University of Alberta

TaMeX: A Task-structure Based Mediation Architecture for Integration of Web Applications Using XML

by

Qihua Gina Situ ©

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Science.

Department of Computing Science

Edmonton, Alberta
Spring 2001

0-612-60498-5

University of Alberta

Library Release Form

Name of Author: Qihua Gina Situ

Title of Thesis: TaMeX: A Task-structure Based Mediation Architecture for Integration of Web Applications Using XML

Degree: Master of Science

Year this Degree Granted: 2001

Qihua Gina Situ
Department of Computing Science
Computing Science Centre
University of Alberta Edmonton, Alberta
mod Canada, T6G 2E8

Date: Feb. 9th, 2001

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled **TaMeX: A Task-structure Based Mediation Architecture for Integration of Web Applications Using XML** submitted by Qihua Gina Situ in partial fulfillment of the requirements for the degree of **Master of Science.**

Dr. Eleni Stroulia
Supervisor

Renée E. Elio

Dr. Renée Elio

Dr. Leiser Silva

Date: Feb 13th. 2001

To my loving parents and sister

Abstract

Nowadays, there exists an enormous number of Web-based applications that offer information and services in support of a variety of activities. These services can be competitive or complementary. Nevertheless, as they do not interoperate, it is up to the user to access them individually, interpret their responses to his requests, and compare or combine these services. In this thesis, in the content of the TaMeX project, we develop a task-structure based mediation architecture for integration of Web applications offering information and services in a specific domain. At the center of an integrated TaMeX application is a mediator, whose behaviors are driven by its task structure. Its responsibilities are to interact with the user and to effectively integrate the available resource applications in order to accomplish the user's task. The integration of the resource applications is based on the availability of a common domain model and a set of wrappers driving and extracting information from their corresponding Web resources. The construction of a wrapper is based on a hierarchical approach to generating grammars for information extraction from HTML documents.

From a methodological point of view, in TaMeX, we explore the use of XML and its related technologies. XML is used as a declarative modeling language for a mediator's domain ontology and task structure, and as an intermediate data structure for information exchange. XPath expressions are used for representing wrapper extraction rules. In addition, XSL stylesheets are used for generating a mediator's user interface.

Acknowledgements

First, I would like to express my appreciation to my supervisor, Dr. Eleni Stroulia, for her insightful suggestions and invaluable comments guiding me towards the completion of this project, for her constant encouragement and trust throughout the course of this work, for her enthusiasm about her work inspiring me to become a serious researcher, and for her being my mentor and friend teaching me the lessons of more than just science.

I thank Roland Penner for his continuous patience and assistance. His experience, intelligence and love of his work gave me tremendous help in the implementation of the working system.

I thank Judi Thomson, Paul Iglinski, Yangjun Chen, Yiqiao Wang, Babita Rana and the rest members in the Software Engineering Research group for their assistance, suggestions and comments.

I also thank Huaxin Vivian Wei, Yanxia Jia, Shu Lin, and Wenbin Alan Ma for their friendship that made my life outside the lab joyful and memorable.

Finally, I thank my parents and my sister for their unconditional love and support in their own different ways. I give my special thanks to my mom for her sacrifice and understanding, and for giving me the opportunity and strength to become who I am. Mom, you're the wind beneath my wings.

Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

As the size of the World Wide Web (WWW) rapidly increases, the scope of the activities based on it expands. An enormous number of Web-based applications have become available to the public. These applications offer information and services for a variety of activities, such as stock-market trading, insurance purchasing, medical-prescription ordering and travel planning. With the advent of electronic-commerce, the need for interoperation of these applications arises for enabling both business-to-business and customer-to-business types of electronic commerce activities. For instance, suppose that a customer buying holiday gifts from an on-line store wishes to have them shipped to their recipients. Furthermore, he wishes to use his personal shipping service instead of the default service offered by the on-line store, because he has a particular large-volume discount. The question that arises then is how an aggregate application could be developed to combine these two currently separate on-line applications, so that the customer could painlessly arrange the desired combination of services.

The interoperation of Web-based applications is a challenge, because there exists neither an agreed upon representation and semantics for the information that these applications require and provide, nor a uniform access mechanism for the services they offer. Even those that provide the same or complementary services employ inconsistent terminology and incompatible interaction models. These problems are due to the original metaphor underlying the design of the Web, i.e., document publishing. The present language of the Web (HTML) provides limited information on the structure and presentation of the documents, but does not specify the semantics of the information presented. Existing thin-client applications provide HTML forms to their users' browsers, sometimes enhanced with client-side scripts in different languages. Assuming that the user appropriately interprets the semantics of the information required by the form and fills it out correctly, the server application responds with another HTML document containing information that the user can interpret as the answer to his original request. It is up to the user to combine the information in the responses of multiple applications and to use it to formulate new requests to yet other applications.

The object-oriented based integration frameworks, such as CORBA and DCOM, provide protocols for repackaging existing applications as object libraries to support agreement of low-level representation and behavior. However, they make no agreement on the semantics of the information exchanged among the applications. The TaMeX (***Ta**sk-based **Me**diation through **X**ML*) project has been motivated by the methodological assumption that the declarative representations for domain-specific semantics, such as representations based on the eXtensible Markup Language [18] (XML), is an approach fundamentally superior to the procedural integration advocated by the object-oriented integration frameworks. Moreover, it is particularly well-suited to Web-based applications. Semantically rich communication protocols can be developed to provide a "semantic glue" among the existing Web applications. Existing applications that do not "speak" these protocols can be "wrapped" with intelligent adapters; the role of each adapter will be to use the native interaction model of the application to "call" the appropriate application methods, and to translate the results of these calls into messages conforming to the communication protocol. In addition, specific types of intelligent intermediary brokers can be developed to understand and execute value-added services, such as price negotiation or comparison shopping.

This thesis investigates this XML-based declarative approach in the context of developing an aggregate application by integrating a set of domain-specific Web applications.

1.2 Background

The problem of integrating heterogeneous systems is not new. A variety of integration approaches have been proposed and developed for different instances of the general problem based on different technologies. Irrespective, of the underlying technology, an integration architecture should establish several types of agreement among the underlying resources that it integrates.

Research in database federation has identified representation and semantic agreement [38, 16] as necessary for semantic interoperability. Different applications may model the same domain in different ways, depending on the types of entities they choose to represent, their attributes, the relationships among them, and their semantics. The integration architecture should establish agreement among the divergent resource ontologies. That is, it should eliminate or bridge the differences among the application domain models assumed by the different database schema, as well as their representations and their semantics.

However, from the end-user perspective, the interaction model used by the user is more important. This is especially true in the case of e-commerce aggregate applications that cater to all types of users, not necessarily computer savvies. Many Web-based applications expose a cumbersome and counter-intuitive interaction model to their users. For example, applications built as interfaces to legacy systems often require navigation through several different documents and links to accomplish a task. Even applications specifically designed for the Web often suffer from a static interaction model and require a static set of inputs in a fixed order. To aggravate the problem, the order in which information is provided by the user and required by the different applications creates precedence relations, which the interoperation mechanism has to accommodate. An integration architecture should support behavioral agreement to provide a consistent model of interaction, i.e., communication and information exchange, between the resources and their common users. Such a consistent interaction model would be easy to learn, easy to remember, and easy to transfer across applications.

Object-oriented integration frameworks, such as DCOM and CORBA, have been proposed as mechanisms for creating interoperable applications. These frameworks provide a protocol for specifying, advertising, requesting and delivering services. The integration model for these frameworks proposes to repackage portions of (or even whole) existing applications as object libraries. Their services are specified using the framework IDL (Interface Definition Language) and then delivered to requesting clients by the Object Request Broker (ORB) of the framework. These mechanisms support low-level representation and behavioral agreement, but they make no provisions for specifying the semantics of the information exchanged. This implies that for every type of interaction between two applications, if they make different assumptions about the semantic content of the information they exchange, error checks or "translation" procedures have to be involved in one or both of the interacting applications. In addition, the repackaging of existing applications into components of the framework can require intrusive, possibly error-prone, modifications to the original application code. Finally, different implementations of these frameworks do not interoperate, and therefore an aggregate application developed with one implementation would not be easy to integrate with another.

The emergence of the eXtensible Markup Language [18] (XML) suggests a new solution to the problems of information integration. In recent years, much effort has been directed towards the effective use of XML as an intermediate representation for interoperating among heterogeneous resources. XML is a subset of SGML that allows user-defined tags expressing the semantics of the data content. The set of tags annotating the entities in a particular domain, their attributes and their hierarchical relationships is described in a DTD (Document Type Definition) or an XML Schema [42]. XML, in concert with the eXtensible Stylesheet Language [19] (XSL), which enables the customization of the information presentation based on its annotation, promises to replace HTML as the language of the Web of future. XML is fast becoming the de-facto standard for communicating on the Web. As more resources provide XML-specified data, information extraction from these resources will become easier. However, it seems unlikely that the existing body of material on the Web will soon be revised to an XML representation, making the advantages of user-defined semantic tagging unavailable to existing Web-based applications. This is why a lot of effort is currently being devoted to the post-publication translation of HTML documents into XML. Such translation processes enable the representation of information produced and consumed by different resources in a common vocabulary. This is a necessary but not sufficient step towards enabling the collaboration between resources. In addition, an integration infrastructure has to be developed for the coordinated communication of information between the user and all the underlying resources.

1.3 Problem Specification

The objective of this thesis is to develop a mediation architecture for task-specific interoperation among Web applications in the same domain. This architecture should relieve the user from the responsibility of translating and integrating information required and acquired when a complex request accesses multiple applications. To that end, two questions must be addressed:

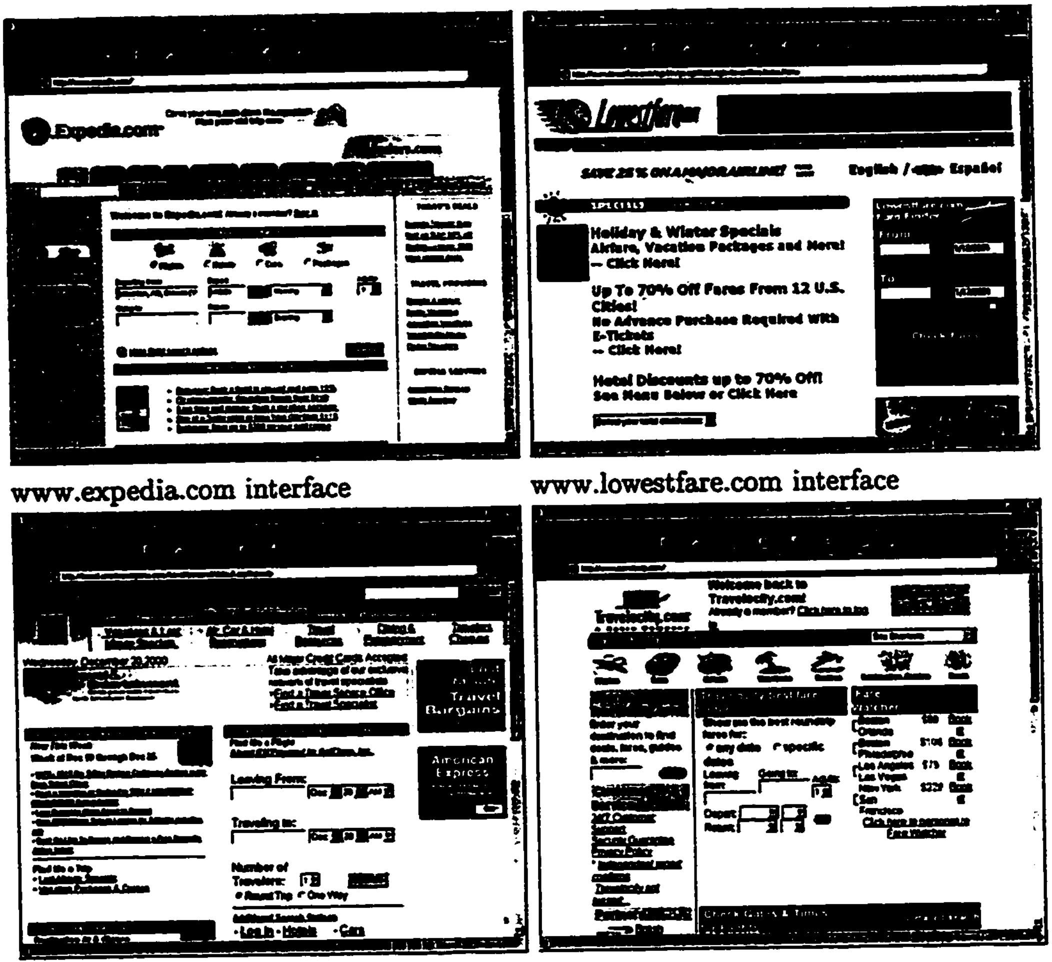
- How should the architecture enable the different applications share a common semantics for the information they consume and produce, so that this information can be unambiguously interpreted by all applications as well as human users?

