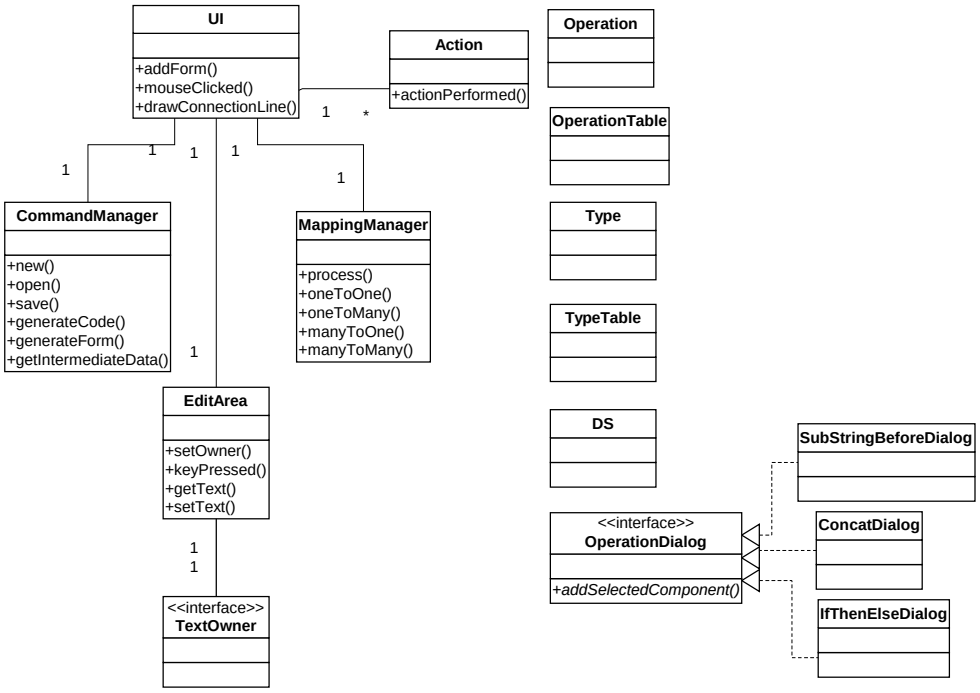


Figure 4.38 Class diagram for user interfacing

Class	Description
UI	For providing a user interface for the user of application, get input from the user
CommandManager	For processing user commands, such as file command: new, open, save; edit command: cut, copy, paste; tool command: generate code; etc.
MappingManager	For processing different mapping type, such as one-to-one, one-to-many, many-to-one and many-to-many
TextOwner	Interface to define the owner of the EditArea
EditArea	An edit area which can create, edit, delete mapping specification
Operation	For defining the operation
OperationTable	Table which contains operation objects
Type	For defining the type
TypeTable	Table which contains type objects
DS	For data structure
OperationDialog	Interface for operation dialog box
ConcatDialog	Dialog box for concatenating strings
SubstringDialog	Dialog box for splitting string
IfThenElseDialog	Dialog box for if-then-else condition statement
Action	Abstract class

Table 4.1The type and description of the classes of user interfacing

4.4.2 Converter

A converter module is responsible for source and target schemas and instances, convert them to the intermediate data structure. It contains the schema parser and instance parser. In order to make the system not depend on the certain parser, a ConverterFactory class is used to serve the purpose.

Its class diagram is shown on Figure 4.39.

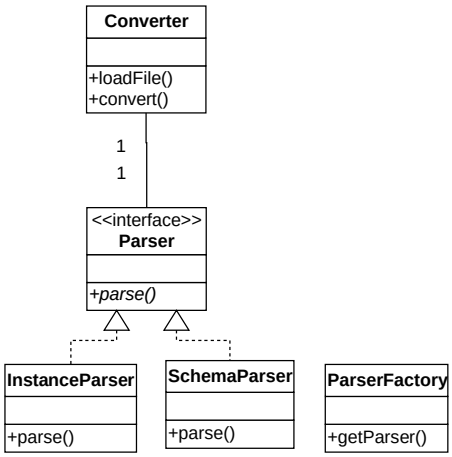


Figure 4.39 Class diagram of converter

The type and description of the objects of converter are summarized in Table 4.2.

Class	Description
Converter	A thread for process client requests
ParserFactory	A factory which produce the different parsers for schemas
Parser	Interface for various parser
SchemaParser	A parser for parsing a schema file to an intermediate data structure
InstanceParser	A parser for parsing an instance to an intermediate data structure

Table 4.2 The type and description of the classes of converter

4.4.3 Form Generator

It is responsible for taking an intermediate data model from the converter, automatically generates forms and imports sample data to the forms. Its class diagram is shown on Figure 4.40.

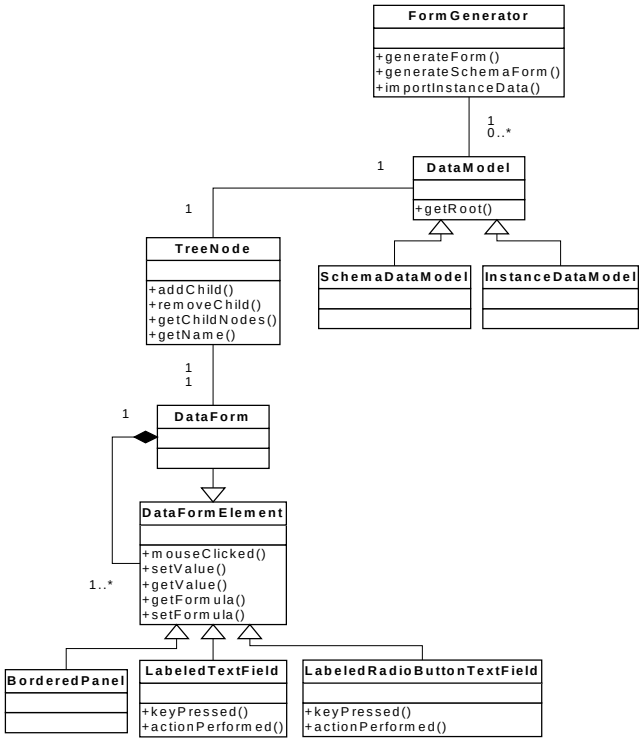


Figure 4.40 Class diagram of form generator

The type and description of form generator are summarized in Table 4.3.

Class	Description
FormGenerator	Generate data form
DataModel	For defining common intermediate tree structure which stores and processes information about data schema and data instance, is a super class of Schema DataModel and InstanceDataModel
TreeNode	For defining attributes and behaviour of node which is a basic node of tree structure of DataModel
SchemaDataModel	For defining tree structure which stores and processes information about data schema
InstanceDataModel	For defining common tree structure which stores and processes information about data instance
DataForm	A panel which is a form element and contains form elements
DataFormElement	Abstract class for form elements
BorderedPanel	A form element, a panel which has a border
LabeledTextField	A form element, a text field which has a label
LabeledRadioButtonTextField	A form element, a text field which has a label and a radio button

Table 4.3 The type and description of the classes of form generator

4.4.4 Code Generator

A code generator accepts the definition of mapping specification, to generate the required mapping specification implementation, and then sends the generated mapping specification implementation and a source instance as inputs of transformer to generate the target result.

Its class diagram is shown on Figure 4.41.

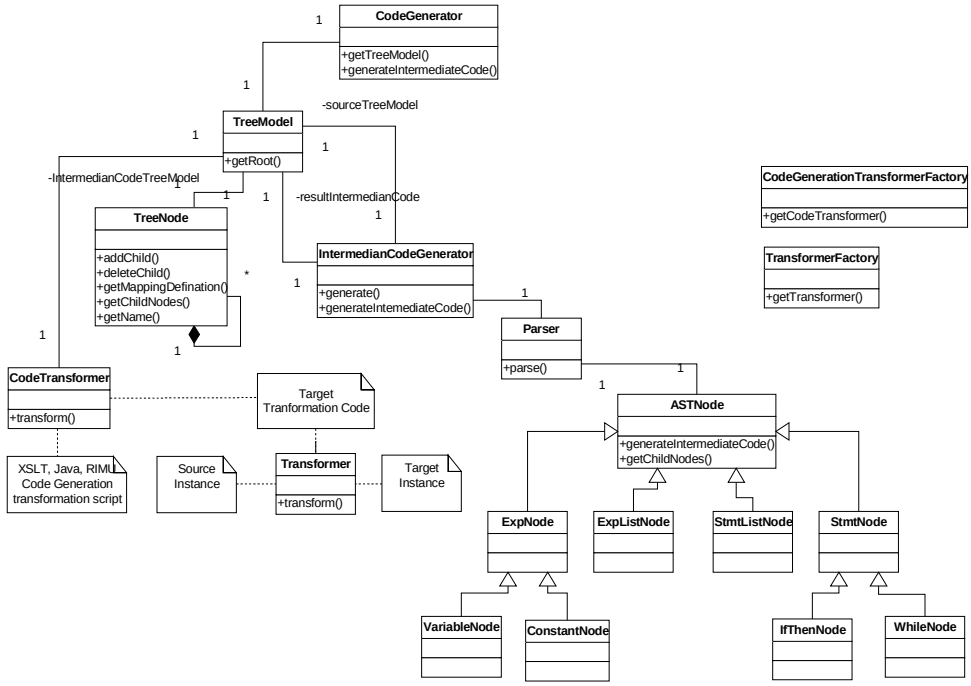


Figure 4.41 Class diagram of code generator module

The type and description of the objects of code generation server are summarized in Table 4.4.

4.4.5 Sequence Diagrams for Some Main Operations

- Create a new project

Figure 4.42 shows a sequence diagram of creating a new project.

Classes	Description
CodeGenerator	For generating the mapping specification implementation
TreeModel	For defining common intermediate tree structure which stores and processes information about the mapping specification
TreeNode	For defining attributes and behaviours of node which is a basic element of tree structure in TreeModel
IntermediateCodeGenerator	For generating an intermediate code from mapping specification which is in a intermediate tree structure
Parser	For parsing user-defined mapping specification for a target field
CodeTransformer	For transforming an intermediate mapping specification to a finally mapping specification implementation
Transformer	For transforming an source instance to target instance according to the mapping specification implementation
MessageParser	For parsing message from loader server
CodeTransformerFactory	A factory class for producing different transformer intermediate code to finally mapping specification implementation
TransformerFactory	A factory class for producing different transformers for transforming a source instance to a target instance
ASTNode	An abstract syntax tree node
ExprNode	An abstract class for AST node for expression
ExprListNode	AST node for expression list
StmtNode	An abstract class AST node for statement node
StmtListNode	AST node for statement node
VariableNode	An expression node for variable
ConstantNode	An expression node for constant
IfThenElseNode	An statement node for if-then-else statement
WhileNode	An statement node for while statement

Table 4.4 The type and description of the classes of code generator

The detailed sequence called is described in Table 4.5.

Sequence	Description
actionPerformed()	The users give the new command action
new()	CommandAction call the commandManager new function to create new tabbed panels
createSourceInforTabbedPanel()	Create source data information tabbed panel
createTargetInforTabbedPanel()	Create target data information tabbed panel
createMappingTabbedPanel()	Create mapping tabbed information tabbed panel
addNewCreatedTabbedPanels()	Add the above new created tabbed panel to the UI

Table 4.5 Meaning of sequence call in creating a new project

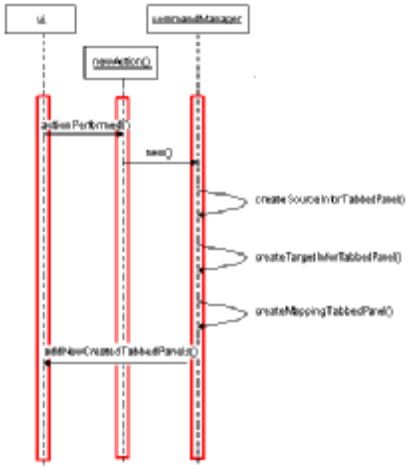


Figure 4.42 Sequence diagram of creating a new project

- Convert a schema and Generate a form

Figure 4.43 shows a sequence diagram of converting a schema and generating a form.