- How should the architecture control the process of information exchange among the user and the applications, so that they can collaborate towards accomplishing complex user tasks?

1.4 An Example

In this thesis, we use the example of a travel planning assistant to illustrate the problems we are interested in and the solutions we provide. Today there are a multitude of Web sites offering travel planning and reservation services. Knowledgeable consumers with specific constraints and preferences must access several different sites to identify available options. Suppose the user wanted to find the lowest price of a round-trip air-ticket from Edmonton to London. He would have to go to *www.itn.net*, *www.travelocity.com*, *www.lowestfare.com*, *www.expedia.com*, and so on. He would have to submit his request several times, and then compare the results from these different sites prior to making a decision. The integration of existing travel-planning applications to support tasks such as comparative shopping is a compelling instance of Web-based application integration.

However, the integration of these travel planning Web applications is not a simple task. Although these applications are in the same domain, their interaction mechanisms and terminologies vary from one to another. For instance, as shown in Figure 1.1, *www.lowestfare.com* requires the user to select a specific airport if there is more than one airport close to his travel origin or destination, while *www.expedia.com* automatically uses all the nearby airports as the origin or destination to search the desired flights; *www.itn.net* allows the user to specify the preferred travel time on an hourly basis, *www.expedia.com* provides only 'morning', 'noon' and 'evening' as options, and *www.travelocity.com* does not provide any options; *www.lowestfare.com* requires the departure and return dates in the format of 'month/day/year', and



English / Español

Holiday & Winter Specials
Airfare, Vacation Packages and More!
-- Click Here!

Up To 70% Off Fares From 12 U.S. Cities!
No Advance Purchase Required With E-Tickets
-- Click Here!

Hotel Discounts up to 70% Off!
See Menu Below or Click Here

Wednesday, December 20, 2000

Leaving From:

Traveling to:

Number of Travelers:

● Round Trip ○ One Way

American Express

Welcome back to Travelocity.com!
Already a member? Click here to log in

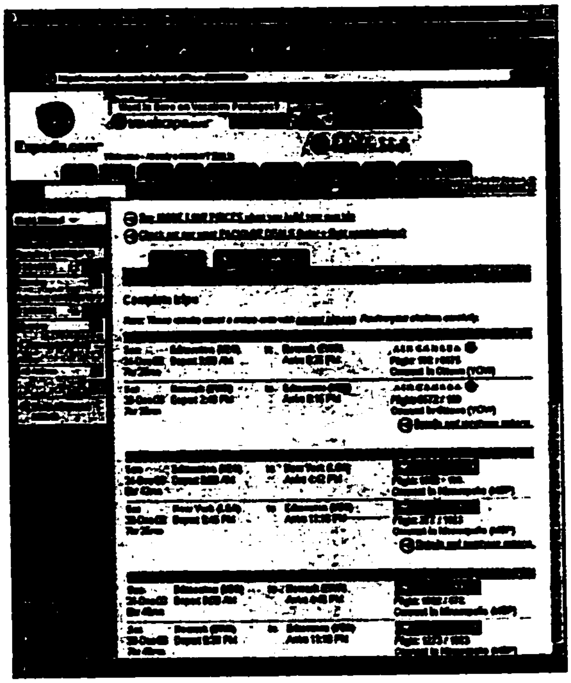
www.expedia.com interface

www.lowestfare.com interface

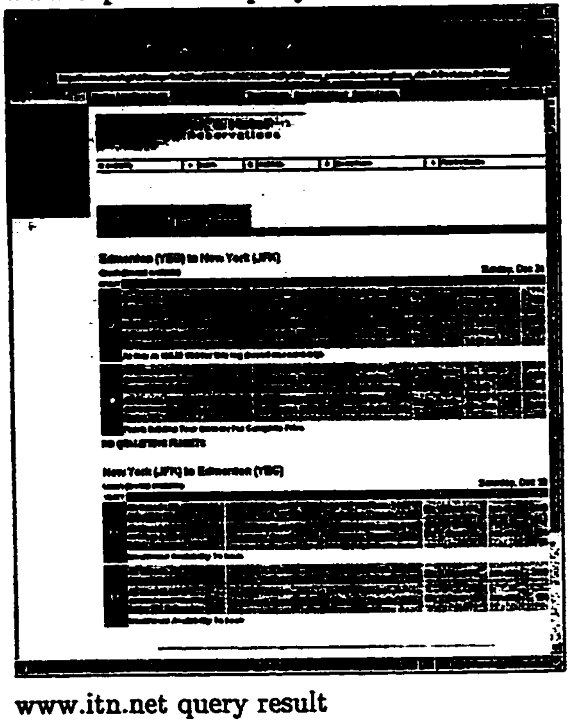
www.itn.net interface

www.travelocity.com interface

Figure 1.1: The Interfaces of Various Travel Web Sites



www.expedia.com query result



Edmonton (YEG) to New York (JFK)

New York (JFK) to Edmonton (YEG)

www.itn.net query result

Figure 1.2: The Different Query Result Presentations

www.expedia.com allows the user to select the dates by an online calendar. In addition, the resulting airticket information returned by these applications are presented in different formats and schema. For example, as shown in Figure 1.2, *www.itn.net* groups all the options of either leg of every ticket in a table, and allows the user to build his own trip by selecting one option from either table; and *www.expedia.com* displays every trip with information of both legs while hiding the details of each flight in links.

These variations and inconsistencies make it difficult to perform a task such as comparison shopping based on these different Web sites. Specific knowledge is required by another application to access the service and to interpret the information provided by each individual application. Therefore, additional software is needed for the transformation of these Web sites, so that they can be accessed with the same type of interaction method and can present the information with the same style of organization.

Another problem involved with the integration of these travel Web sites is how their services should be integrated and managed so that they work together to provide extra functionalities and to accomplish complex goals, such as "finding a cheap ticket to California in this summer". To accomplish such tasks, extra Web resources, such as an online airport database and a calendar, are needed to provide the additional information. Furthermore, an integration infrastructure is needed to support the collaboration among these different resources.

TaMeX aims to solve the above problems by providing a structured process and a set of supporting tools to the user who needs to develop an aggregate system based on a set of Web applications.

1.5 Anticipated Contributions

The central contribution of this thesis is the development of TaMeX, a mediation architecture for task-specific interoperation among domain-specific Web applications through XML.

In this architecture, Web-based resources providing information and services in a particular domain are encapsulated within wrappers. The role of a wrapper is that of an adapter between the resource's original API and a new API based on a common XML schema for the domain. The wrappers interact with a mediator that acts as an intelligent and task-specific intermediary between the user and all the wrapped resources. The mediator is responsible for acknowledging the capabilities of the wrappers and for coordinating their collaboration by appropriately communicating with them and invoking their services. The mediator interacts with the user, receives and elaborates the specification of the user's task, sends requests to the wrapped resources for solutions to sub-problems, post-processes the responses of the wrappers and combines their responses into solutions for the complex user's task.

Specifically, this thesis makes the following contributions.

- An XPath-based wrapper construction environment that supports a user in

    1. defining the XML schema of the information of interest,
    2. constructing parsers for sending requests and for extracting data of interest from HTML pages,
    3. and composing this data into XML documents conforming to the target schema.

- A task-structure centered integration infrastructure environment for developing new applications on a set of existing Web resources in a specific domain, so that they communicate in a common XML-defined language and collaborate towards complex tasks.

1.6 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 introduces the recent research work in the area of information integration of Web applications. The research mainly focuses on two types of problems, wrapper construction and integration infrastructure. Chapter 3 describes the process of developing an aggregate system in the TaMeX environment, including model development, mediator construction and wrapper generation. Chapter 4 illustrates the roles and run-time behaviors of each component through a travel assistant mediator, a prototype developed in the TaMeX environment. Finally, Chapter 6 concludes the thesis with discussions and evaluations of TaMeX, and summaries of the future work and the contributions of the thesis.

Chapter 2

Related Work

In general, there are two types of research work in the area of information integration of Web applications.

- **Wrapper construction** to support a user to define the schema of the information of interest, to construct parsers for generating requests and for extracting data of interest from HTML pages, and to compose this data into documents conforming to the target schemas.
- **Integration infrastructure** for designing new applications based on existing resources to be wrapped so they are able to communicate in a common vocabulary and collaborate to accomplish complex tasks.

2.1 Wrapper Construction

A wrapper is a bi-directional procedure for translating an application problem to a resource-specific query, and extracting tuples of information from the response presented by the resource. A wrapper for a Web resource bridges an application and a Web site by composing a resource-specific query in HTTP protocol using the application input data and then mapping the HTML document responses to a pre-defined schema.

The database community has been focusing on the development of SQL-like languages [2, 40, 13] for querying Web sites, so that other applications are able to access Web resources as if they were database. However, they make little effort on mapping HTML documents to other structures. People in the AI community, on the other hand, focus on information extraction from HTML documents. Many wrappers have been developed based on the techniques of natural language processing.

They parse HTML documents through a linear inspection and attempt to identify irrelevant parts at the beginning and the end of the HTML document, and generate rules for parsing the rest for potentially useful information. Dynamically generated documents can be a challenge for such approaches, since the size, the appearance and the types of information contained in these irrelevant parts may change from request to request. Another widely used approach is to identify the delimiters or landmarks of the start and end of the data of interest in the HTML document while parsing the document sequentially through a finite-state automaton.

In the area of wrapper construction, there are two dimensions of research focus. One emphasizes on the development of toolkit

- for providing users wizard-like interfaces to define and test the extraction grammars and mapping rules to convert HTML documents to desired format structures such as XML, and
- for generating wrappers based on these grammars and rules.

The other dimension of research works on how to automatically or semi-automatically learn wrappers with various techniques in artificial intelligence. To learn a wrapper for a specific target Web source, the learning samples include the example queries, the response pages corresponding to the example queries to the source, and the labeled instances of the target concept on the pages. The output is a wrapper that is able to query the source and extract all the concept instances from all the pages of the same type as the learned pages.

The following lists some of the most recent works in this area. Among them, W4F and XWRAP concentrate on the development of toolkit and the rest address the wrapper learning problem.

2.1.1 W4F

W4F (World Wide Web Wrapper Factory) [3, 4] is a development environment for constructing wrappers for Web sources. It offers a set of wizard-like visual tools for a user to develop and test a wrapper by specifying the rules and grammars.

W4F provides two expressive languages, HEL (HTML Extraction Language) and NSL (Nested String Lists), allowing a user to describe the extraction grammars and store the extracted data, respectively. HEL is a DOM-centric language, like XPath [41], though more expressive. It allows for the extraction of data in complex structures using index variables, conditions, regular expressions, and some simple operators, such as 'split' and 'match'. NSL is a data structure used for storing extracted data. It has its own API for manipulating data. W4F also supports a declarative mapping from NSL to XML documents or JAVA objects with automatic generation of the corresponding pre-defined structure, such as JAVA objects and XML. After the user is satisfied with the extraction grammars and the mapping rules, they are compiled into a JAVA component to generate a wrapper.

The wizards provide user assistance during the process of wrapper construction. The extraction wizard annotates each text field of the HTML document with its canonical path in the document tree, and allows the user to 'see' the extraction rule of the text of each of the instances of the desired attribute. The user then generalizes the rule manually using HEL to generate the rules for extracting all the instances of the desired attribute in the HTML document. Another useful wizard is the one used for testing and refining the wrapper by visualizing the three steps of wrapping a Web source in XML, namely fetching the HTML page, extracting the data and mapping to the schema. Correspondingly, the user specifies the Web source and request method, the extraction grammar in HEL, and the mapping rules to the desired XML schema. The wizard displays the extracted data in NSL and in XML. The user is then able to refine the grammars and the rules to achieve the desired result.

The main contributions of W4F are

- the visual tools for a user to 'see' every step of the construction process, and
- the expressiveness of the languages that allow the manipulation of the data through HEL and the NSL operators.

The limitations are the use of tags as delimiters instead of as containers, and the lack of semantics in the rules. For instance, the information 'hidden' in an attribute node would be overlooked. The information delimited by any punctuation could not be distinguished.

2.1.2 XWRAP

XWRAP [29] is a project similar to W4F. It is an XML-enabled semi-automatic system for constructing wrappers for Web sources using XML as an intermediate data type for information exchange. It provides an interactive interface with a library of commonly used functions and some source-specific facilities. It generates the extraction rules by allowing a user to identify the key words and the presentation layout structure. It targets only three types of regions in HTML documents: tables, lists and paragraphs.

XWRAP consists of four components for constructing wrappers through four interactive phases. Step one, *Syntactical Structure Normalization*, fetches the target web document as a sample, cleans up the errors and parses it into a tree. The *Information Extraction* component derives the extraction rules through a sequence of user interactions. A user identifies the interesting regions and semantic tokens on the page, and the system determines the hierarchical structure for the content presentation of the page. This component generates a set of declarative extraction rules in XML. The third component, *Code Generation*, applies the extraction rules to produce the wrapper program. The last phase is *Testing and Packing*, which allows the user to supply a set of alternative URLs for testing the generated wrapper program, and view the result. The wrapper is released and packaged once the user is satisfied with the result.

In XWRAP, the extraction rules are also DOM-based. But unlike W4F, in XWRAP, the tags are used to identify regions instead of simply as delimiters. In addition, the semantic tokens play an important role in XWRAP for identifying the data of interest.

2.1.3 Kushmerick et al.

Kushmerick et al. [33, 34] categorize wrappers into six different classes of complexity, and describe inductive learning [1] algorithms for the construction of wrappers of each type. For the wrapper learning problem, instances correspond to pages, labels correspond to the content of the pages, hypotheses correspond to wrapper template parameters, and oracles correspond to sources of the example queries and labels on the responses. Oracles are divided into PageOracles, which generate example pages from a specific Web resource, and LabelOracles, which produce correct labels on the example responses through some reusable, domain-specific heuristics, i.e., recognizers.

They identify six classes of wrappers, among which the Left-Right (LR) is the

[1] Inductive learning is learning by example - where a system tries to induce a general rule from a set of observed instances.

least complex. This class of wrappers is for extracting data from pages in which every element of the instance tuple is indicated by specific left- and right-hand delimiters. Therefore, the wrappers can be characterized by a vector of delimiters. Learning such wrappers involves collecting all the valid left- and right-hand delimiters for each element, while executing such wrappers involves scanning the document text sequentially to look for the left- and right- delimiters. The LR wrapper class is simple and very restrictive. The variants are classified into five groups as follows.

1. The Head-Left-Right-Tail (HLRT) class extends the LR class for extracting data from pages having the additional head and tail delimiters indicating the regions where all tuples are located.

2. The Open-Close-Left-Right (OCLR) class extends the LR class for extracting data from pages having the additional open and close delimiters indicating the start and end of each tuple.

3. The Head-Open-Close-Left-Right-Tail (HOCLRT) class has the properties of both the HLRT and OCLR classes.

4. The Nested-Left-Right (N-LR) class extends the LR class for extracting data from pages with nested structures, unlike the classes above that are for tabular pages.

5. The Nested-Head-Left-Right-Tail (N-HLRT) class, the most complex, extends both the HLRT and N-LR classes for handling more complicated pages.

All of the above classes are delimiter-based linear finite-state automata, where each delimiter is a state with two out-going edges, one for extracting texts and the other for skipping. According to the experiment, these six classes of wrappers are able to handle 70% of the surveyed Web sites. The learning algorithm of each class is similar to that of the LR class with certain degrees of variations.

2.1.4 SoftMealy

SoftMealy [12] presents a wrapper representation formalism. Similar to Kushmerick's approach, SoftMealy also uses a finite transducer to model the wrapper behavior. Instead of the delimiter-based linear approach, it discovers contextual rules from training examples to describe the context for separating two adjacent attributes, and encodes every distinct attribute permutation as a successful path of the transducer. This approach is less restrictive than Kushmerick's, which does not accept attribute permutation and which does not work well where there are no tags separating two distinct attributes. SoftMealy is more expressive in the sense that it is able to tolerate Web pages with problems like missing attributes, multiple attribute values, variant attribute permutations, exceptions and typos. However, SoftMealy has to "see" all the cases in the training examples in order to create the edges covering them.

2.1.5 Stalker

Stalker [24, 25] is a wrapper induction algorithm developed in the project Ariadne described in 2.2.1. It takes a hierarchical approach to information extraction, based on an embedded catalog formalism for modeling the hierarchy of semi-structured documents.

In this formalism, a semi-structured page can be described as a tree with the items of interest as leaf nodes, while an internal node is a homogeneous list with each element a heterogeneous tuple of leaf node(s) and/or list(s). Each edge in the tree is associated with an extraction rule, and every list node is associated with a list iteration rule. To extract an item of interest is to find a path from the root of the tree to the corresponding leaf by applying a set of extraction rules and list iteration rules. The set of extraction rules in Stalker is called a *disjunctive landmark automaton* which is used to consume the irrelevant text and search for the landmarks of the interests, which is similar to Kushmerick's use of delimiters. Stalker is a sequential covering algorithm. It takes, as the input, a set of prefixes that need to be consumed to extract the concept of interest, generates rules accepting as many positive examples as possible by repeatedly refining the rules with the uncovered positive examples.

While Stalker is able to handle missing attributes and various attribute orderings like SoftMealy, it does not require "see"ing all the possible cases of ordering in order to cover them. However, since each landmark automaton is only responsible for one attribute, a Web page has to be scanned several times during the run-time.

2.1.6 Chidlovskii et al.

Chidlovskii et al. [7] present an incremental grammar induction algorithm for wrapper construction with an adaptation of the concept of string edit distance.

"The *edit distance* $D(s_1, s_2)$ between two strings of symbols s_1 and s_2 is the minimal number of insertions or deletions of symbols, needed to transform s_1 into s_2." They adapted the algorithm to calculate the distance between a grammar and an instance. Before learning, the example pages are pre-processed by inserting tags to specifying the beginning and end of the first instance as well as every attribute to be extracted. When the learning starts, the algorithm takes one pre-processed Web page as input. It generates the initial extraction rule, called the item grammar, based on the labeled instance. Then it iteratively finds the next instance, applies the adapted string edit distance between the current instance and the first one to update the item grammar. The process halts when all the instances on the page are found. Finally the algorithm post-processes the grammar and returns it as the wrapper. An item grammar is represented by a simple finite-state automaton (sFSA), similar to SoftMealy. The wrapper execution takes two steps, locating the beginning and end of every instance, and extracting each attribute with the item grammar.

The main feature of this approach is the limited user interaction. Only one page is needed and only one instance is labeled. The grammar induction is based on the string edit distance, which sometimes makes big generalizations. Therefore, sFSA is able to tolerate any missing attributes or any different attribute orderings. But on the other hand, this might produce incorrect wrappers. Another drawback is, like Kushmerick's approach, this algorithm only considers tags as separators. Therefore, attributes not separated by tags cannot be extracted.

2.2 Integration Infrastructure

Information integration is not a new research problem. People in the database community have been working on database middleware systems [44, 11, 5, 30] for integrating data from multiple relational data sources. The resources we are interested in are Web applications. These applications provide semi-structured data, which is a set of data where the desired information can be located using a concise and formal grammar and its data model is more implicit. Ariadne, MIX and XIB are the recent examples of research in the development of integration infrastructure for Web resources. These systems focus on the collaboration among underlying resources and the evaluation of complex user queries.

2.2.1 Ariadne

Ariadne [43, 10] is a project that focuses on developing an infrastructure for integrating the wrapped resources of semi-structured documents using a planner.

In Ariadne, an HTML page is modeled by the embedded catalog formalism described in 2.1.5. An application source is modeled in terms of a common domain model which ties the source together with others. The run-time process is based on the models (the domain model and the source models) and a Planning-by-Rewriting (PbR) method. The entire query process has two phases, a preprocessing phase and a planning phase. During the first phase, the system finds all the possible ways of combining the available sources to answer the query by an source selection algorithm, which, based on the domain model and the source models, dynamically selects the source candidates according to the attributes in the query. In the second phase, Ariadne first generates a sub-optimal plan from the candidates, and iteratively improves it via a local search process.

The modeling-source-by-domain-model approach makes the system extensible and flexible. However, it is unable to reason over recursive relations, such as a page with a 'more' button or a site with variable depth of hierarchy like Yahoo.

2.2.2 MIX

MIX (Mediation of Information using XML) [8, 9] is a project that aims to develop systems for mediation across heterogeneous information sources, including databases, GIS systems and Web sites. MIX takes a database-centric approach to integration and develops a query planning mechanism for querying various resources viewed as XML databases.

In MIX, data exchange and integration relies entirely on XML. Instances are stored in XML documents; the domain ontology and individual sources are represented in XML DTDs; and queries are expressed in XMAS, a XQL-like query language with features for grouping and ordering new XML "objects" from the existing ones. Query execution is demand-driven through an interactive GUI, called BBQ (Blended Browsing and Querying). It automatically generates an XML query from the definition given by users through navigating the tree structure of the DTD in the mediator view and adding desired constraints to corresponding attributes. Complex queries are then unfolded based on the individual source DTDs resulting in simpler queries against the underlying resources. The results are presented to users as a DOM tree for browsing.