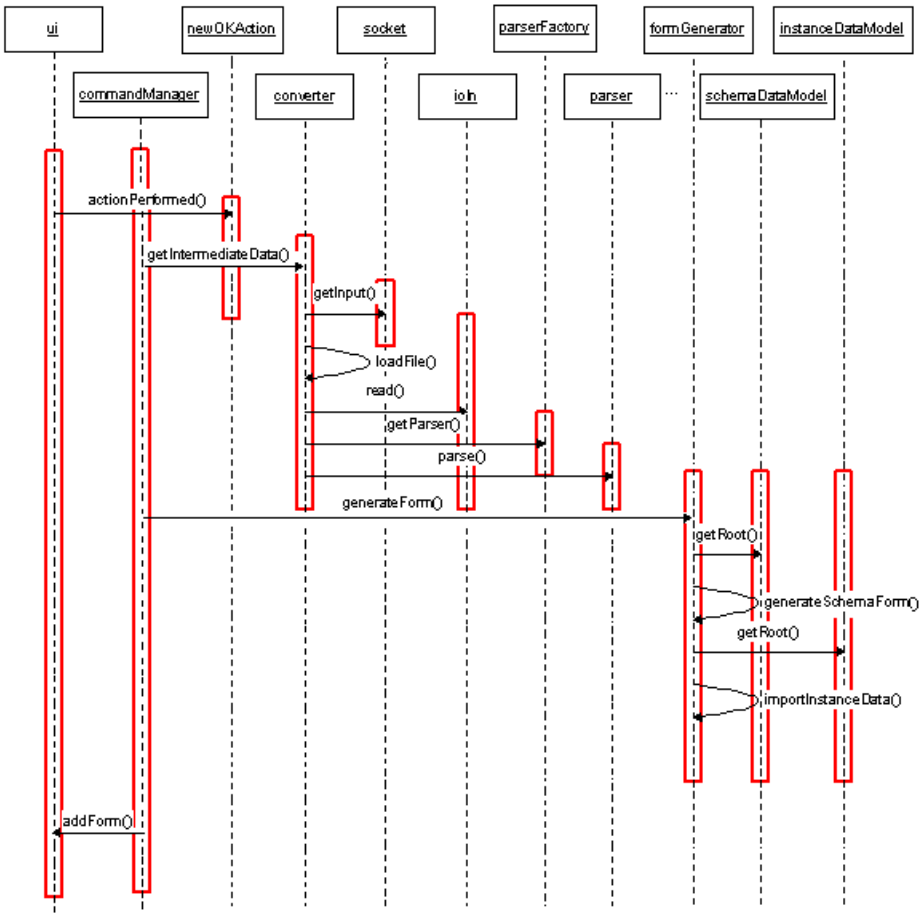


Figure 4.43 Sequence diagram of converting process and form generation

The detailed sequence called is described in Table 4.6.

Sequence	Description
ActionPerformed()	The users press on the OK button of the source data information panel OK button and invoke the actionPerformed() function
GetIntermediateData()	The actionPerformed call the commandManager to invoke the converter's getIntermediateData () function to produce the intermediate data
GetInput ()	Converter calls the schemaSocket or instanceSocket input read() operation to read the message of schema or the instance
LoadFile()	Converter calls itself loadFile() to load a schema file or an instance from the location the users specified
Read()	Converter read the content of the schema or instance from IO
GetParser()	Converter calls parser factory to produce a parser for the certain schema or instance
Parse()	Converter calls the parser to parsing the message of schema or the instance to a intermediate tree structure
GenerateForm()	CommandManager calls formGenerator generateForm() operation to generate form
GetRoot()	FormGenerator calls schemaDataModel getRoot() to get schema root tree node
GenerateSchemaForm()	FormGenerator calls its generateSchemaForm() operation to generate form
GetRoot()	FormGenerator calls instanceDataModel to getRoot() to get instance root tree node
importInstanceData()	FormGenerator calls its importInstanceData() operation to fill in the instance data in the form
AddForm()	Command manager call UI addForm() operation to add the generated form to main windows

Table 4.6 Meaning of sequence call in converting process

- Define a one-to-one copy mapping specification

Figure 4.44 shows a sequence diagram of defining a one-to-one copy mapping specification.

The detailed sequence called is described in Table 4.7

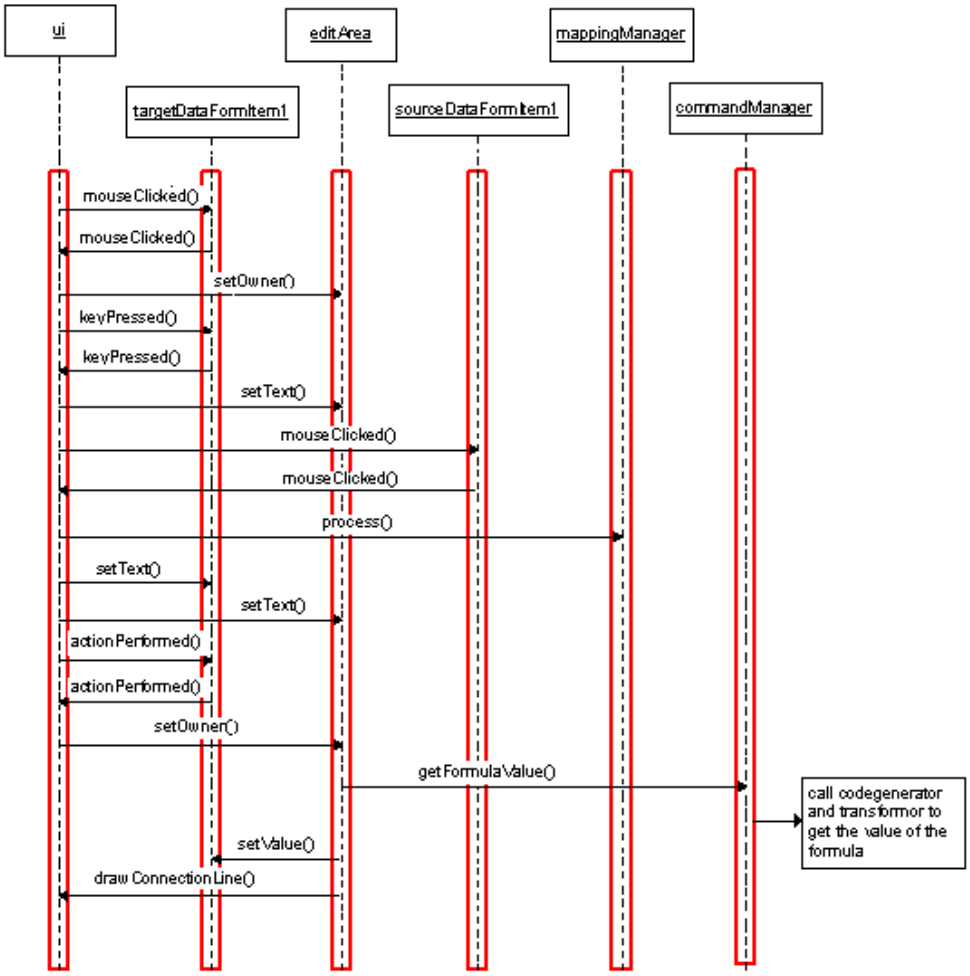


Figure 4.44 Sequence diagram for defining one-to-one mapping specification

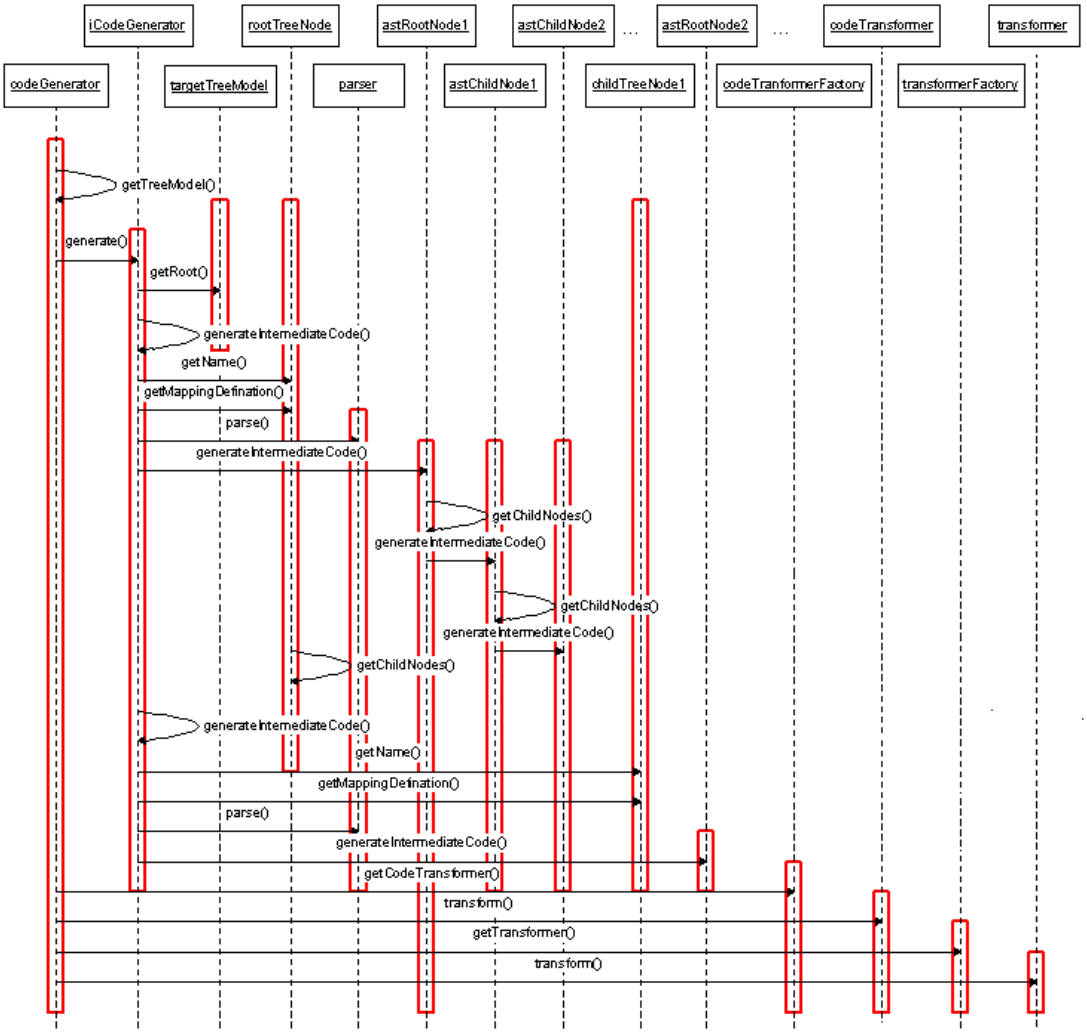


Figure 4.45 Sequence diagram of code generation and debugging process

Sequence	Description
MouseClicked()	The user clicks on one target field, the field mouseClick() operation is invoked
MouseClicked()	Call back UI mouseClicked()
SetOwner()	The UI calls the editArea setOwner() operation to set the owner of the editArea to the target form item
KeyPressed()	The user enters "=" invoking editArea keyPressed() operation to make the editArea to a select mode in which the user is allowed to a select form field from the source form, target form, or self-defined form, then a path reference in the formula in editArea will point to that field
KeyPressed()	Call back UI KeyPressed()
SetText()	Set the editArea text as the text in the form field
MouseClicked()	The user clicks on desired source form item, the form item mouseClick() operation in invoked
MouseClicked()	Call back UI mouseClicked()
Process()	The UI calls mappingManager process() operation to give the next response according to cardinality of the source and target form item. For one-to-one mapping, the mappingManager will just add the path reference to the source form item in the formula in the target form item
SetText()	Update the formula in the target form field
SetText()	Update the formula in the editArea
ActionPerformed()	After the formula definition is finished, press return to invoke following action
ActionPerformed()	Call back UI actionPerformed()
SetOwner()	The UI calls the editArea setOwner() operation to set the owner of the editArea to the current selected target form item
GetFormulaValue()	Call commandManager to get the formula value
SetValue()	The editArea calls the previous target form item setValue() operation to show the result on the form item field
DrawConnectionLine()	The editArea calls the UI drawConnectionLine() operation to draw the connection line between the current target form field and source form fields which have a formula association with the target form field

Table 4.7 Meaning of sequence call in defining one-to-one mapping specification

- Generate code

Figure 4.45 shows a sequence diagram of code generation process after the user gives a code generation command.

The detailed sequence called is described in Table 4.8.

Sequence	Description
GetTreeModel()	CodeGenerator calls itself function to get the targetTreeModel which contains target structure and mapping specifications
Generate()	CodeGenerator calls intermediate code generator—iCodeGenerator—generate() function with target tree mode as a parameter to generate the intermediate code
GetRoot()	Get the root tree node by calling target tree model getRoot() function
GenerateIntermediateCode()	Call internal function to generate intermediate code by taking the root tree node as a parameter
GetName()	Get the root name
GetMappingDefinition()	Get the mapping specification of the root node
Parse()	Parse the root mapping specification to get the AST
GenerateIntermediateCode()	Generate intermediate code for the AST
GetChildNodes()	Get the child node of root node of AST
GenerateIntermediateCode()	Generate intermediate code for the child node
GetChildNodes()	Get the child node of the above child node
GnerateIntermediateCode()	Generate intermediate code for the child node
	… Recursively until the leaf node is reached
GetChildNodes()	Get the child node of root node of treeModel
GenerateIntermediateCode()	Generate the intermediate code for the child node
GetName()	Get the name of the child node of treeModel
GetMappingDefinition()	Get the mapping specification of the child node
Parse()	Parse the mapping specification to get the AST
GenerateIntermediateCode()	Generate intermediate code for the AST
	… Recursively until the leaf node of the treeModel is reached
GetCodeTransformer()	Get the intermediate code transformer
Transform()	Transform the intermediate code to the required mapping specification implementation
GetTransformer()	Get the transformer for transforming source data instance to target data instance
Transform()	Transforming source data instance to target data instance

Table 4.8 Meaning of sequence calls in code generation and debugging process

4.5 Summary

Through converting various schemas to a graphs data structure, and further removing technical items and rebuild a intermediate tree structure with business data and its relations, we present a concrete, understandable business form to the business analyst. Sample data importing and re-layouting capability make the form metaphor more

meaningful to the business analyst. A visual spreadsheet-styled mapping specification environment utilizes the business analyst's previous knowledge and problem solving skill. In the mean while, with help by programming by demonstration technique, high-level mapping operations and a type system, the business analysts directly manipulate concrete form elements and data to define the mapping specification without being bothered by programming primitives. Most of simple and complex mapping specifications are designed be able to be directly and easily defined by drag-and-drop and type-and-select. But there is the cost for the high-level abstraction: the system will be very complex in order to deal with the mapping between the high-level objects and low-level objects; the user interface development will become very complicated; the designer have to consider very high-level mapping operation case, otherwise the high-level mapping operations will be not enough when mapping specifications get more complex. Architecture and OOD aims to produce a usable system with good flexibility, maintainability and extensibility. We have chosen a 2-tiered system for our prototyping because of its simplicity. Both the OOD and UI design help us to develop a correct system in the implementation stage.