The integration in MIX is based on navigation-driven lazy mediators. A mediator translates a complex user query, which is expressed as navigation on an XML view, into an algebraic plan. Each operator of the algebraic plan acts as a lazy mediator. This decomposition process finally produces a tree of lazy mediators with a set of simpler algebraic plans corresponding to navigation on underlying sources. Thus, an integrated query evaluation scheme is generated. Like Ariadne, algebraic plans are optimized with respect to navigational complexity. Nevertheless, the order of operators in a plan is deterministic, which limits the degree of optimization.

2.2.3 XIB

XIB (eXtensible Information Brokers) [27] is another project that utilizes XML technologies for integration of Web services. XML DTDs are used for modeling Web services. XSL is used for translation of information between HTML and XML. XML-QL is used for composing query results.

In XIB, a mediator is considered as an information broker. XIB provides tools for supporting interactive generation of wrappers and an integrated query interface. A *WrapperBuilder* helps a user to wrap up a service by providing a service description and registering the service in the service server. A *BrokerBuilder* allows a user to define a broker by selecting the services to integrate and the logic of the integration.

XIB provides an information service description language, called XIBL, to model Web services in terms of XML. It uses DTDs for the input and output of the service, and uses XML elements for the input and output values. A service description includes information on the location of the service, the types of input and output of the service, and the information extraction rules that are defined in Compaq's Web Language [22]. As a consequence, the integration of Web services in XIB is viewed as the integration of DTDs in the form of XML-QL, and the decomposition of user queries is translated to the decomposition of XML elements. The service description of a Web application to be integrated is critical in XIB. However, XIB does not provide any support to its construction.

2.3 Comparison with TaMeX

In the research and development of TaMeX, we are equally interested in both the wrapper construction problem and the integration infrastructure problem.

We have developed an example-based wrapper-construction process. Unlike the projects mentioned in the previous sections, which mainly focus on the extraction of the result information, our work also addresses the problem of learning resource protocols with a simple XML-based language. In terms of the learning of information extraction, the main feature of the TaMeX process is that it is designed for more complex types of target concepts and pages. Documents may contain multiple instances of the target concept intermingled with an unknown amount of irrelevant information. According to Kushmerick's definition, TaMeX is able to generate N-HLRT wrappers, which is the most complex class in that system. In TaMeX, an HTML page is represented by a DOM [14] structure. The wrapper learning is achieved semi-automatically by observing the regularities of the tree structure of the document, instead of collecting the delimiters or separators of the target concept. The data extraction takes a hierarchical approach based on XPath, instead of the sequential approach taken by SoftMealy and Chidlovskii et al., which are based on finite-state automata. Unlike W4F, which provides only assistance and still requires significant effort from the developer to construct the extraction grammar, the TaMeX wrapper-construction process learns the grammar automatically from the concept instances provided by the developer without requiring the manual inspection of HTML documents.

In terms of the integration infrastructure, the TaMeX mediation architecture is task-structure centered. The task-structure of a TaMeX aggregate system reflects the run-time behavior of the mediator. In addition, it provides a roadmap for developing and integrating different components of the system. Similar to MIX and XIB, TaMeX heavily relies on XML technology. XML is used not only for information exchange and for domain and task modeling, but also for user interface generation. However, in contrast to MIX, which focuses on single-step queries, the TaMeX infrastructure aims at developing mediators in support of complex and multi-step tasks. In that sense, it is more akin to Ariadne and XIB, with the difference that the TaMeX mediators execute task-specific plans, instead of planning from scratch every time they receive a user request.

Chapter 3

The TaMeX Development Environment

The TaMeX development environment is designed to support a developer facing the problem of constructing an aggregate system by integrating a set of existing Web applications. These Web applications share a common domain but they may have slightly different models of the entities in this domain and they most likely have different representations for the their data. They interact with their users through forms, implemented in various underlying technologies, and, in response to their users' queries they return dynamically produced HTML documents containing multiple instances of the concept about which the users inquire. The developer's questions are:

- How should these different applications share a common semantics of the information?
- How should the collaboration among these applications be controlled?
- How could the information provided by any individual application be extracted, in order to be forwarded to the other applications?

The TaMeX environment provides a structured process and a set of supporting tools as the solutions to the above developer's questions. According to this process, the developer needs to follow these three steps.

1. Develop the XML specification for a canonical model of the application domain.
2. Design the task-structure model of the applications' interactions and develop the mediator.

3. Construct wrappers for the existing Web-based applications, that will enable the translation of information from(to) the canonical domain model to(from) their internal representations.

At step 1, the application domain is modeled by describing the application domain objects, their composition through domain-specific relationships and their constraints in XML. This domain model establishes a uniform vocabulary for communication among the application user, the mediator and the wrappers.At step 2, a mediator is developed by implementing the user interface and the logic of the integrated system process, based on the task structure of the aggregate system. The last step involves the construction of a set of wrappers corresponding to the existing applications, by learning a grammar for parsing the user queries to the application and the application responses to the user in terms of the canonical domain model.

The rest of this chapter describes each of these three steps in detail through a simple glossary example. Suppose that a certain developer[1] recognizes the need for an on-line internet glossary. However, none of the ones he has found is complete. To avoid the inconvenience of looking up several different glossaries, he decides to develop a new glossary by integrating two existing ones. The following are the steps he has to follow.

3.1 XML Specification of Domain Model Development

The domain model specifies the concepts in the application domain and the relationships among them. The role of the domain model is to establish a uniform vocabulary among the user, the mediator and all the wrappers of the integrated resources. This common model is used for specifying the information provided by the user as part of the problem specification, the information produced by the mediator as it elaborates the user problem, and the information provided to and by the wrappers.

The domain model consists of two components. One depicts the domain ontology and the other specifies the domain constraints. The domain ontology describes, in XML Schema [42], the hierarchical structure of the application domain. It specifies the objects, their attributes and the relationships among them, such as inheritance and composition. For example, in the glossary domain as shown in Figure 3.1, a

[1] The term "user" in this chapter refers to the TaMeX development environment user, i.e.e, the aggregate dictionary developer

```
<?xml version="1.0" ?>
<Schema name="glossary" xmlns="urn:schemas-microsoft-com:xml-data"
xmlns:dt="urn:schemas-microsoft-com:datatypes">
    <ElementType name="glossary" content="eltOnly">
        <element type="entry" />
    </ElementType>
    <ElementType name="entry" content="eltOnly">
        <attribute type="word" />
        <attribute type="definition" />
    </ElementType>
    <AttributeType name="word" dt:type="string" required="yes" />
    <AttributeType name="definition" dt:type="string" required="no" />
</Schema>
```

Figure 3.1: The Glossary Domain Model in XML Schema

glossary contains zero or more entries; each entry has one word and one definition. The purpose of this view is to define a uniform data schema for the information exchanged among the user, mediator and the wrappers. For a more complex domain model, such as the travel domain we will discuss later, there is also the domain constraints component. It describes in XML a set of semantic constraints on the domain objects and their attributes, and enhances the logical view of the domain. In the travel domain, the origin of each leg should not be the same as the destination, and the arrival time should be after the departure time after converting to the same time zone. The constraints are to be used by the wrappers to reconstruct the instances of the domain objects from a set of instances of the object attributes.

3.2 Development of a Task-structure Based Mediator

The mediator is the heart of a system developed within the TaMeX environment. Its role is to interact with the user of the integrated application, and to coordinate the communication among the wrappers of the underlying resources towards accomplishing complex user tasks. The TaMeX environment adopts a declarative approach to the development of the mediator and represents the mediator's processing mechanism with its task structure model.

3.2.1 Task Structure Development

Chandrasekaran [6] characterizes a task by the type(s) of information it consumes as input and produces as output, and the nature of the transformation it performs between the two. A complex task may be decomposed into a partially ordered set of simpler subtasks. A simple, non-decomposable task, i.e., a leaf task, corresponds to an elementary action. The control of processing moves from higher-level complex tasks to their constituent subtasks. Information flows through the task structure as it is produced and consumed by the tasks. The actual process is non-deterministic, because there may exist more than one decomposition for a given task, and the subtasks of each decomposition are only partially ordered. High-level complex tasks get accomplished when all their subtasks have been accomplished.

This view of processing can be naturally mapped to the run-time process envisioned for a TaMeX-developed integrated application consisting of a mediator and a set of wrappers. The behavior of the overall application can be modeled in terms of a non-deterministic task structure. The wrappers of the individual sources implement elementary leaf tasks using the services of their underlying sources. The mediator implements a set of high-level complex tasks, with non-deterministic decompositions. The user's interaction with the mediator decides which task is active at each point in time and which decomposition is employed. When a particular task is sufficiently decomposed into elementary tasks, the mediator requests the relevant wrappers to accomplish them and to return their results. In addition to decomposing high-level tasks into elementary ones, the mediator task structure also provides the road map for composing the individual wrappers' results in a coherent solution to the overall task at hand.

TaMeX adopts the SBF-TMK language [15, 16] [2] to describe the internal processes of the mediator and the wrappers. We have developed an XML representation for the SBF-TMK language, shown in Figure 3.2. In addition, we have also developed a general XSL Stylesheet [3] for presenting task-structure models as hierarchical menus: the high level tasks of the mediator correspond to the top level menus whose options become subtasks represented in sub-menus. This gives any specific mediator a simple intuitive interface through which the user can traverse the mediator's task structure and invoke the tasks that it can accomplish.

The task structure of a TaMeX mediator consists of three different types of tasks.

- User-interaction tasks refer to the tasks performing communications between

[2] SBF-TMK (Structure-Behavior-Function models of Task Methods and Knowledge) models are one of the various formalisms developed for representing task structures.

[3] Extensible Stylesheet Language (XSL) is a language with capabilities for presentation and manipulation of XML documents.

```
<!ELEMENT taskModel (task+)>
<!ELEMENT task ( input*, output*, (subtask+) | operator )>
<!ELEMENT subtask EMPTY>
<!ELEMENT operator (#PCDATA)>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ATTLIST taskModel name CDATA #REQUIRED>
<!ATTLIST task
    name CDATA #REQUIRED
    id ID #REQUIRED
    type CDATA #REQUIRED
    description CDATA #REQUIRED
>
<!ATTLIST subtask
    name CDATA #REQUIRED
    id IDREF #REQUIRED
>
<!ATTLIST operator
    xmlfilename CDATA #IMPLIED
    xslfilename CDATA #IMPLIED
    servlet CDATA #IMPLIED
>
<!ATTLIST input required (yes | no) #REQUIRED>
```

Figure 3.2: The DTD for Task Structure

the mediator and the user, including the mediator's requests for information from the user and display of information to the user.

- Information-collection tasks refer to the mediator's interaction with the wrappers of the underlying applications to collect information.
- Finally, internal tasks refer to information processing internal to the mediator. Such tasks include the translation of a user's problem to an XML specification understood by the mediator and all the wrappers, as well as the composition and post-processing of the information collected from the wrappers.

When designing a mediator in the TaMeX environment, the developer first identifies the exchanges of information between the mediator and the user of the aggregate application. These exchanges constitute the user-interaction leaf tasks of the mediator. The information exchanged between the user and the mediator is within the domain of the application. Therefore each screen of the interface through which the interaction is performed is easily constructed by writing an XSL Stylesheet to specify how the concepts in the domain are to be presented to the user.