Chapter 5 System Implementation

A proof-of-concept prototype is developed based on the requirements and design in previous chapters. In this chapter, the details of our prototype implementation are described.

5.1 Overview of Prototype

In our prototype implementation, Java is chosen as an implementation programming language because of its plenty of resources and platform independence. XML DTDs [W3C 2000 XML] are used as the source and target schemas due to a popularity of XML [W3C 2000 XML] in system developments and integrations nowadays. An XSLT [W3C 1999 XSLT] is used as our mapping specification implementation language corresponding to the XML-to-XML transformation. They are detailed in the followed sections.

The whole implementation structure of our prototype is shown on Figure 5.1. In this implementation, source and target XML DTD schema files and their XML instance files are parsed to objects with a DOM structure by the DTD parser and the XML parser respectively. Then the objects are fed to the form generator to generate Java Swing forms. The users interact with the mapping specification environment to define the mapping specification. After mapping specification for one field in target form is finished, the mapping specification is sent to the XSLT code generator to produce partial XSLT transformation code. The XSLT code and source XML file then are sent to the XSLT transformation engine to produce the partial target XML instance, then the target instance data is sent back to form for the users to check if the mapping specification is correct. After the entire mapping specification is defined, a XSLT code generator produces the final XSLT transformation code and a target XML instance is produced by XSLT transformation engine. Both of them are sent back to mapping specification environment for feedback to the users.

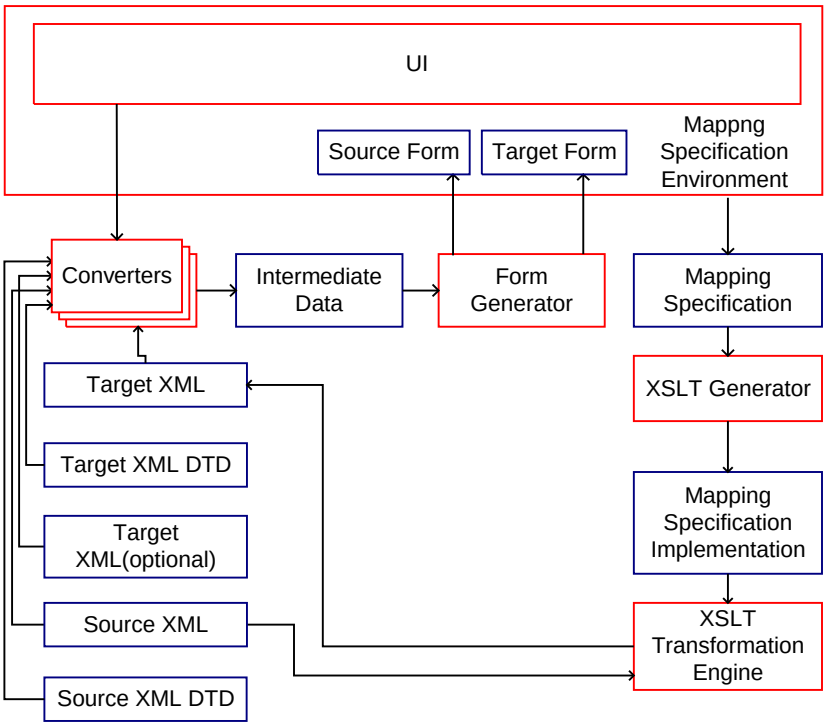


Figure 5.1 Implementation structure of prototype

5.2 Language Chosen

Theoretically many languages, such as C++, C# [Microsoft 2003 .NET], Delphi [Borland 2003 Delphi] and Python [Python 2003] etc, can be used as an implementation language for our design of our mapping tool. Java language is chosen based on following considerations:

- There are a lot of free Java resources available, including standard Java API-Java2SE [Sun 2003 J2SE], XML parsing API, XML transformation engine [Apache 2003], Java lexical analyzer, and Java parser etc. The free resources not only provide free use of the API but also the source code, which makes code extension or modification very easy.

- Platform independent. Considering the users of our prototype in the evaluation stage may have different operation systems, such as Windows, Unix, Linux, MacOS, we decided that Java is the best choice. By using Java, we can write it once and run it anywhere. But program developed by using Delphi can only be used in MS Windows. C++ code is platform-dependent. Theoretically C#

program can be platform-independent, but it can only be used in MS Windows .Net platform [Microsft 2003 .NET] in current time.

- The most proficient programming language for author.
- Borland JBuilder [Borland 2003 JBuilder], a Java IDE, which provides powerful GUI builder to make user interface development very efficient, is available in the university computer lab.
- A simple, elegant, pure object-oriented programming language which suits for the object-oriented design of our mapping tool.

The JBuilder with Sun Java 2 SDK1.4 [Sun 2003 J2SE], which integrates the Java™ API for XML Processing (JAXP) [Sun 2003 JAXP], is used as an IDE for prototype development.

5.3 XML/XML Parsing

5.3.1 XML and XML DTD

XML (Extensible Markup Language) is the meta-language defined by the World Wide Web Consortium (W3C) [W3C 2000 XML] that can be used to describe a broad range of hierarchical mark up languages. XML makes use of tags and attributes and defines only the structure of the document and does not define any of the presentation semantics of that document. It is a set of rules, guidelines, and conventions for describing structured data in a plain text, editable file. Using a text format instead of a binary format allows the programmer or even an end user to look at or utilize the data without relying on the program that produced it.

XML is increasingly used as a communication mediator in recent system integration, because it is simple and text-based, and provides a platform-independent and language-independent application integration methodology [Morgenthal 2001].

With increasing use of XML technologies, all the types of data mapping, such as objects, database, EDI etc, can be eventually transferred to a XML data-mapping domain. There are a lot of researches on how to transfer objects, database and their schemas to XML and its schema, and then transfer the XML to the objects or records of

database. A lot of related tools have been being developed [Skogan 1999] [Fong 2001]. From author's point of view, this makes XML data mapping will be a core part of other types of data mapping in the future. This is why we focus attention on XML data mapping in our prototyping.

In a schema of XML, we define the structure of an XML document, its elements, the data types of the elements and associated attributes, and most important, the parent/child relationships among the elements. The schema is used for not only validation XML, documentation and but also querying support, data binding and guide editing. There are two common schemas used for XML document are the XML DTD and XML Schema [W3C 2000 XML Schema].

The DTD is the validation scheme that is defined as part of the XML 1.0 specification. The DTD provides some basic capabilities for limiting the type and number of elements within a document. It also allows the document author to control the names and, to some extent, the contents of element attributes. But it does not allow authors any control over the character content of elements, making it a poor choice for sophisticated applications. A DTD is itself not an XML document. Due to above limitations of DTD, the XML Schema standard was developed. The XML Schema provides much finer control over the placement and contents of elements within a document and itself actually a XML document.

Based on that the DTD has a longer history and more stable standardization while the standard schema specification put forward by the W3C is still a candidate proposal, we chose the DTD as the first schema of XML in our prototyping. But we believe that by using XML Schema, our prototyping implementation and definition of mapping specification would be benefit from the XML Schema being actually a XML document and its more rich data types.

5.3.2 DTD Parsing/XML Parsing

The first thing of implementation of our prototype is XML DTD and XML parsing. According to the design, we need to parse both XML DTD and XML instance to an intermediate data structure. There are two types of XML parsing, one objects-based such as DOM (Document Object Model) [W3C 2002] and another event-based such as SAX (Simple API for XML) [saxproject 2002].

DOM is a set of interfaces defined by the W3C DOM Working Group [W3C 2002]. It describes facilities for a programmatic representation of a parsed XML document. DOM is a way of looking at a tree of XML data, and a group of APIs for reading and manipulating it. These APIs are common among DOM-compliant applications, so what works in one environment should work in another.

SAX is an XML processing method that was created by the members of the XML-DEV mailing list to solve problems that DOM just didn't solve. It provides an event-driven interface to the process of parsing an XML document. An event driven interface provides a mechanism for a “callback” notification to application’s code as the underlying parser recognizes XML syntactic constructions in the document. Rather than looking at an XML file as one giant lump of data that must be digested all at once, SAX looks at it as a stream of events, each of which carries information. Once this stream starts flowing, an application can examine it as it goes by and react accordingly, eliminating the need to store a huge amount of data that may never be needed.

In our tool, the schema data structure is frequently visited and its size won’t increase with increasing of data of its XML instance. So parsing the schema data to objects with the DOM structure is better than the SAX stream. Parsing an XML instance to the DOM structure will consume more memory when the XML instance gets larger. But in order to easily manipulate elements and attributes in the XML instance, in our implementation, we just ignore the memory consumption problem and parse the XML instance to objects with the DOM structure.

There are a lot of XML parsers implemented by Java. Sun JAXP [Sun 2003 JAXP] uses a factory class to enable applications to parse and transform XML documents independent of a particular XML processing implementation. Depending on the needs of the application, developers have the flexibility to swap between XML processors (such as high performance vs. memory conservative parsers) without making application code changes. So Sun JAXP with default Apache Xerces 1.44 XML parser [Apache 2000] was used as our XML parser.

Because XML DTD is not an XML file, so it should have its own parser. The parser of XML DTD in Apache Xerces 1.44 is only for internal XML validation and can’t be called by external program, a parser for XML DTD, named DTDParser which is mostly

modified from the internal parser of Xerces, is developed by author. The parser parses a XML DTD file to objects with DOM structure shown as Figure 5.2(1).

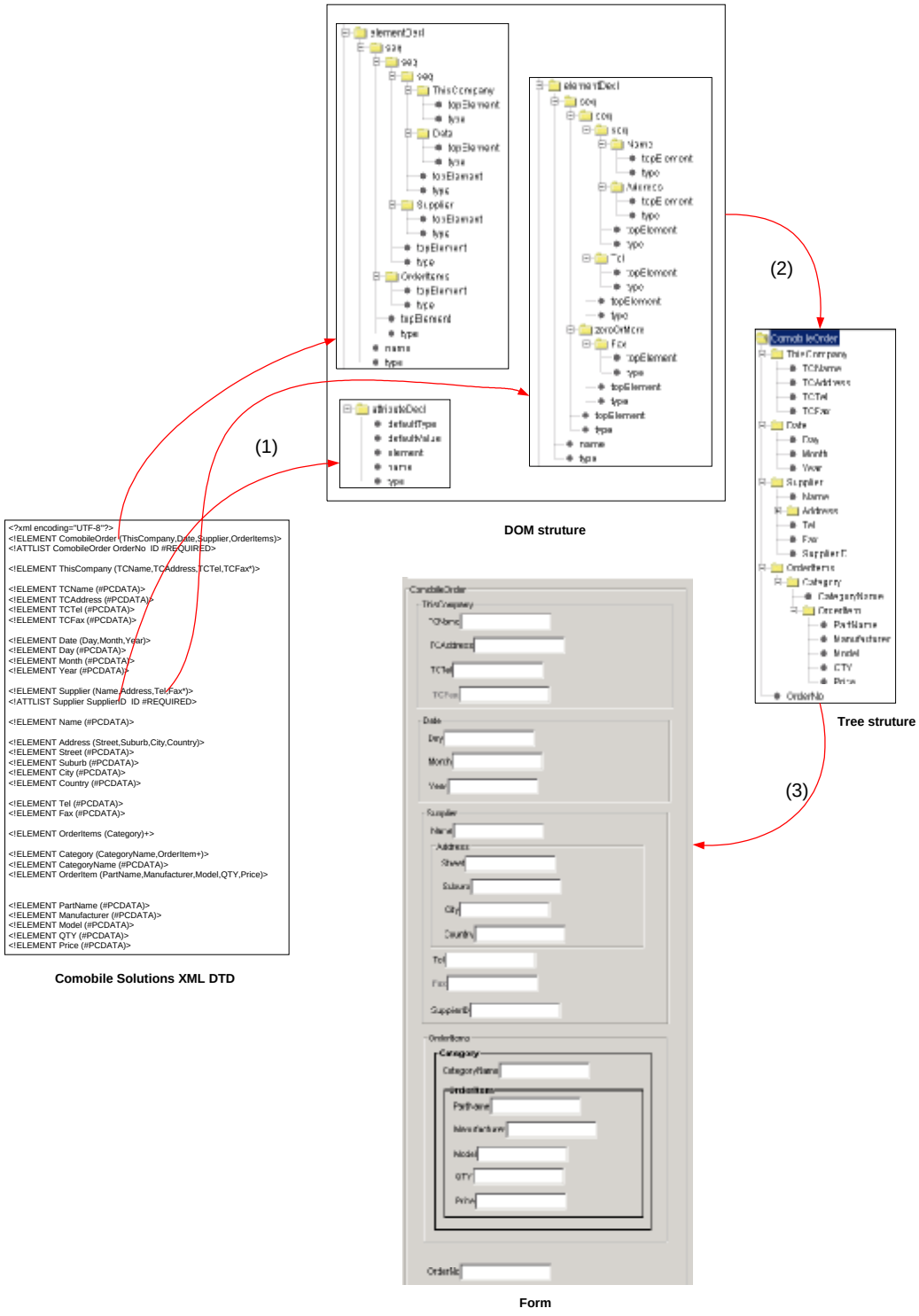


Figure 5.2 From XML DTD to form layout

5.4 Form Generation

Form Generator accepts the DTD objects with DOM structure and converts to a Java TreeModel which then is rendered to JTree for a tree view and a form view which is rendered by Java Swing components, such as JPanel, JLabel and JTextField etc (see Figure 5.2(2,3)). The structure of TreeNodes in the TreeModel is corresponding to DOM structure of the XML DTD except the nodes for attribute. An attribute in the DTD is a tree node with the same parent of the element the attribute belongs to in the TreeModel. In order to make good communication between form elements and TreeNodes, and TreeNodes and DTD DOM objects, two Hashtables are created for them. Thus operations on a form element are very easily mapped to the corresponding DTD DOM object.