The information-collection leaf tasks are identified by examining the services provided by the existing applications to be wrapped. The developer first determines a set of existing Web applications providing the functionalities required by the mediator. For each of the distinct functionalities provided by an existing application, a corresponding information collection task is identified and a wrapper is constructed to deliver the functionality specified by the task from the underlying source application. The wrapper construction process is described in Section 3.3. Any other desired functionalities, not accomplished by the existing resources, are specified as additional leaf tasks, for which, new components are developed to accomplish them.

Finally, every internal task is implemented individually within the mediator to perform the operation required. After having specified all the leaf tasks of the mediator, they are composed in a hierarchical task structure by introducing intermediate high-level tasks corresponding to the intermediate sub-goals of the overall process.

3.2.2 Mediator Development

The TaMeX mediator is designed as a client-server application. The thin client component is supported by an XML-enabled Web browser, such as Internet Explorer, to provide a user interface based on the domain model and the task structure of the aggregate system. The server-side component is implemented as a Java Servlet to support the logic of the mediator and the connection between the client-side component and the wrappers providing the desired services.

The lightweight user interface heavily relies on XML and XSL technologies. They are generated by applying XSL stylesheets to the domain model and the task structure expressed in XML. The main screen of the TaMeX interface is a menu presenting the higher level tasks in the task structure and allowing the user to select the tasks to perform. The menu is produced by a generic XSL Stylesheet conforming to the schema of the task model. This menu leads the user to a screen either requesting input from the user or providing output to the user. Both the input data and the output data are the instances of the concepts in the domain model. For every concept in the domain that must be specified by or provided to the user, there is a corresponding XSL Stylesheet defining the presentation of this information, producing a lightweight interface for the mediator and enabling its consistent interaction with the user.

The server-side component is implemented as a Java Servlet. It is connected to the browser through the $HTTP :: GET/POST(param_i*)$ request. Along with the input or output data exchanged between the client and the server, each request specifies the task invoked by the user. For each different task, a method is implemented to perform the appropriate action. If it is a user-interaction task, the mediator either receives the data transported with the request, or returns the data as the response of the request. If it is an information-collection task, it contacts the appropriate wrappers to retrieve the desired information. If it is an internal task, it processes the data accordingly. Thus, during run-time, the mediator is able to dynamically perform the tasks in the task structure.

3.3 Wrapper Construction

The most critical step in the process of developing an integrated application is the construction of wrappers for the existing Web applications.

In TaMeX, we exploit the hierarchical structure of HTML documents when inspecting their contents. HTML pages served by Web applications in response to a particular request are usually generated automatically. Although individual responses may differ in content, all responses from the same application for the same type of requests usually have a similar organization. This implies that two HTML

1. Let a user demonstrate the use of the resource to be wrapped and generate the training examples
   (a) Define a set of example problems for the user to demonstrate
   (b) For each such problem
   While the user at his browser interacts with the resource server to solve the problem, record the trace of the interaction and the final response page that contains the instances of the target concept that are the solution to the current problem.
2. Learn the protocol of the interaction with the resource
3. Learn the grammar for extracting the instances of the target concept from the resource's responses

Figure 3.3: The Overall Wrapper Construction Process

documents served by an application as a response to the same type of request are more likely to have commonalities at (and close to) the documents' roots than at (and close to) their leaves. Once the regularities of the sub-trees containing the data of interest are discovered, the HTML document can be parsed into a tree representation rooted at the <html> tag, and the information of interest can be efficiently located, using a DOM [14] API.

To wrap a Web resource, first one needs to know the protocol by which a request is sent to and the response is retrieved from the resource Web server. This ensures that requests of the same type can be automatically sent by the wrapper to the resource server in order that the correct responses can be obtained. Furthermore, one needs to find out the rules for locating the instances of the target concept of interest from a response, so that the data of interest can be extracted from the HTML document.

The overall process for wrapper construction in TaMeX is shown in Figure 3.3. It consists of a demonstration phase and a learning phase. The purpose of the demonstration phase (step 1 in Figure 3.3) is to collect examples, on the basis of which both the parameters of the method to be invoked on the underlying application and the grammar for parsing its response can be learned. The input to the demonstration phase is the specification of a set of problems and their solutions that the application can accomplish. The underlying idea is that the wrapper's information-collection task is already known when integrating a new application. Therefore, examples of this task, i.e., particular problems and their solutions, can be defined and solutions can be extracted from the applicatiŏn responses.

The demonstration phase relies on a proxy based on the Muffin library [32]. The purpose of the proxy is to record the "conversations" between the application server and the user's browser. While the user sends a sequence of requests to the server to perform the functionality of the resource to be wrapped, the proxy "sits" between the browser and the server, and keeps track of their interactions, including both the requests sent by the browser and the responses returned by the server.

The next phase of the wrapper-construction process is the learning phase (steps 2 and 3 in Figure 3.3). The objective of this phase is twofold. First, the resource protocol for sending the resource server a request corresponding to a user's problem specification must be learned. Second, the grammar rules for parsing the response into the solution corresponding to a user's problem must be learned. The details of these two learning processes are described in Section 3.3.1 and Section 3.3.2. The knowledge learned about the resource is expressed as XML documents. They are passed as input into a generic wrapper to generate a wrapper specific to this resource. For details on the generic wrapper, see Section 4.2.

3.3.1 The Protocol Learning Process

The purpose of a wrapper in the context of an aggregate application is to automatically perform translation on behalf of the user. Initially, a wrapper needs to know how to translate a user's problem to a request to its corresponding Web resource. In the TaMeX environment, the resource protocol is learned by observing the traces of the "conversations" between the browser and the resource server captured during the demonstration phase.

During the demonstration of sending a request to the Web resource, the proxy records the trace including a sequence of HTTP requests sent by the browser to the resource server and a sequence of responses returned by the server to the browser. TaMeX is only interested in the first request that corresponds to the user's action of sending the request that initializes the subsequent interactions between the server and the browser. The recorded request contains the resource URL location, the request method, and the request parameters. For the same type of service provided by the same Web resource, the URL location and the request method are constant across different problem specifications. Some of the request parameters are part of the variables in the problem specification, and vary with the different requests.

Then, the variables in the XML problem specification are mapped to the parameter values in the corresponding trace. To a learned protocol, the parameter values in a trace are replaced by the XPaths for locating the corresponding variables in the XML problem specification.

The learned protocol for a specific resource is represented in XML. It specifies the resource URL, the request method, and a set of method parameters. For those parameters whose values vary with different problems, the mapping rules are specified in XPath for converting the parameters of a problem specification to those of the corresponding request method. During run-time, a specific request can be produced by assigning to the parameters the values located in the problem specification by using the mapping XPath rules

As mentioned previously, often, a single user's request might result in a sequence of request-response "conversations" between the user's browser and the resource server. This is because the server sends to the browser cookies that the browser uses to resubmit its request. Cookies are automatically handled by the run-time wrapper and need not be learned at the development stage.

3.3.2 The Grammar Learning Process

The more difficult, and arguably more important, step of wrapping a Web source is to discover the grammar for extracting the information of interest from the HTML documents and converting it into a desired schema. In TaMeX, we take a hierarchical approach to semi-automatically learning extraction rules based on the tree-structure of HTML documents. In the following, we first describe the properties of the HTML documents we are interested in, and then introduce an algorithm for learning to extract data from these types of documents.

A classification of HTML Documents

Our approach to wrapping Web applications relies on the use of XML technology for creating a semantic map of document segments. As we have already mentioned, the mediator's domain model is represented in XML. When the mediator sends an information-collection request to a wrapper, it expects to receive an XML document containing instances of some entity in the domain model[4]. Let the target concept C_t

[4] There can be instances of more than one entity of interest contained in the response. However, for the sake of clarity of the discussion we use the singular number. Our method can also deal with multiple target concepts.

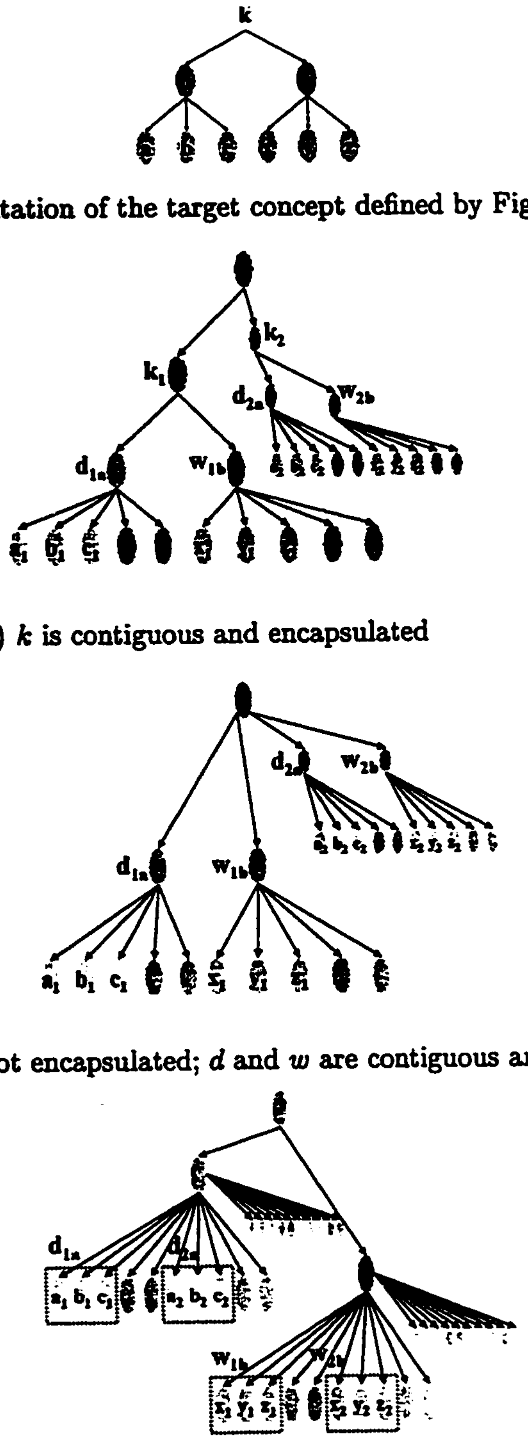
be the entity of the domain model, instances of which must be extracted from the responses of the resources to be wrapped. It is important to note here that the TaMeX environment focuses on wrapping applications of the target concept.

XML documents are hierarchical in nature. An XML element is composed of simpler XML elements. In order to extract an instance of the target concept from the application response, the instances of all the constituents of the target-concept must be identified. Most Web-based applications provide information encoded using HTML. Our wrapper construction process assumes that the resource response is a well-formed HTML document, possibly "cleaned up" using an application like the HTMLTidy [39] utility. Presently it cannot automatically create wrappers for information that would normally require a browser plug-in for presentation (e.g. PDF or Shockwave). Henceforth, the term resource response R will refer to an HTML document returned by an application in response to an invocation of some $HTTP :: GET/POST(param_i*)$ method. A further assumption of the wrapper-construction process is that several instances of the domain model concepts can be extracted from the response and mapped into a well-formed XML document.