When importing XML instance data to the form, because of the similarity of the structure of the treeModel and XML instance DOM model, the form generator traverses the tree in the treeModel and XML instance DOM model simultaneously, and fills the values that are got from the corresponding elements or attributes in the XML instance DOM model in the form. For collection nodes, only first child's value in the XML instance is imported to the corresponding form field(s).

5.5 UI Implementation and Mapping Specifications

Java provides the AWT [Sun 2003 JFC] and Swing [Sun 2003 JFC] API for windowing user interface development. They are all based on the Model-View-Controller (MVC) architecture. All Swing components are lightweight Components while the AWT components are heavyweight. For a component to qualify as lightweight, it cannot depend on any native system classes, also called "peer" classes. In Swing, the components do not depend on any peer classes for their view. The Swing library supports a cross-platform look-and-feel that remains the same across all platforms wherever the program runs. But the AWT library doesn't. The MVC-based architecture allows the lightweight Swing components to be replaced with different data models and views. The Swing library provides an API that gives more flexibility than the AWT API in controlling user interface widgets and determining the look-and-feel of applications, because of its high-level abstraction. Considering that our prototype may be running on different platforms, in order to keep the user interface consistent when it runs on

different platforms, Java Swing 2.0 is used as a main API for our user interface development.

For the form visualization, we use a JavaBean container as the base of form. A non-leaf form item is not only a JavaBean container and but also a JavaBean. A leaf form item is a JavaBean. The properties, such as font style, border types, etc. of the JavaBean can be easily modified according to the notation of form visualization for different nodes. And also the form items can be easily resized and moved.

A connection line between elements in the source and target forms is drew on the the same layer as the forms. A connection line between type sub-forms, and source and target forms are draw on the same layer as sub-forms. The line type is determined by number of source fields and target fields node, and the type of the root node of the abstract syntax tree (AST) produced from a mapping specification in the target field. After the mapping specification in target field is defined, the system will parse the mapping specification to generate XSLT code and then produce the target source and show the value on the target field. At this time, the line type is defined and system draws the line between the elements of source and target.

Swing's complexity on repainting different panel layer causes bugs on drawing connection lines between form items, and drawing a small button for collapsing and expanding type sub-forms.

5.6 XSLT Generation

Corresponding to XML, an XSLT (XSL Transformations) is used as our mapping specification implementation. An XSLT code generator is coded to generate an XSLT code. The XSLT code generator first traverses the target tree model to generate the XSLT document. In order to get the value of each generated target element, the code generator extracts the mapping specification from the node in the tree model, and parses the mapping specifications to generate the code. We expect that the generated XML-based XSLT code can be further input to a XSLT transformation engine as an XML source instance. Then with different XSLT code, the transformation engine can produce different mapping specification implementation. The code generation process is shown on Figure 5.3.

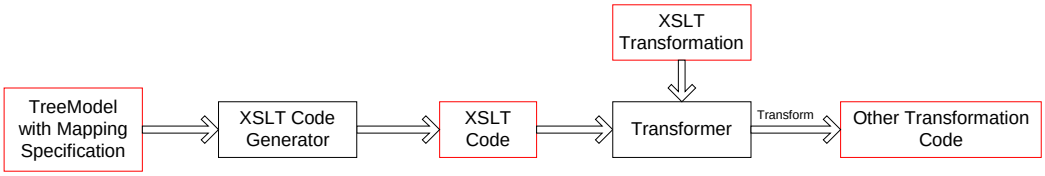


Figure 5.3 The code generation process

5.6.1 XSLT

XSLT describes a language for transforming XML documents into other XML documents or other text output. It was defined by the W3C. XSLT itself is XML documents.

5.6.2 JLex/CUP

The definition of mapping specification in each target fields is expressed in formulae or procedures in a text format that has certain lexicon and grammar defined by author. In the code generation process, the mapping specification is filtered by a lexical analyzer to tokens and then parsed to abstract syntax tree by a parser. And then a traversal of the abstract syntax tree is needed to generate XSLT code. The parser is built by using JLex [Berk 2003] and CUP [Hudson 1999], which are most familiar to the author (see Figure 5.4). JLex and CUP both are free available.

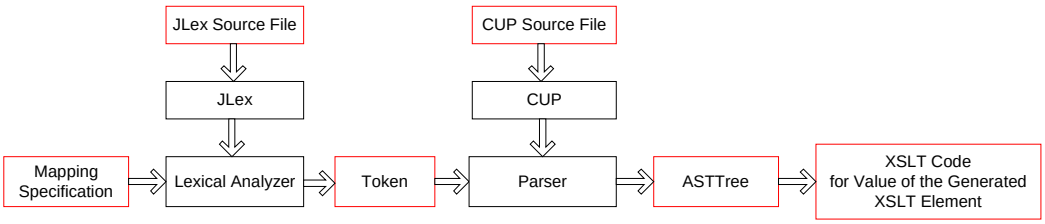


Figure 5.4 Compiling the mapping specification

JLex is a generator of lexical analyzer. JLex takes a JLex source file and compiles it into a Java implementation of lexical analyzer. The JLex source file for our mapping language is shown on Figure 5.5.

```
package relationParser;

import java.io.*;
import java_cup.runtime.*;

%%

%public
%type                   Symbol
%char

%{
        public Symbol token( int tokenType ) {
                System.out.println( "Obtain token " + sym.terminal_name( tokenType )
                                                + " \"" + yytext() + "\"" );
                return new Symbol( tokenType, yychar, yychar + yytext().length(), yytext() );
                }
%}

%init{
        yybegin( NORMAL );
%init}

%eofval{
        System.out.println( "Reach $END" );
        return new Symbol( sym.EOF, yychar, yychar + yytext().length(), "$END" );
%eofval}

intconst    =       ([0-9]+)
octDigit    =       ([0-7])
hexDigit    =       ([0-9a-fA-F])
escchar             =       (\\([ntbrfva\\\'\"\?]|{octDigit}+|[xX]{hexDigit}+))
schar               =       ([^\'\"\\\r\n]|{escchar})
charconst   =       (\'{schar}\')
stringconst =       (\"{schar}*\")
ident               =       ([A-Za-z_][A-Za-z0-9_]*(:[A-Za-z_][A-Za-z0-9_]*)?)
realnumber =        ([0-9]*\.[0-9]+)
space               =       ([\ \t])
newline             =       (\r|\n|\r\n)
%state NORMAL ERROR

%%

<NORMAL>{newline}   { }
<NORMAL>{space}             { }

<NORMAL>"("                 { return token( sym.LEFT ); }
<NORMAL>")"                 { return token( sym.RIGHT ); }

<NORMAL>or                  { return token( sym.OR ); }
<NORMAL>and                 { return token( sym.AND ); }
<NORMAL>not                 { return token( sym.NOT ); }
<NORMAL>"<"                 { return token( sym.LT ); }
<NORMAL>">"                 { return token( sym.GT ); }
<NORMAL>"<="                { return token( sym.LE ); }
<NORMAL>">="                { return token( sym.GE ); }
<NORMAL>"=="                { return token( sym.EQ ); }
<NORMAL>"<>"                { return token( sym.NE ); }
<NORMAL>"+"                 { return token( sym.PLUS ); }
<NORMAL>-                   { return token( sym.MINUS ); }
<NORMAL>"*"                 { return token( sym.TIMES ); }
<NORMAL>"/"                 { return token( sym.DIVIDE ); }
<NORMAL>%                   { return token( sym.MOD ); }

<NORMAL>return              { return token( sym.RETURN );}
<NORMAL>if                  { return token( sym.IF );}
<NORMAL>then                { return token( sym.THEN );}
<NORMAL>while               { return token( sym.WHILE );}
<NORMAL>else                { return token( sym.ELSE );}
<NORMAL>":="                { return token( sym.ASSIGN );}
<NORMAL>do                  { return token( sym.DO );}
<NORMAL>for                 { return token( sym.FOR );}
<NORMAL>upto                { return token( sym.UPTO );}
<NORMAL>downto              { return token( sym.DOWNTO );}
<NORMAL>";"                 { return token( sym.SEMICOLON );}
<NORMAL>":"                 { return token( sym.COLON );}
<NORMAL>,                   { return token( sym.COMMA );}
<NORMAL>"["                 { return token( sym.LEFTSQ );}
<NORMAL>"]"                 { return token( sym.RIGHTSQ );}
<NORMAL>"{"                 { return token(             sym.LEFTCURLY );}
<NORMAL>"}"                 { return token( sym.RIGHTCURLY );}
<NORMAL>"."                 {return token(sym.DOT);}


<NORMAL>true                { return token( sym.BOOLVALUE ); }
<NORMAL>false               { return token( sym.BOOLVALUE ); }
<NORMAL>{intconst}  { return token( sym.INTVALUE ); }
<NORMAL>{charconst} { return token( sym.CHARVALUE ); }
<NORMAL>{stringconst}       { return token( sym.STRINGVALUE ); }
<NORMAL>{ident}             { return token( sym.IDENT ); }
<NORMAL>{realnumber}        { return token( sym.REALVALUE );}


<NORMAL>.                   {
                                    yybegin( ERROR );
                                    return token( sym.ERROR );
                            }
<ERROR>";"                  {
                                    yybegin( NORMAL );
                                    //              return token( sym.SEMICOLON );
                            }
<ERROR>.            { }
<NORMAL>"//".*              { }
```

Figure 5.5 A partial JLex source file for our mapping language

CUP is a parser generator. It takes a CUP program and generates a Java program that will parse input that satisfies that grammar and produce an abstract syntax tree. The CUP program for our mapping language is shown on Figure 5.6.

```
start with Program;

Program::=
                    Expr:expr
                    {:
                    RESULT = expr;
                    :}
          |
                    StmtList:stmtList
                    {:
                    RESULT = stmtList;
                    :}

          |
                    COLON IDENT:ident LEFT FormalParamList:paramList RIGHT ASSIGN StmtList:stmtList
                    {:
                    RESULT = new FunctionDeclNode(ident, paramList, stmtList);
                    :}
          |
                    COLON IDENT:ident ASSIGN StmtList:stmtList
                    {:
                    RESULT = new FunctionDeclNode(ident, stmtList);
                    :}
          ;
FormalParamList::=
                    FormalParamList:paramList COMMA IDENT:ident
                    {:
                    paramList.addElement( ident );
                    RESULT = paramList;
                    :}

           |        IDENT:ident
                    {:
                    FormalParamListNode paramList = new FormalParamListNode();
                    paramList.addElement( ident );
                    RESULT = paramList;
                    :}
          ;
StmtList::=

                    StmtList:stmtList Stmt:stmt
                    {:
                    stmtList.addElement(stmt);
                    RESULT = stmtList;
                    :}
          |
                    Stmt:stmt
                    {:
                    StmtListNode stmtList = new StmtListNode();
                    stmtList.addElement( stmt );
                    RESULT = stmtList;
                    :}
          ;
Stmt::=
                    SEMICOLON
                    {:
                    RESULT = new NullStmtNode();
                    :}
          |
                    Path:path ASSIGN Expr:expr2 SEMICOLON
                    {:
                    RESULT = new AssignStmtNode(path, expr2);
                    :}
          |
                    RETURN Expr:expr SEMICOLON
                    {:
                    RESULT = new ReturnStmtNode(expr);
                    :}
          |
                    LEFTCURLY StmtList:stmtList RIGHTCURLY
                    {:
                    RESULT = new CompoundStmtNode(stmtList);
                    :}


          |
                    IF Expr:expr THEN Stmt:stmt1
                    {:
                    RESULT = new IfThenStmtNode(expr, stmt1);
                    :}
          |
                    IfThenElseStmt:stmt
                    {:
                    RESULT = stmt;
                    :}

          |
                    WHILE RelExpr:expr DO Stmt:stmt1
                    {:
                    RESULT = new WhileStmtNode(expr, stmt1);
                    :}
          |
                    DO Stmt:stmt1 WHILE RelExpr:expr SEMICOLON
                    {:
                    RESULT = new DoStmtNode(stmt1, expr);
                    :}
          |
                    FOR IDENT:ident ASSIGN Expr:expr1 UPTO Expr:expr2 DO Stmt:stmt
                    {:
                    RESULT = new ForUpStmtNode(ident, expr1, expr2, stmt);
                    :}
          |
                    FOR IDENT:ident ASSIGN Expr:expr1 DOWNTO Expr:expr2 DO Stmt:stmt
                    {:
                    RESULT = new ForDownStmtNode(ident, expr1, expr2,stmt);
                    :}
          |
                    error SEMICOLON
                    {:
                    RESULT = new ErrorStmtNode();
                    :}
          |
                    error RIGHTCURLY
                    {:
                    RESULT = new ErrorStmtNode();
                    :}
          ;
```

Figure 5.6 A partial CUP program for our mapping language

5.6.3 Debugging Mapping Specifications

We designed two stages to debugging mapping specifications, the first is every time when the mapping specification for one target field is finished and it gives users an immediate feedback whether the mapping specification is correct, The second is for all the defined mapping specifications after the users give an code generation command.

For the first one, we only take the mapping specification in the individual field, and compile it to generate a mini XSLT code. This avoids from traversing all the target data structure and transformation of the whole source XML document, and makes the debugging process very fast. In order to reuse the code generator that takes the whole target tree structure as a parameter, when producing the mini XSLT code, we build a temporary mini tree model which only contains the target field node, and take it to the code generator as the target tree structure. The coder generator will take the sample source data, the temporary mini tree model and formula defined in the target field to generate mini XSLT code. The sample of the generated mini XSLT code for splitting *Address* in *CS* form to *City* in *TP* order is shown on Figure 5.7. Then the mini XSLT code and source XML data are fed in a XSLT transformation engine to produce a mini XML target data. Finally the data shows on the target form.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="/">
    <City>
      <xsl:variable name="AfterSuburb">
        <xsl:variable name="afterStreet">
          <xsl:value-of select="substring-after(ComobileOrder/ThisCompany/TCAddress, ',')"/>
        </xsl:variable>
        <xsl:value-of select="substring-after($afterStreet, ',')"/>
      </xsl:variable>
      <xsl:value-of select="substring-before($AfterSuburb, ',')"/>
    </City>
  </xsl:template>
</xsl:stylesheet>
```

Figure 5.7 Mini XSLT code for splitting *Address* in *CS* form to *City* in *TP* order

For the second one, all the mapping specifications so far are compiled and XSLT code is produced by code generator, and then the whole XML document is transferred by the XSLT transformation engine and the results are shown on the target form to the user. This gives the user the whole picture of debugging the mapping specifications so far. The partial XSLT code for transfer *CS* order to *TP* order is shown on Figure 5.8.

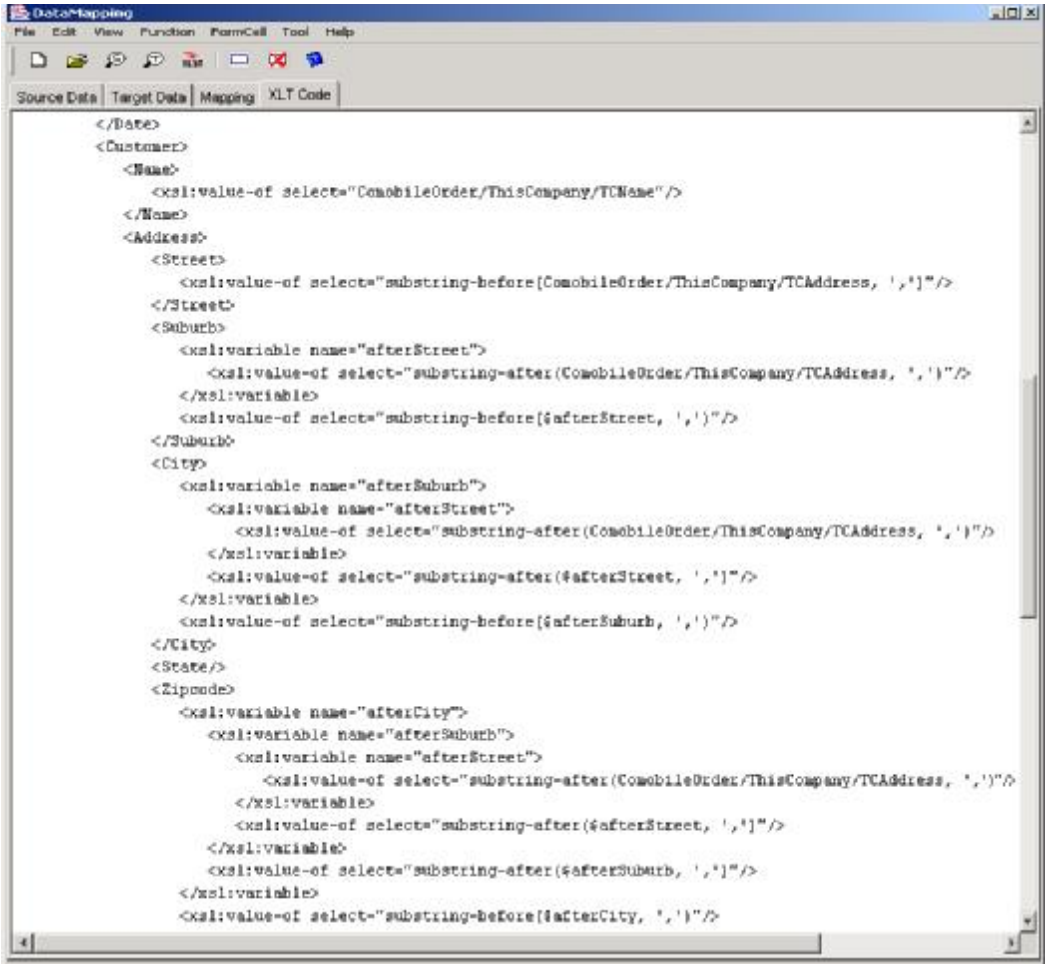


Figure 5.8 The partial XSLT code for transfer *CS* order to *TP* order

5.6.4 XSLT Transformation Engine Implementation

Again here by using JAXP API, we can isolate our application from the internal implementation details of a given Transformer. For the Transformer, there is an abstract Factory class with a static *newInstance()* method that instantiates a concrete Factory which wraps the underlying implementation. These *newInstance()* methods use system property settings to determine which implementation to instantiate.

Apache Xalan [Apache 2003] is used as a Transformation Engine in our implementation. Xalan provides high-performance XSLT stylesheet processing. Xalan fully implements the W3C XSLT and XPath [W3C 1999 XPath] recommendations.

5.7 Summary

By using Java as programming language, we implemented a flexible standalone data-mapping prototype according to our design. The prototype supports generating a XSLT mapping specification implementation for XML-to-XML transformations by using XML DTD as its source and target schema to define mapping specifications by an end user. Some simple mapping specifications and a few complex mapping specifications are supported by simply drag-and-drop and type-and select at this stage. A type system now is under way. The difficulties here are complex interactions between underlying data structures and user interface, mapping between different layers of data model made by the high level abstraction of operations and data model

Chapter 6 System Evaluation

In this chapter, the usability of system implemented in the previous chapter is evaluated. Because of limited available research time, a notational evaluation based on cognitive dimensions framework is discussed mainly.

6.1 Usability Evaluation

Usability refers to the characteristic of how easy it is to learn and to use a system. Usability can be defined as the extent to which the users can use a system to achieve their goals with effectiveness, efficiency and satisfaction in specified context and environment [Nielsen 1994].

To achieve better usability, there are two key principles of designing for usability. The one is early and continual evaluation, and the other one is iterative design and development [Nielsen 1994]. Prototyping provides a good model for evaluation and then the feedback from the evaluation can be used for further iterations of the design. The evaluation steps enable the designers to incorporate feedback from the users to their next iterative design until the system reaches an acceptable level of usability.

There are quite a few methods developed for the measurement of effectiveness and efficiency, such as inspection methods, e.g. Heuristic evaluation [Nielsen 1994], Cognitive Walkthroughs [Rowley 1992, Wharton 1994, Spencer, R. 2000], guideline checklists [Wixon 1994, Nielsen 1995], etc, which most time needs HCI specialists and can be used in the software design and prototyping stage, testing methods, e.g. Think Aloud protocol [Dumas 1993, Lindgaard 1994, Rubin 1994], co-discovery method, performance measurement, Question asking protocol [Dumas 1993, Lindgaard 1994, Rubin 1994] etc, which needs the real users, and observation methods, e.g. video records, eye-tracking, etc, such as those taken in a usability lab, which may be particularly useful in giving richer information about users' performance . In these methods, the designers need to design a set of standard tasks, and/or checklist, and/or

questionnaires for the users to work through with the software and make sure that these tasks represent what the users want the software to do. It gives the best impression of how a system would be used in the real world. The designers can assess the effectiveness and efficiency on the basis of end results and time taken to achieve then.

Inquiry methods, such as questionnaires [Kirakowski 2003] and surveys [Salant 1994], are usually used for measurement of satisfaction. Questionnaires are written lists of questions to distribute to the users. The users fill out the questionnaires and return them to the designers. Questionnaires and surveys are often performed before or after the users work through the prototype system, and are identified to be an efficient way to get users' expectations of the system and pick up some usability problems.

It is necessary to take above principles and some usability evaluation methods on our prototype to get the users feedback and find the problems of our system to make it achieve better usability. Available research time limits us for taking above methods on our prototype system at this stage. Instead, we conducted a notation analysis by using Cognitive Dimensions to evaluate the usability of our system in this thesis.

6.2 Cognitive Dimensions

Contrary to the above traditional usability evaluation approaches, cognitive dimensions provide vocabulary for designers or users to talk about the usability of the system in a broad-brush style rather than lengthy, detailed analysis. It provides a tool for non–HCI specialists to evaluate usability of information artifacts [Blackwell 2002].

The cognitive dimensions for visual programming and meaning of each dimension are already presented in the Chapter 2. In the following section we conduct a designer-led notational analysis [Blackwell 2002] by using cognitive dimensions for our prototype system. The steps of the analysis are described as following:

- Identify the main notation of the system, describing the media in which the marks of the notation are expressed and the environment in which it is manipulated.
- Identify sub-devices. A sub-device is a part of the main system that can be treated separately because it has different notation, environment, and medium.

- Consider each notation in terms of the list of dimensions, identifying any usability problems where the system characteristics on that dimension are inappropriate to the user activity.
- Identify problems and then consider how to improve them.

6.3 Evaluation

6.3.1 Notation of System

The main notation of system is our form-based metaphor, includes primitive form elements and their groups, different inter-elements links, and formulae in the form elements. All marks of notations are stored in computer memory and shown on the computer screen. The environment of the notation system is shown on Figure 4.10.

6.3.2 Sub-devices

Type abstraction management sub-device

The type abstraction management is used for the users to define types and their formats, delete types and their formats.

Operation abstraction management sub-device

Operation abstraction management is used for the users to define, edit and remove the operation.

Because the above two sub-devices are not forced for the user to use. We ignore the cognitive dimensions analysis for these two sub-devices in this thesis.

6.3.3 Cognitive Dimensions for Main Device

Abstraction gradient

For the form rendering in our system, all the data schemas are abstracted to concrete business forms. The data and relations in the schema are abstracted to primitive form

elements, such as labels, text fields, radio boxes, fonts, colors etc, and groups of primitives (see Figure 4.9).

For the mapping specification, links between fields represent formulae converting source data item(s) and group(s) to target data item(s) and groups(s) (Figure 4.19(8)).

All these build-in abstractions make our mapping specification environment more like the users' problem domain.

In order to make user easy to convert some data format, such as date, name, address etc, a type system is introduced. Some data types and their formats are pre-defined in the system. The users can also builder their own data types for reuse in the future. Also some common used operations, such as substring_before(), concat(), if_then_else, etc, for data transformation are built in the system. The user can add new function to the system as well. These abstractions lower the viscosity.

From above analysis, our system is an abstraction–tolerant system which permits but do not require user-defined abstractions. If the users just don't define the user-defined types or operations, there is low abstraction barrier for our system.

Closeness of mapping

Our form-based data transformation tool uses a concrete metaphor—a business form—to support data mapping specification (see Figure 4.10). Its visual representation thus maps directly onto business analyst's (the end users) cognitive model of their problem domain. The purpose of allowing generated form layout modification is to support even closer mapping allowing analysts to tailor the generated layout to be closer to the actual screen and hard-copy business form layouts they are familiar with.

Defining mapping specification can be mostly done by drag-n-drop and then following instructions of dialog box just like linking the relevant fields in actually hard copy business forms. Also the mapping specification environment closely maps to a spreadsheet model, which is very familiar to the business analysis.

Consistency

Both source and target form representations use the same visual form elements. All inter-form element links are rendered the same way. The little boxes along the link discriminate the differences of mapping. Also the form representations are consistent with the form-based mapping specification environment, because each field in the forms can be treated as the cell of the spreadsheet-styled programming environment (see Figure 4.10).

Diffuseness