Figure 3.4(a) illustrates a hypothetical target concept k that consists of two simpler concepts, d and w, which in turn get decomposed into a, b, c and x, y, z respectively. The set of k's constituent components consists of all the concepts represented by the trees rooted at k, d, w, a, b, c, x, y, z. The DTD of this concept is shown in Figure 3.5.

Let us assume that there are at least two instances of k (and consequently two instances of each d, w, a, b, c, x, y, z) in a particular HTML document produced as a response of the application to be integrated. If the constituents of one instance of concept k are separated from all the parts of all other instances of k, i.e., the instances of the constituents of the different concept instances are not intermingled, then k is said to be contiguous.

The three trees shown in Figure 3.4(b), Figure 3.4(c) and Figure 3.4(d) represent the internal structure of three different HTML documents. Each circular node in the tree represents a pair of matching HTML tags enclosing a part of the document. k is contiguous in the trees of Figure 3.4(b) and (c). However, in the tree (b), it is also encapsulated, i.e., there is a distinct pair of HTML tags completely enclosing all of the parts of one instance and none of the parts of any other instance. k is not encapsulated in the tree (c), but d and w are. The tree (d) represents a case where



(a) A tree representation of the target concept defined by Figure 3.5

(b) *k* is contiguous and encapsulated

(c) *k* is contiguous but not encapsulated; *d* and *w* are contiguous and encapsulated

(d) *k* is not contiguous; *d* and *w* are contiguous but not encapsulated

Figure 3.4: A Domain-model Entity and its Instances in the HTML Responses of Three Applications

```
<!ELEMENT k (d, w)>
<!ELEMENT d (a, b, c)>
<!ELEMENT w (x, y, z)>
<!ELEMENT a (#PCDATA)>
<!ELEMENT b (#PCDATA)>
<!ELEMENT c (#PCDATA)>
<!ELEMENT x (#PCDATA)>
<!ELEMENT y (#PCDATA)>
<!ELEMENT z (#PCDATA)>
```

Figure 3.5: The DTD for the XML Document in Figure 3.4(a)

```
<root>
  <k>
    <d>
      <a>a1</a>
      <b>b1</b>
      <c>c1</c>
    </d>
    <w>
      <x>x1</x>
      <y>y1</y>
      <z>z1</z>
    </w>
  </k>
  <k>
    <d>
      <a>a2</a>
      <b>b2</b>
      <c>c2</c>
    </d>
    <w>
      <x>x2</x>
      <y>y2</y>
      <z>z2</z>
    </w>
  </k>
</root>
```

Figure 3.6: The XML Document for the example shown in Figure 3.4(b)

```
<root>
  <d>
    <a>a1</a>
    <b>b1</b>
    <c>c1</c>
  </d>
  <w>
    <x>x1</x>
    <y>y1</y>
    <z>z1</z>
  </w>
  <d>
    <a>a2</a>
    <b>b2</b>
    <c>c2</c>
  </d>
  <w>
    <x>x2</x>
    <y>y2</y>
    <z>z2</z>
  </w>
</root>
```

Figure 3.7: The XML Document for the Example Shown in Figure 3.4(c)

```
<root>
  <node>
    <a>a1</a>
    <b>b1</b>
    <c>c1</c>
    <a>a2</a>
    <b>b2</b>
    <c>c2</c>
  </node>
  <node>
    <x>x1</x>
    <y>y1</y>
    <z>z1</z>
    <x>x2</x>
    <y>y2</y>
    <z>z2</z>
  </node>
</root>
```

Figure 3.8: The XML Document for the Example Shown in Figure 3.4(d)

k is not contiguous, i.e., the component instances of the different concept instances are interspersed in the HTML page. The corresponding XML documents of these three tree structures are shown in Figure 3.6, Figure 3.7 and Figure 3.8.

In order to extract the instances of k from an HTML document, a set of rules must be formulated specifying how to traverse the tree-structure of the document in order to locate the instances of the constituents of k. Because the tree is traversed hierarchically, the rules can be formulated using XPath [41] expressions. Therefore, the main objective of the wrapper construction process is to learn a grammar, specified as XPath expressions, for the locations of the instances of the leaf components in the target concept set.

The contiguity and encapsulation properties of the target concept in the application response have an interesting effect on the structure of the grammar. If the concept is encapsulated, then an XPath rule can be formulated in order to localize the HTML subtree that spans all the concept constituents. The rules for its constituents can then be formulated relative to the root of this subtree. If the concept is not encapsulated but it is contiguous, then a set of indexed XPath rules can be formulated in order to locate its constituents[5]. Finally, if the concept is not contiguous, then rules for the extraction of its elements must be discovered and then the constraints describing the invariants of the concept may be used to combine them into concept instances. Next, we discuss the nature of these rules and the algorithms to learn them in detail.

The Grammar Learning Algorithm

Each trace obtained from the demonstration phase includes the application's response. The second objective of the learning phase is to learn the grammar for extracting each constituent of the target concept from the response. In TaMeX, we focus on the tree structure of an HTML document and take a hierarchical approach based on its XPath [41] expressions.

The HTML documents that are generated by the application as responses to the user's requests are produced automatically by programs. The page presentation is structured, i.e. the organization of each instance of the target concept is the same. This means that each instance has the same structure if viewed as a document sub-tree, although the text content within each sub-tree is different. Therefore, the

[5] The underlying assumption is that the concept instances have a common internal structure.

```
1. FOR EACH instance, $p$, of the concept $C_t$
   (a) locate the components of the instance
   (b) identify the XPath rule, consisting of the index and the attribute list, for the location of the instance components
   (c) identify the XPath rule, consisting of the index and the attribute list, for the location of the instance as the minimum spanning tree[6] of its components locations
   (d) formulate the locations of the instance components, relative to the instance location
       i. IF the path expression of its parent component is the prefix of the current path rule, remove the prefix from the current hypothesis
       ii. IF it is a parent component, generalize the index of the last tag of the path to include all instances
2. FOR EACH index of every rule hypothesis
   (a) IF it is constant, remove its associated attribute list
   (b) IF the set of XPath rules for the examples vary only by index number, formulate one of the following (and increasingly general) hypotheses for the rule $xp(C_t)$
       i. FOR EACH variable index, generalize the expression to a domain containing only 2 possible values
       ii. FOR EACH variable index, generalize the expression to a linear progression of the index values
       iii. FOR EACH variable index, generalize the expression to include all possible index values
```

Figure 3.9: The Algorithm for Learning Grammar for Encapsulated Concept

paths from the root of a document to the different instances of a concept all consist of the same sequence of HTML tag nodes but with different indices for some of the tags. Our goal is to discover the regularities of the indices and find the general path in the HTML document leading to all the concept instances.

The algorithm for learning how to extract instances of an encapsulated concept from the HTML response is shown in Figure 3.9. Given a set of specific examples of the target concept and a response that is known to contain the given examples, the learning algorithm generates a set of XPath rules that will extract the information for all instances of the target concept from resource responses with the same structure. The following discussion illustrates the given algorithm with a small example from a fictitious glossary Web page.

```
<html><body>
  <table>...</table>
  <table>...</table>
  <table>...</table>
  <table>...</table>
  <table border=“1”>
    <tr><td colspan=“2”>A Fictitious Glossary</td></tr>
    <tr><td>word</td><td>definition</td></tr>
    <tr bgcolor=“red”>
      <td>abbey</td>
      <td>convent</td>
    </tr>
    <tr bgcolor=“red”>
      <td>abduct</td>
      <td>kidnap</td>
    </tr>
    <tr>...</tr>
    <tr>...</tr>
    <tr bgcolor=“red”>
      <td>capital</td>
      <td>money</td>
    </tr>
  </table>
</body></html>
```

Figure 3.10: The HTML document of the fictitious glossary

```
<dictionary>
  <entry>
    <word>abbey</word>
    <definition>convent</definition>
  </entry>
  <entry>
    <word>abduct</word>
    <definition>kidnap</definition>
  </entry>
  <entry>
    <word>capital</word>
    <definition>money</definition>
  </entry>
</dictionary>
```

Figure 3.11: An XML Document with Three Sample Instances of Glossary Entries

concept	instance	absolute path	relative path
Example 1			
Entry:		/html/body/table[5]/tr[bgcolor=”red”][3]/	
Word:	abbey	/html/body/table[5]/tr[bgcolor=”red”][3]/td[0]	td[1]
Definition:	convent	/html/body/table[5]/tr[bgcolor=”red”][3]/td[2]	td[1]
Example 2			
Entry:		/html/body/table[5]/tr[bgcolor=”red”][4]/	
Word:	abduct	/html/body/table[5]/tr[bgcolor=”red”][4]/td[0]	td[1]
Definition:	kidnap	/html/body/table[5]/tr[bgcolor=”red”][4]/td[2]	td[1]
Example 3			
Entry:		/html/body/table[5]/tr[bgcolor=”red”][7]/	
Word:	capital	/html/body/table[5]/tr[bgcolor=”red”][7]/td[0]	td[1]
Definition:	money	/html/body/table[5]/tr[bgcolor=”red”][7]/td[1]	td[2]

Figure 3.12: The XPath Expressions for the Locations of the Two Glossary Entries in the Response

concept	grammar
Entry	/html/body/table[5]/tr[bgcolor=“red”]
Word	td[1]
Definition	td[2]

Figure 3.13: The Generalized XPath Expressions for the Location of the Glossary Entries

Suppose that the resource response consists, in part, of a glossary containing a list of words and their definitions, shown in Figure 3.10. Further, suppose that the task is to find the rule for extracting all the entries in the glossary, but none of the additional information on the page. Let us examine the process required to create a grammar for extracting the word-definition pairs from this hypothetical example.

In the glossary example, the target concept is the glossary entry and its components are the word and its definition. Figure 3.12 shows three instances of the target concept that are included in the resource response. These instances are part of the input of the algorithm. Let us assume that each glossary entry in the HTML response of the resource is contained in a separate row of a table and the word and its definition are data fields in the table. Let us further suppose that each entry's background is red. In the particular response page, the entries for the words “*abbey*”, “*abduct*” and “*capital*” are located in the fourth, seventh and eighth rows of that table.

The first step of the algorithm (steps 1.a and 1.b) is to use a post-order searchtraversal of the response document tree to identify the locations of the entries on the page and to formulate the XPath rules for these locations. Figure 3.12 shows the rules for the locations of the components of the three entries in the page. The word "*abbey*" can be located by the path "html/body/table[5]/tr[3]/td[0]" in the HTML document tree, and the definition of the word, "*convent*", can be located by the path "html/body/table[5]/tr[3]/td[1]". The word "*capital*" can be located by the path "html/body/table[5]/tr[7]/td[0]", and its definition, "*money*", can be located by the path "html/body/table[5]/tr[7]/td[1]", and so on.

Based on the XPath rules of the locations of its components, the rule for the location of the target concept instance can be formulated (step 1.c). The XPath rule for the location of each entry is the minimum-spanning tree containing its word and definition, i.e., the maximum common prefix of the locations of its constituents. In this example, the location of the "*abbey*", "*abduct*" and "*capital*" entries are at "html/body/table[5]/tr[3]/", "html/body/table[5]/tr[4]/", and "html/body/table[5]/tr[7]/" correspondingly. The fourth column of Figure 3.12 shows the locations of the constituents of each instance relative to the location of the instance itself, produced by step 1.d of the algorithm.