Compared to other abstract approaches to representing data transformation, our form-based data mapping tool employs a more verbose visual language that can include elements not directly used in the mapping process e.g. business form layout groups, labels, lines and boxes. In contrast, mapping specifications using meta-data renderings such as trees and entity-relationship diagrams seldom include elements not directly used in the meta-data mapping specification.

The use of a concrete form-based metaphor in our approach necessitates a less terse notation to support the desired visual metaphor. But all of these can make our mapping specification environment closely mapping to the users cognitive model—business form copying—and make them define the mapping specification without the knowledge of underlying technologies.

Error-proneness

Typing errors and syntax errors are reduced dramatically by using clicking, selecting, drag-n-drop on the visual form metaphor and popup dialog boxes.

In order to make the end-users not to be bothered by underlying data model, we ignore some details of the data model when rendering the data model to the concrete data form, e.g. the attributes in the XML element are treated as the same as the child elements of the element. So when we add a new element in the source or target form, it could be reflected on the underlying data model incorrectly.

Hard mental operations

For end-users, hard mental operations are greatly reduced by having visualization close to their cognitive model of inter-business data exchange. The users needn't to know any data schema and programming language, memorize key words, functions and syntax etc. All the users needed, such as the form fields, operations, types, are presented on the mapping specification environment. The users just need to directly click on them or browse to find them when the users need them.

Hidden dependencies

Within forms, element groupings are all explicitly represented. All inter-form dependencies are explicitly represented as links between form elements and groups.

Premature commitment

- *Commitment for form layout*

The users need to decide the position and size of form element when re-layouting the form elements. For example, the rearrange of the *day*, *month* and *year* fields in *CS* form. See Figure 6.1.

- *Commitment for construction of formula*

The users need to decide where they should start when constructing mapping formulae in target field. Create an intermediate element first or directly define the formula in the target field? For example, the definition of *Suburb* field in *TotalPDAs* target form. Where do we start? See Figure 6.2.

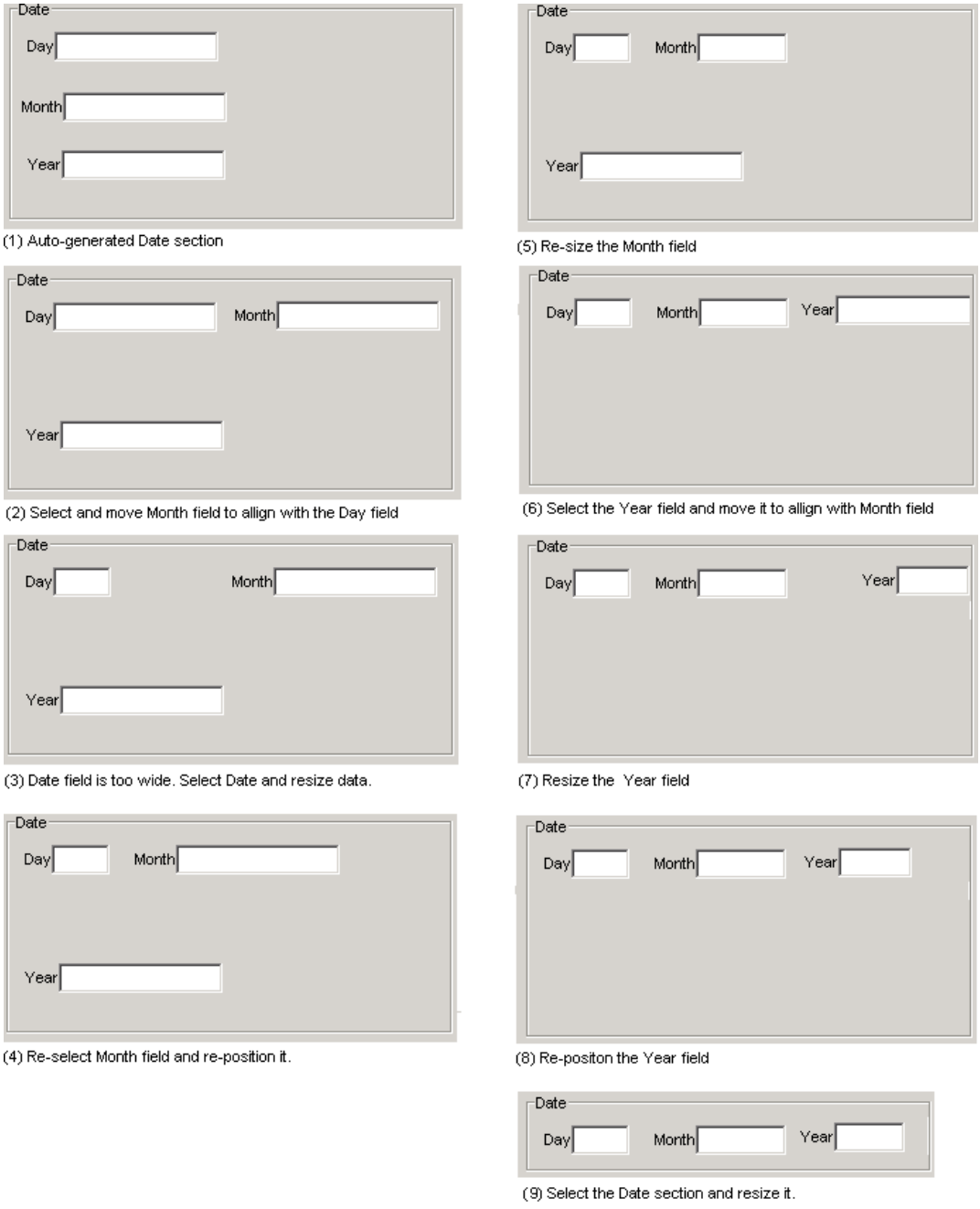


Figure 6.1 Rearrange elements in Date section in CS order form

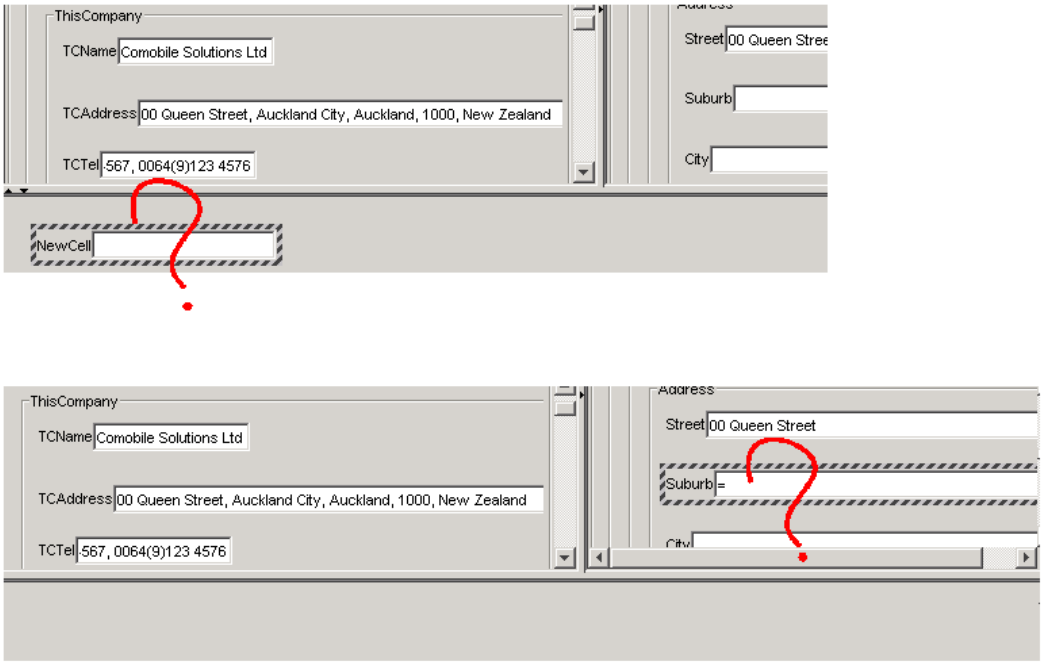


Figure 6.2 Where do we start when define the Suburb mapping specification?

Progressive evaluation

The formula definition can be immediately compiled to generate mapping specification implementation. The implementation and source instance are then fed to transformation engine to get the target instance. Then data in the results are shown on corresponding form elements in the target form. This makes the users be able to check if individual formula definition for each target field is correct or not.

Provisionality

When rearrange the layout of the form and constructing the formula, the users can try different possibilities. If there is something wrong, it's difficult for the users to go back to their original state and try again. They have to manually do the reverse operations they have done to restore the system to its original state. For example, after the users changed the layout of the day, month and year fields from 1 to 2 by resizing and moving each field. If they want to back to the state 1, they have to do all the reverse operations to go back.

Role-expressiveness

Concrete representations are used for all form elements that denote their role. Enclosure of elements by groups provides an additional role specification, that of the elements' relationship to others in the group.

Secondary notation

The business analyst can reorganize automatically generated form layouts, creating their own cognitively meaningful business form representation using form element layout, appearance and grouping. Our tool supports the use of this secondary notation relating to form element layout as it is cognitively important to the user and has meaning in terms of the grouping of form elements. The form layout and appearance has no effect on generated mapping code but regrouping or retyping form elements does.

No unstructured form annotations are currently supported though this may be a useful addition allowing end users to make notes against form s and form element links.

Viscosity

Modifications on our mapping tool include modifying form elements for change of data schema and modifying the formula defined in the target field.

Now the mapping tool cannot support the small changes of the data schema. Because the schema-to-form process is not bi-directional, it is impossible to directly add elements or regroup elements to reflect the change of data schema. When the data schema changes, a new project needs to be created to load the changed data source or target schema. Thus the whole mapping specifications need to be redefined from beginning and we can't reuse the mapping specification we have done before. For example,

Modifying formula takes many steps. If a formula for a target field needs to be changed, the user needs to trace paths of creation of formula and then directly modify the formula on the text-based expression except functions in the formula. For a function in the formula, the user needs to double-click on the function text to invoke the dialog box for the operation to edit it.

Visibility & Juxtaposability

The form-based mapper has explicit inter-form element links providing a good visibility, but the links between form elements and element groups to the underlying meta-model is hidden. When the user modifies form layout e.g. by adding or rearranging grouping, this linkage is blurred and is not visible in the visual form-based visualization or tree-based structure views.

A formula and the result of the formula for only one target field are visible simultaneously. The users have to click on target fields one by one to see their formula and can't view the formula definition of all the target fields simultaneously. It's impossible for the users to compare some formula definition in different form fields at one time.

Two views are supported in our tool: a concrete form-based visualization and tree-based structure visualization, which are viewed side-by-side. Sub-views are currently supported by using the tree-based view, or right clicking on the selected form elements and then selecting the "sub-view" menuitem, to select a portion of the form for display, but multiple views displayed simultaneously are not currently supported.

6.4 Some Improvements on Current Prototype

According to above evaluation, following improvements are supposed. Some of the improvements may conflicts to others.

Abstraction gradient

It's better to add more build-in types and operations in the system in development time to minimize possibility of end users directly defining types and operations.

Error-proneness

See *viscosity* for solving errors when adding elements into form to reflect the change of the schema.

Premature commitment

- *Commitment for form layout*

We may automate resizing a field according to another field position or the size of section in which the field is. See Figure 6.3.

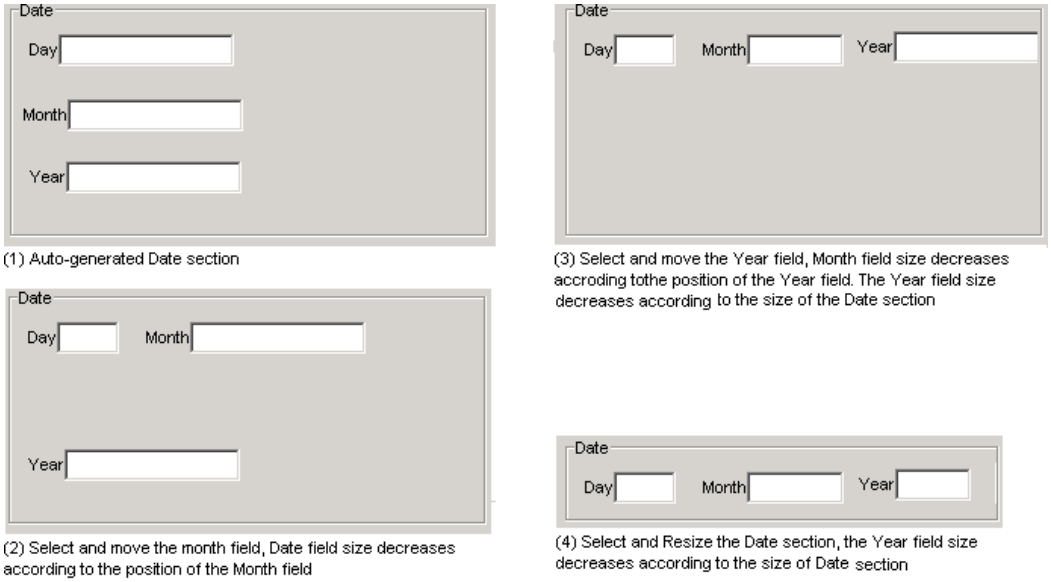


Figure 6.3 Revising of rearranging elements in the Date section in CS order form

- *Commitment for construction of formula*

See Provisionality bellow.

Provisionality

Keep the history of user's action so that the users can try the different layout of form and definition of the formula, and then easily come back to original state where they started the try by using "undo" command.

Secondary notation

Add unstructured annotations for the users to put comments on the mapping specification. This will increasing the viscosity, because when the mapping specification is changed, we may need to change the unstructured annotations.

Viscosity

When schema changes, let the users load changed schema to the system produce a form representation (Form1), and then compare it to the form of previous schema (Form 2), which mapping specifications are already defined, then the users can drag-n-drop the changed form elements from Form 1 to the Form 2. The program can get the underlying data model element corresponding to the form element in Form1, then create the same data element in the underlying data model corresponding to the Form 2. Through this we can reuse the previous data mapping specification to our changed data schemas.

Visibility & Juxtaposability

Make a formula definition box with each form element, not share a common box, to make the users view the formula for all form elements simultaneously.

Use multiple windows to display the mapping specification environment in parallel to make the users be able to refer to the mapping specification they have already defined while defining new mapping specification.

6.5 Summary

From the above evaluation results, we conclude that our mapping tool have a good support for the end user, because of its closely mapping to the user's problem domain, low hard mental operations, high level abstraction on data schemas, operations and types, and low abstraction barrier. And also it has a consistent user interface from form presentations of data schemas to spreadsheet-styled mapping specification environment, low hidden dependence and good progressive evaluation. But some difficulties of the tool on modification and exploratory activities need to be improved.

Chapter 7 Conclusions and Future work

7.1 Conclusions

This research has identified some of the main problems with current data transformation systems and their development process. These include the involvement of programmers increasing errors of mapping specifications, and the associated increase in the cost and time of development of these systems. Current mapping tools mainly focus on supporting professional programmers or data modelers. We have argued that a business analyst is the best person to define mapping specifications. This is because they know the business processes and the context of use of business data. In order to eliminate the involvement of programmers, a new mapping tool should provide better end-user support for such business analysts.

This research has analyzed the system requirements for a data mapping tool for end-users—the business analysts. Due to the business analyst having no knowledge of complex data schemas or a programming background, our mapping tool needs to provide a visual presentation to hide the complex underlying data structures. It also must provide a visual mapping specification environment to give the user a direct manipulation interface to define mapping specifications. It must generate the mapping specification implementation and give the users immediate feedback for debugging purposes.

This research has led to the design and implementation of a java prototype of the form-based mapping tool based on the 2-tiered architecture due to its simplicity. It uses a business form metaphor to represent the complex underlying data structures to a concrete, meaningful business form. The user can import the source and target instance to the form and make the form more concrete and more understandable. The user can also further customize the automatically generated forms to fit their cognitive model of forms. A business form copying metaphor, i.e. a form-based/spreadsheet-styled mapping specification environment, is provided to utilize the business analyst's