At this point, step 1 of the algorithm shown in Figure 3.9 has been completed. The goal of the next step is to formulate, first, an abstract hypothesis for the location of all entry instances in the resource response relative to the root of the document tree, and second, a set of hypotheses for the locations of the concepts' constituents, i.e., word and definition, relative to the locations of the concept instances.

Many of the resource servers respond with HTML pages constructed by scripts collecting data from a database. These pages are highly regular in structure, and more often than not, the XPath rules of the locations of all the instances are the same with the exception of the indices for some of the path tags. Thus, the generalization step attempts to discover the regularity in the values of the differing indices. Currently, in the case where the index values are not constant, our algorithm tries three different choices of generalization, each using progressively more aggressive "guessing" (step 3). If there are only two different index values, the algorithm generalizes the value of this tag index to be either one of the given values. If there exist more than two different values, it tries to discover a linear progression over the given index values. If it fails, it goes one step further and accepts all possible values. When "guessing" is involved, the attributes are used as constraints for finding the concept instances. In another words, when the document structure does not provide enough knowledge to precisely locate the target, the visual presentation is used for further recognition.

From the XPath rules for the location of the entry instances, shown in Figure 3.12, the learning algorithm can determine that the target concept (a glossary entry) is found consistently in rows of the fifth table of the document. The generalized XPath rule for the location of all the glossary entries is "/html/body/table[5]/tr", i.e., the index of the row has been generalized to allow any value. Because this is a quite aggressive generalization step, the attributes of presentation of each entry, i.e., the red background, are kept as part of the XPath rule for its location.

The same generalization process applies to the XPath rules of the relative locations of the sub-components of the entry. In this example, the word and definition of each entry can be found in the first and the second cells of the entry row. The sub-components locations then are "td[0]" and "td[1]". The final rule for extracting the instances of the entry concept in the resource response is shown in Figure 3.13.

If the "*abduct*" instance were located in the tenth row instead, the algorithm would have chosen to generalize the expression for the entry location to "/html/body/table[5]/tr[bgcolor="red"][position() mod 2 = 3]" instead.

If the target concept is not encapsulated, step 1.c of the algorithm will return the XPath rules to the locations of multiple root nodes, instead of the root of a single tree spanning all the constituents of the target concept. The rest of the algorithm steps will be performed for each of these nodes, thus resulting in learning a set of rules for extracting instances of the constituents of the target concept instead of the target concept itself. The generic wrapper component of the architecture is capable of using these fragments of the target concept and the invariants of the target concept as defined in the domain model to combine the fragments into instances of the target concept. This way, if sufficient variants are defined for the target concept, it can be extracted even if it is not encapsulated in the response of the application to be wrapped.

Chapter 4

A Travel Assistant Prototype

Let us now consider the problem of a traveler, who wants to check airfares online by exploring different options from several different travel-planning applications on the Web. He wants to compare the possible air tickets in terms of fare, number of stops, and so on. At the same time, he does not want to commit to specific airports, rather to his origin and destination cities. As the problem illustrated in Section 1.4, the user[1] has to visit multiple Web sites and tentatively commit to origin and destination airports and repeat the queries with different airports. Then he has to save the intermediate solutions, infer additional attributes not explicitly produced by the applications and finally compare the tickets manually. In this chapter, we describe a travel assistant mediator developed in the TaMeX environment to assist him in solving his problem.

The travel-assistant prototype integrates two popular travel-planning Web applications, *www.lowestfare.com* and *www.itn.net*, and one static Web page offering worldwide airport code database, *www.airportcitycodes.com/aaa/ctydef*. The overall architecture of the travel-assistant mediator is diagrammatically shown in Figure 4.1. It consists of four layers. They are the user interface layer, the mediator server layer, the wrapper layer and the resource layer. These four layers are mapped to three physical tiers, with a thin-client interface at the top tier, a set of existing resources at the lowest tier, and the "business logic" of the aggregate application consisting of the mediator and the wrappers in the middle tier. The following sections describe the main components of the travel assistant and the run-time behavior of the application with a simple scenario.

[1] The term "user" in this chapter refers to the traveler, who is using an aggregate planning system developed by the TaMeX development environment

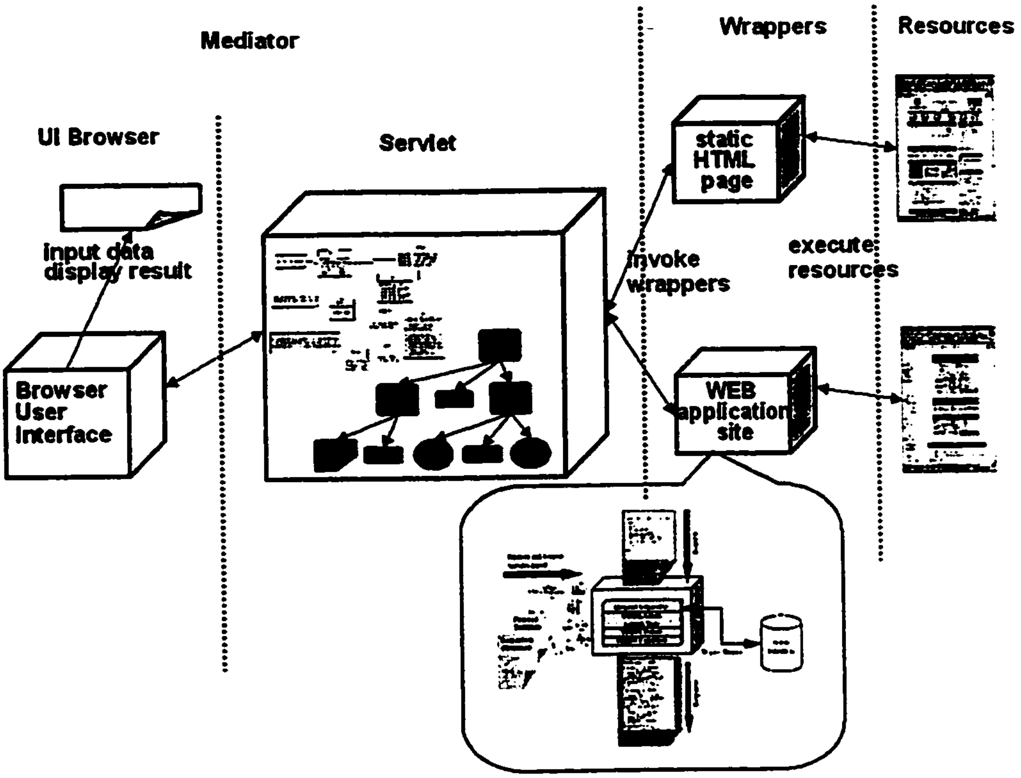


Figure 4.1: The Travel-planning Assistant

4.1 The Mediator

Figure 4.2 shows the travel domain model in extended UML[2]. The shaded rectangles represent the objects and their attributes in the model; the lines show the relationships, such as inheritance and composition, among the objects; and the folded rectangles describe the constraints on the objects.

As shown in Figure 4.2, a *two-way ticket* is composed of a *forward leg*, a *return leg* and an *airfare*. A *leg* consists of one or more *hops*, such that the origin of a hop is the destination of the previous hop, and the origin of the first hop of the forward leg is the destination of the last hop of the return leg and vice versa. Each *hop* is described in terms of the origin and destination, which are both of type of *location*, the departure and arrival time, which are both of type of *time*, and the flight.

An interesting aspect of the domain model of the travel-planning assistant is that *location* and *time* are both instances of *domainLevelObject*, i.e., the attributes of both *location* and *time* have *domain level* relationships. For example, *country*, *state*, *city* and *airport* are all attributes of *location*, but at different levels of specificity.

[2] We extend the vocabulary of UML with the notations for describing the domain constraints.

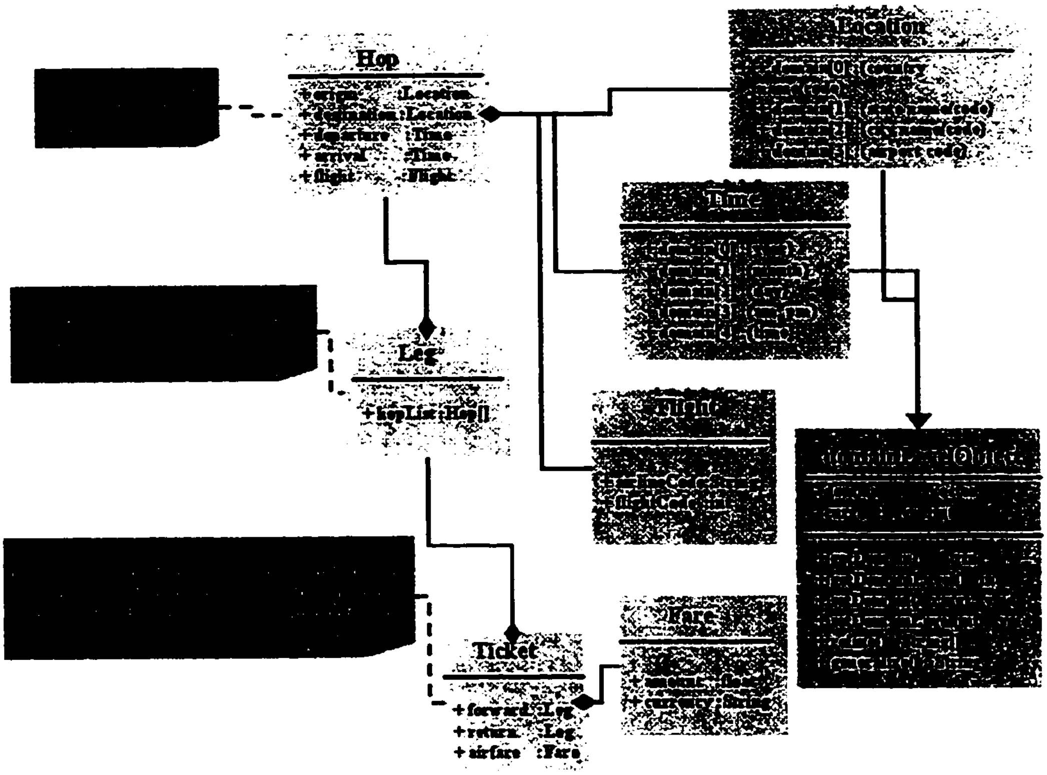


Figure 4.2: The Travel Domain Model in extended UML

state is at a lower (more specific) level than *country*, and at a higher (more general) level than *city*. The underlying intuition is that values at one level are collapsed and summarized at the next higher level, and a value at one level corresponds to a collection of values at the next lower level.