previous domain knowledge and problem-solving skill. It supports the end-users to define the most of one-to-one, one-to-many, many-to-one and many-to-many conditional or unconditional mapping specifications on field-, section-, collection-level by direct manipulating source and target form elements and concrete data sample through drag-and-drop, type-and-select. A type system is introduced to further simplify defining the mapping specification. After the mapping specification for one element in the target form is finished, the system can automatically produce the result in the target form for debugging purpose, and generate the mapping specification implementations on demand. All of these efforts make our mapping tool get ready for a business analyst to define mapping specifications and produce mapping specification implementations.

The java implementation of prototype supports a XSLT mapping specification for data from XML to XML transformations, which are very popular in today's system development and integration practices. This is due to its simplicity and platform independence. The source and target XML DTDs are parsed into DOM tree structures and these are used to automatically generate the form rendered by Java Swing. Form elements can be rearranged through resizing and moving the elements. Data in the source and target XML instances can be imported to the form. A built-in type system allows the users to apply a type to a form field or section to simplify mapping operations. When generating XSLT code, our tool traverses the target data structure, and then parses the mapping specification with the nodes to a abstract syntax tree by a parser generated by JLex and CUP and finally generates the mapping specification implementation by traversing the abstract syntax tree. A built-in XSLT transformer can transform a source XML instance to a target XML instance according to mapping specification defined and give immediate feedback to the user for debugging after each field-, section-, collection-level mapping specification is finished.

This research also conducted an initial evaluation on the prototype in a broad-brush manner through a notational and visual tool analysis by using the cognitive dimensions framework. Through this analysis, positive usability results are obtained for most of the key dimensions, but some usability problems of the prototype are identified. These include: not supporting changes to data schemas, no unstructured notation support for the user to make comments on the mapping specification, no history records saved for the user to easily explore possibilities and go back to a previous specification, the premature commitment when the user rearrange the form elements, and no multi-views

support for displaying the mapping specification definition simultaneously for comparison and reference. Possible improvements for these are suggested. The cognitive dimensions analysis is helpful to evaluate some usability aspects of system to identify problems where the system characteristics on that dimension are inappropriate to the user activity and consider design maneuvers to adjust that dimension.

Through all the above work, we can conclude that this research results in a new way—business form-based data mapping specification by the end-users—in data transformation area. The initial prototype has provided the user a concrete business form metaphor with high level abstraction on the complex underlying data structure and a form-based/spreadsheet styled mapping specification environment to mimic the end-users' mental model to assist them to define simple mapping specifications and some of complex mapping specifications.

7.2 Summary of Main Contributions

This work has produced the following contributions to the field of data mapping systems:

- Identification of a set of requirements for an end-user (business analyst) supporting data transformation specification environment. These include the use of a business analyst-focused metaphor for representing complex data and data correspondences, and generation of mapping implementations from these high-level descriptions.

- Design of a form-based representational metaphor for complex business data and a form field copying-based metaphor for specifying data transformations between these business form elements. This approach provides a more concrete and business-centric view of data and data transformations for business analysts to work with.

- Implementation of a proof-of-concept prototype of this form-based data mapping environment. This has included the import and visualization of complex business data using a business forms representation; the specification of data mappings between business form elements using drag-and-drop

programming-by-example; and generation of XSLT-based data transformation implementations from these specifications.

- Analysis of the potential usability advantages and disadvantages of our prototype data mapping environment using the cognitive dimensions framework. This has identified a number of strengths and weaknesses with the prototype tool.

7.3 Future Work

A number of areas exist in which we can improve the design and prototype implementation of the form-based mapping tool:

Further improve the usability of prototype for the future evaluation according to the findings made by cognitive dimensions analysis. The improvements include:

- Adding support for reuse of some of mapping specifications when a data schema changes. Through importing the changed data schema and visualizing it to a form, the user can compare it with the original form and find the differences. Then the user can take proper actions, such as drag-and-drop, deleting, renaming, to invoke the system to change the underlying original data schema to be the same as the changed data schema.
- Adding unstructured notation support for the user to make comments on mapping specifications. Add a text box associated with the desired form field on the top layer of the panel for the user to input notes for the mapping specification in the form field.
- Saving history records for the user to easily explore possibilities and go back to a previous state. Add each incremental mapping specification to a collection to keep the history of operations, then the user can go back or forward to switch among them by just clicking on a back or a forward button.
- Providing automatically resizing a field and section to minimize the premature commitment when the user rearranges form elements. When a field or section is resized or moved, the system can check the position of elements in its parent

section, and adjust the size of the elements to fit the position of the moved element.

- Providing multi-views to display the mapping specification definition simultaneously for comparison and reference. Use multiple external or internal frames to make the user be able to open new windows to display mapping specifications in parallel, or add a formula definition box on each form element to shown its mapping specifications other than all the form elements sharing one text area to show the definition of mapping specifications.

We should conduct further usability evaluations on the current prototype to get more feedback from HCI experts and actual users. We will then redesign the interface of our mapping tool based on this feedback to achieve iterative improvement on the current user interface design of the mapping tool. Because current our prototype is in a primary stage, we are planning to first use continual evaluation by using inspection method performed by computer science graduates who have some experiences on HCI to identify some usability problems with our user interface design to iteratively improve the design and our prototype. Then we will use survey/questionnaire method to let the real users—the business analysts—use our tool to perform the real world data mapping specifications and get the feedback from them. Our mapping tool is aim to enable the end user who has no programming knowledge to define mapping specification, their experience on the tool and whether they are satisfied with the tool are the most valuable feedback for evaluating and improving the tool.

Investigate the possibility of directly importing the scanned business forms or some electronic forms, such as HTML forms, Microsoft Access forms, etc. to provide the end-users an exact business form metaphor. This will then not to require the user to rearrange the form layout, and avoid the premature commitment on customizing form layout. The idea we have is that we can first directly import the scanned forms or electronic forms or reconstruct the forms from the scanned forms [Casey 1992] [Lam 1993] [Mao 1996] [Tang 1993] [Atalay 1999] and then automatically mapping the elements in the underlying data schemas and data instances to the pre-printed fields in the imported forms or reconstructed forms.

Investigate automatic mapping for simple mapping specifications by using imported sample data. Through traversing the source and target data structure, the system

compare the sample data value in the source and target instances, if there is a sample data from source, which equals to, or contains, or is part of a sample data in the target, the system can initially automatically define the mapping as one-to-one copy, or splitting, or combining. Then the user can further investigate correctness of the mapping specifications and modify them.

Extend the code generation to generate more mapping specification implementations based on the generated XML-styled XSLT code by using XML transformation approach, i.e. further transforming the XSLT code through a XSLT transformation engine to produce the Java, or other mapping specification implementation according to an XSLT-to-Java or XSLT-to-other mapping specification.

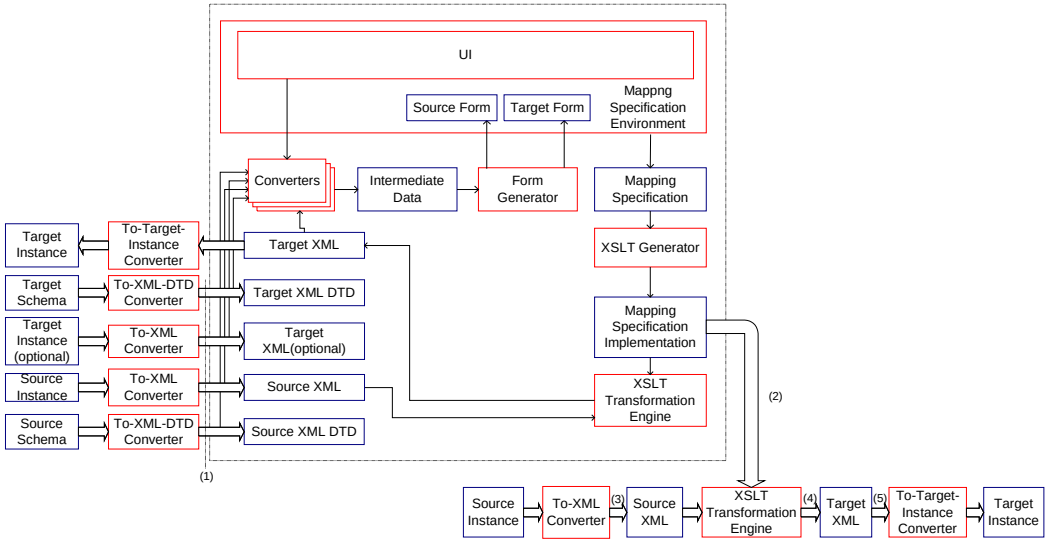


Figure 7.1 Non-XML source and target data transformation by an XSLT transformation engine

Support non-XML transformation by using an XSLT transformation engine. The author believes that the current prototype can form a core part of mapping tool to generate XSLT mapping specification implementation from non-XML data schemas and their instances. The architecture of the system is shown on Figure 7.1. In order to achieve that, at first, the non-XML data schemas and their instance need to be converted to XML DTD presentations and XML instances [Skogan 1999][Fong 2001] (see Figure 7.1(1)). Then we take the converted XML DTD and XML instances as inputs of our prototype and define the mapping specification to generate the XSLT transformation code. And then the XSLT transformation code is fed to XSLT transformation engine

(see Figure 7.1(2)) to translate the source XML instance converted from the non-XML source instance (see Figure 7.1(3)) to target XML instance (see Figure 7.1(4)). Finally the target XML instance is converted to the required non-XML target instance (see Figure 7.1 (5)).

References

Amor, R., Hosking, J.G. and Mugridge, W.B. (1999). "ICAtect-II: A Framework for the Integration of Building Design Tools." *Automation in Construction* 8 (3) (1999) pp. 277-289.

Apache Software Foundation (2003) "Xalan-Java version 2.5.1." http://xml.apache.org/xalan-j/index.html. Last access on June 8, 2003.

Apache Software Foundation (2000) "Xerces-Java Parser Readme." http://xml.apache.org/xerces-j/. Last access on June 8, 2003.

Atalay, Volkan, Erhan Arslan (1999). "An SGML Based Viewer for Form Documents." *ICDAR 1999*: 201-204.

Berk, E. J. and C. Scott Ananian (2003), http://www.cs.princeton.edu/~appel/modern/java/JLex/. Last access on June 8, 2003.

Blackwell, Alan, Thomas Green (2002). "Cognitive Dimensions of Notations: A tutorial." Presented at IEEE Symposia on Human-Centric Computer (HCC02), Washington DC, September 2002.

Borland Software Corp (2003). "Delphi", http://www.borland.com/delphi/. Last access on June 8, 2003.

Borland Software Corp (2003). "Jbuilder." http://www.borland.com/jbuilder/. Last access on June 8, 2003.

Brown, P. and J. Gould (1987). "Experimental study of people creating spreadsheets." *ACM Trans. Office Info. Sys.*, 5(3):258-272, July 1987.

Burnett, Margaret (1999). "Visual Programming." In *Encyclopedia of Electrical and Electronics Engineering* (John G. Webster, ed.), John WIley & Sons Inc., New York, 1999.

capeclear.com (2001). "CapeStudio technical overview." http://www.capeclear.com/products/whitepapers/CapeStudio_Product_Overview.pdf. Last access on June 8, 2003.

Casey, R., Ferguson, K. Mohiuddin, and E. Walach (1992). "Intelligent forms processing system." *Machine Vision and Applications*, 5:143-155, 1992.

Ceri, S., S. Comai, E. Damiani, P. Fratemali, S. Paraboschi, and L. Tanca (1999). "XML-GL: a graphical language for querying and restructuring XML documents." *8th WWW conference*, 1999.

Cypher, Allen, Daniel C. Halbert, David Kurlander, Henry Lieberman, David Maulsby, Brad A. Myers, and Alan Turransky (1993). *Watch What I Do: Programming by Demonstration*. The MIT Press.

Data Junction Corporation (2003). "Integration Studio." www.datajunction.com . Last access on June 8, 2003.

Dumas, JS, and Redish, Janice (1993) "A Practical Guide to Usability Testing." Ablex, Norwood, NJ, ISBN 0-89391-991-8.

Emmanuel Pietriga, Jean-Yves Vion-Dury, Vincent Quint (2001). "VXT: a visual approach to XML transformations." In *Proceedings of the 2001 ACM Symposium on Document engineering*, ACM Press New York, NY, USA, Pages 1 to 10.

Erwig, Martin (2002), "Xing: A Visual XML Query Language." *Journal of Visual Languages and Computing*, Vol. 13.

Floyd, (1984) "A Systematic Look at Prototyping." In Budde, R. et al (Eds.), *Approaches to Prototyping*, Springer-Verlag.

Fong, J., Pang, F. and Bloor, C. (2001). "Converting Relational Database into XML Document." 1st International Workshop on Electronic Business Hubs, Germany, September 2001.

Gilmore D.(1995). “Interface design: Have we got it wrong?” In K. Nordby, D. Gilmore, and S. Arnesen, editors, INTERACT’95. Chapman and Hall, London.

Goodell Howie (1998). “Methods of End-User Programming.” http://www.cs.uml.edu/~hgoodell/EndUser/methods.htm. Last access on June 8, 2003.

Gray, W. and J. R. Anderson (1987). “Change-Episodes in Coding: When and How Do Programmers Change Their Code.” *Empirical Studies of Programmers: Second Workshop*. G. M. Olson, S. Sheppard and E. Soloway. Norwood, NJ, Ablex: 185-197.

Green, T.R.G. and Petre, M. (1996). “Usability Analysis of Visual Programming Environments:A 'Cognitive Dimensions' Framework.” *Journal of Visual Languages and Computing* 7(2): 131-174.

Grundy, J.C., Mugridge, W.B., Hosking, J.G. and Kendall, P. (2001). “Generating EDI Message Translations from Visual Specifications.” In *Proceddings of the 2001 IEEE Automated Software Engineering Conference*, San Diego, CA, 26-28 Nov 2001, IEEE CS Press.

Hays, J.G. and Burnett, M.M. (1995). “A Guided Tour of Forms/3.” Oregon State University: Dept. of Computer Science Technical Report 95-60-6.

Hoc, J.-M. and A. Nguyen-Xuan (1990). “Language Semantics, Mental Models and Analogy.” *Psychology of Programming*. J.-M. Hoc, T. R. G. Green, R. Samurçay and D. J. Gilmore. London, Academic Press: 139-156.

Hudson, Scott, Frank Flannery, C. Scott Ananian (1999). “CUP Parser Generator for Java.” http://www.cs.princeton.edu/~appel/modern/java/CUP/. Last access on June 8, 2003.

Kramer, J. and Magee, J. (1997). “Distributed Software Architectures.” http://www.ixs.uci.edu/pub/icse97/program/tutorials. Last access on June 8, 2003.

Johnson, B., and Shneiderman, B. (1991). “Treemaps: a space-filling approach to the visualization of hierarchical information structures.” *Proceedings of the 2nd Intemutional IEEE Visualization Conference*, San Diego, pages 284-291, 1991.

Kirakowski, Jurek (2003). "Questionnaires in Usability Engineering: A List of Frequently Asked Questions (3rd Ed.)." http://www.ucc.ie/hfrg/resources/qfaq1.html. Last access on June 8, 2003.

Lam, S., L. Javanbakht, and S. Srihari (1993). "Anatomy of a form reader." *Proc. 2nd Intl. Conf. On Document Analysis and Recognition*, 2:506-509, 1993.

Lewis, C. and G. M. Olson (1987). "Can Principles of Cognition Lower the Barriers to Programming?" *Empirical Studies of Programmers: Second Workshop.* G. M. Olson, S. Sheppard and E. Soloway. Norwood, NJ, Ablex: 248-263.

Li, Yongqiang, John C. Grundy, Robert Amor, John G. Hosking (2002). "A Data Mapping Specification Environment Using a Concrete Business Form-Based Metaphor." *IEEE Symposia on Human Centric Computing Languages and Environments* 2002: 158-

Lindgaard, G. (1994). *Usability Testing and System Evaluation: A Guide for Designing Useful Computer Systems*. Chapman and Hall, London, U.K. ISBN 0-412-46100-5.

Mao, J., M. Abayan, and K. Mohiuddin (1996). "A model based form processing subsystem." *Proc. 13th Intl. Conf. On Pattern Recognition*, 2:691-695, 1996.

Microsoft Corporation (2003). "Microsoft Excel." http://www.microsoft.com/office/excel/evaluation/guide.asp. Last access on June 8, 2003.

Microsoft Corporation (2003). "Microsoft Visual C#." http://msdn.microsoft.com/vcsharp/. Last access on June 8, 2003.

Microsoft Corporation (2003). "Microsoft .NET." http://www.microsoft.com/net/ . Last access on June 8, 2003.

Milo, T., and S. Zohar (1998). "Using Schema Matching to Simplify Heterogeneous Data Translation." In *Int. Conference on Very Large Data Bases*(VLDV), New York.

Morgenthal, J.P. (2001). "XML: The New Integration Frontier." *EAI Journal*, Feb. 2001, www.eaijournal.com.

Myers, Brad (1998). "Natural Programming: Project Overview and Proposal." Carnegie Mellon University School of Computer Science Technical Report, no. CMU-CS-98-101 and Human Computer Interaction Institute Technical Report CMU-HCII-98-100. January, 1998.

Nardi, B. A. (1993). *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA, The MIT Press.

Nielsen, J. (1992). "The usability engineering life cycle." *IEEE Computer* 25, 3 (March), 12-22.

Nielsen, J. (1994). "Heuristic Evaluation." *Usability Inspection Methods,* J. Nielsen and R.L. Mack. New York, John Wiley & Sons: 25-62.

Nielsen, Jakob (1995). "Usability Inspection Tutorial." 1995, *CHI '95 Proceedings*

Pane, J. F. and B. A. Myers (1996). "Usability Issues in the Design of Novice Programming Systems." Pittsburgh, PA, Carnegie Mellon University. CMU-CS-96-132.

Pane, J. F., B.A. Myers, and L.B. Miller (2002). "Using HCI Techniques to Design a More Usable Programming System." In *proceedings of IEEE 2002 Symposia on Human Centric Computing Languages and Environments* (HCC 2002), Arlington, VA, September 3-6, 2002, pp 198-206.

Pane, J. F. (2002a). *A Programming System for Children that is Designed for Usability*, CMU-CS-02-127, May 3, 2002

Panko, R. (1998). "What we know about spreadsheet errors." *J. End User Comp*., pages 15021, Spring 1998.

Pausch, R., M. Conway, et al. (1992). "Lesson Learned from SUIT, the Simple User Interface Toolkit." *ACM Transactions on Information Systems,* 10(4): 320-344.

Rothermel, K. J., L. Li, C. DuPuis, and M. Burnett (1998)."What you see is what you test: A methodology for testing form-based visual programs." In The *20th Intl. Conf. Softw. Eng.*, pages 198-207, Apr. 1998.

Rothermel, K. J., C. R. Cook, M. M. Burnett, J. Schonfeld, T. R. G. Green, and G. Rothermel (2000). "WYSIWYT Testing in the Spreadsheet Paradigm: An

Empirical Evaluation." *Proceedings of the 22nd International Conference on Software Engineering,* June, 2000, pages 230-239.

Rowley, David E., and Rhoades, David G (1992). "The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure." *CHI `92 Proceedings,* (May 3-7, 1992): 389-395.

Rubin, Jeffrey (1994). *Handbook of Usability Testing,* John Wiley and Sons, New York, NY ISBN 0-471-59403-2.

Python.org (2003). "Python." http://www.python.org/. Last access on June 8, 2003.

Salant, Priscilla, and Dillman, Don A.(1994). *How to Conduct Your Own Survey*, John Wiley & Sons, New York, NY, ISBN: 0471012734.

saxproject.org (2002). "Simple API for XML(SAX)." http://www.saxproject.org/. Last access on June 8, 2003.

Skogan, David (1999). "UML as a Schema Language for XML based Data Interchange." In *Proceedings of the 2nd International Conference on The Unified Modeling Language* (UML'99), 1999. http://citeseer.nj.nec.com/david99uml.html.

Smith, D. C., A. Cypher, et al. (1994). "KidSim: Programming Agents Without a Programming Language." *Communications of the ACM.* 37(7): 54-67.

Sonic Software Corporation (2003). http://www.sonicsoftware.com. Last access on June 8, 2003.

Spencer, Rick (2000). "The streamlined cognitive walkthrough method." *CHI 2000 Proceedings*, (April 1 - 6, 2000): Pages 353-359.

Sun Microsystem Corp (2003). "Java API for XML Processing (JAXP)." http://java.sun.com/xml/jaxp/index.html. Last access on June 8, 2003.

Sun Microsystems Corp (2003). "Java 2 Platform, Standard Edition (J2SE)." http://java.sun.com/j2se/1.4.1/. Last access on June 8, 2003.

Sun Microsystems Corp (2003). "Java Foundation Classes: Cross-Platform GUIs & Graphics." http://java.sun.com/products/jfc/. Last access on June 8, 2003.

Svendsen G.(1991). "The influence of interface style on problem-solving." Intl. J. *Man-Machine Studies*, 35:379-397.

Swatman, P.M.C., Swatman, P.A., Fowler, D.C. (1994). "A model of EDI integration and strategic business reengineering." *Journal of Strategic Information Systems*, vol.3, no.1, March, 1994, pp.41-60.

Tang, Y., C. Yan, M. Cheriet, and C Suen (1993). *Automatic analysis and understanding of form documents*, Pages 625-654, 1993.

W3C (1999). "XSL Transformations (XSLT) Version 1.0." http://www.w3.org/TR/xslt. Last access on June 8, 2003.

W3C (1999). "XML Path Language (XPath) Version 1.0." http://www.w3.org/TR/xpath.html. Last access on June 8, 2003.

W3C (2000). "Extensible Markup Language (XML) 1.0 (Second Edition)." http://www.w3.org/TR/2000/REC-xml-20001006#NT-document. The last access on June 8, 2003.

W3C (2000). "XML Schema 1.1." http://www.w3.org/XML/Schema. Last access on June 8, 2003.

W3C (2002). "Document Object Model (DOM)." http://www.w3.org/DOM/. Last access on June 8, 2003.

Wharton, Cathleen, et. al. (1994). "The Cognitive Walkthrough Method: A Practictioner's Guide." In Nielsen, Jakob, and Mack, R. eds, *Usability Inspection Methods*, 1994, John Wiley and Sons, New York, NY. ISBN 0-471-01877-5 (hardcover).

Wixon, Dennis, et. al. (1994). "Inspections and Design Reviews: Framework, History, and Reflection." In Nielsen, Jakob, and Mack, R. eds, *Usability Inspection Methods*, 1994, John Wiley and Sons, New York, NY. ISBN 0-471-01877-5.

Wilcox, E., J. Atwood, M. Burnett, J. Cadiz and C. Cool (1997). "Does continuous visual feedback aid debugging in direct-manipulation programming systems?" In *ACM CHI'97*, pages 22-27, Mar. 1997.

Wilcox, E. and Burnett, M. "Programming a Single Digit LED in Forms/3." http://ww.cs.orst.edu/~burnett/Forms3/LED.html. Last access on June 8, 2003.