The concept of *domain level* is very important for supporting flexible, exploratory search for a desired service. It enables the interpretation of vague problem statements, i.e., statements expressed in terms of attribute values at a higher level, into a collection of specific ones, i.e., expressed in terms of their corresponding values at lower levels. In such a way, the user can avoid early commitment to undesired constraints. For example, when the user does not want to commit to a specific destination airport but only to a city, the mediator is able to translate this vague problem statement into a collection of problems, each of which has one of the airports close to the given city as the destination airport. Alternatively, when the user has made strong and mutually exclusive commitments, the mediator could explore possible relaxation by replacing some specific elements of the problem statement with more general ones at a higher level of abstraction, thereby providing the user

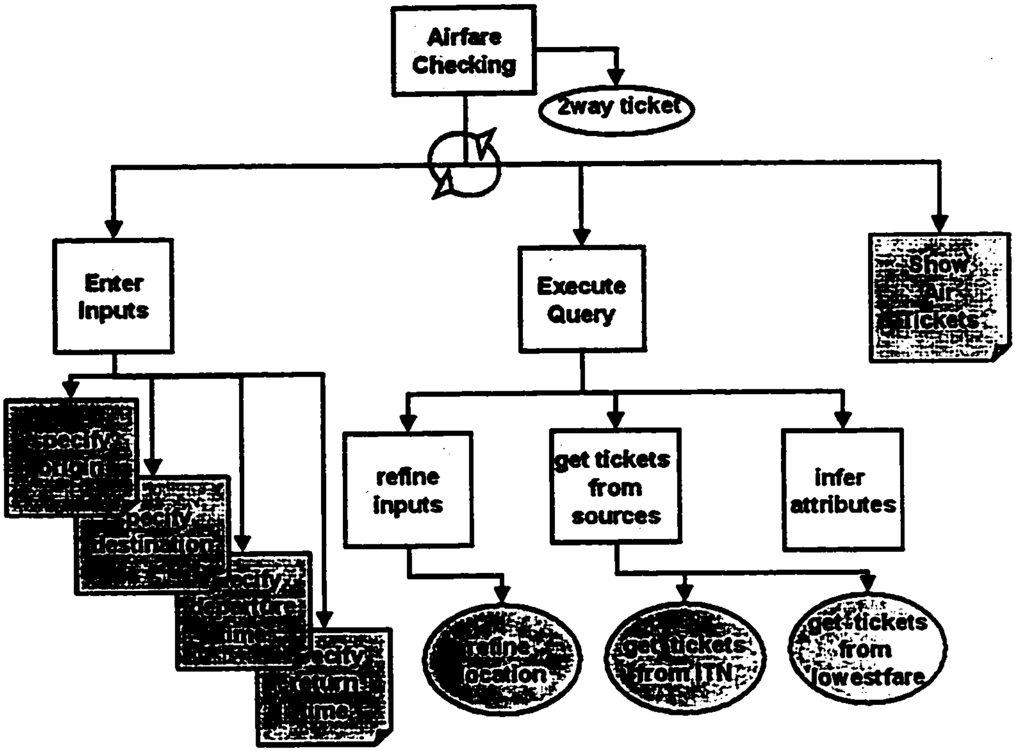


Figure 4.3: The Task Structure of the Travel-assistant Mediator

additional search space for problem solutions.

The task, for which the travel-assistant mediator is designed, is to solve ill-structured problems in the sense that a precise specification of the desired output is not available. Instead, the user explores the solution space following a least commitment strategy [35]. Consider, for example, the problem of finding air tickets for a vacation "to California in July". "July" and "California" are too vague for querying any of the current travel-agent Web applications. The specific destination and date may eventually depend on the availability of flight seats and the price of the airfare. But in order to make more precise decisions, the user may want to evaluate the possible options by comparing the tickets with different attributes, some of which are presented by the resources, and some of which are inferred by the mediator. For such problems, the specification of the problem may shift around within the domain. The above specification implies a range of destinations and a range of departure and arrival dates.

Figure 4.3 diagrammatically depicts the travel mediator's task structure. In this task structure, information-collection tasks are shown as ovals, interaction tasks are shown as rectangles with folded corners, and internal tasks are shown as rectangular boxes. The overall task of the mediator, i.e., to find airfares, involves three tasks. The first one is to ask for the user problem by allowing the user to specify the different attributes of the desired ticket. The second task is to solve the problem by refining the problem, invoking the Web resources to find the desired tickets, and inferring the additional attributes. The last task is to display the tickets to the user. These tasks are further decomposed until reaching the non-decomposable operators. Among them, there are user-interaction tasks, for which appropriate XSL stylesheets are developed, and information-collection tasks, whose corresponding wrappers are constructed with the process described in section 3.3.

4.2 The Wrappers

Every information-collection task, such as *get ticket from ITN*, is initiated by the mediator and performed by a corresponding resource wrapper, such as the wrapper for the Web resource *www.itn.net*. When invoked by the mediator, a wrapper delivers two important functionalities. First, it maps the problem specification to a set of parameters for a particular method of the application, and invokes this method. Once the application response is returned, the wrapper parses the HTML document to extract the relevant pieces of information expected as an answer by the mediator.

Each wrapper in our integration architecture contains the following information:

- the common domain model including the domain ontology and constraints,
- the URL of the wrapped application,
- the resource protocol that the wrapper invokes, including the required parameters for the method and how to map a problem specification represented in terms of the domain model to an HTTP request, and
- the grammar for information extraction, i.e., a domain-model extension that describes the XPath expressions for the location of instances of the target concept in the application's HTML response.

The canonical domain model, such as the travel domain described previously, is shared by the mediator and all the wrappers of the integrated resources. The

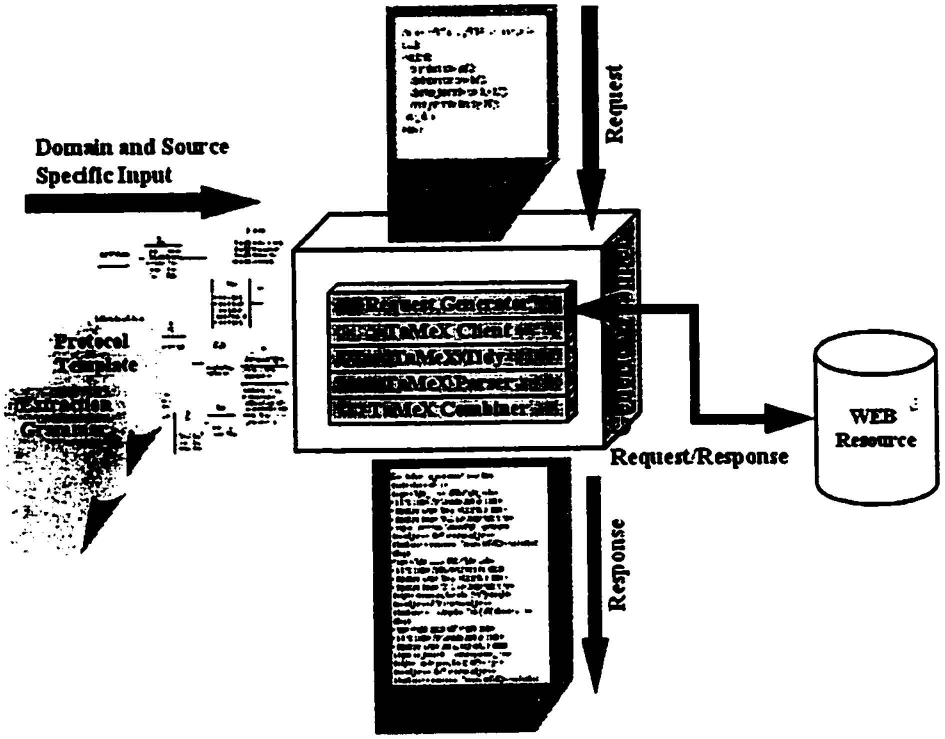


Figure 4.4: The Generic Wrapper

remaining items are specific to each wrapper. The resource protocol specifies the translation of a user problem to a request understood by the underlying resource. The extraction grammar defines a set of rules for appropriately interpreting the resource information in terms of the common domain model.

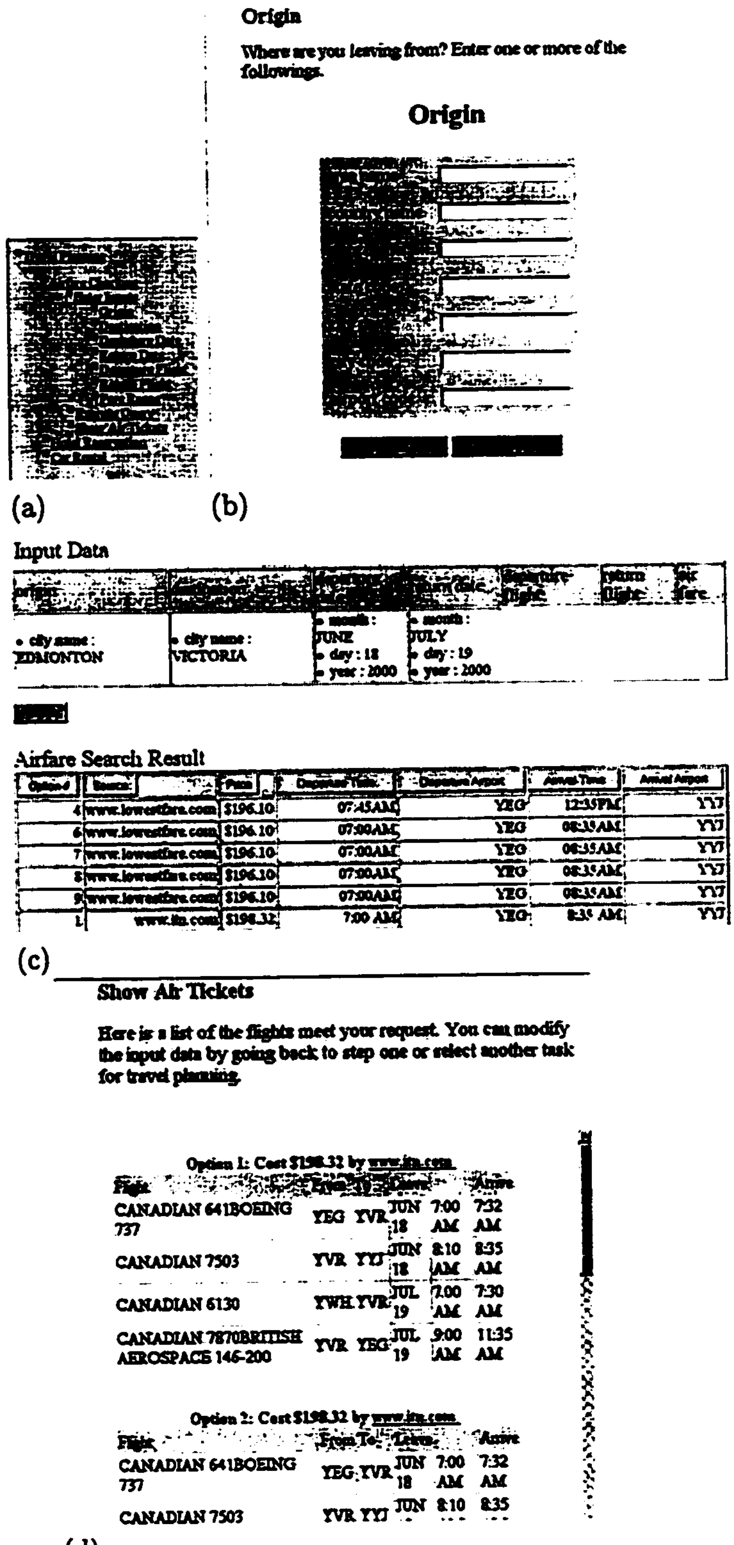
In the TaMeX environment, a resource-specific wrapper is constructed by passing the above information to a generic wrapper. The generic wrapper consists of five components as shown in Figure 4.4. They are the *Request Generator*, the *TaMeX Client*, the *TaMeX Tidy*, the *TaMeX Parser* and the *TaMeX Combiner*.

The *Request Generator* is responsible for mapping a problem specification to a resource protocol. After receiving a problem specification from the mediator, it uses the application-specific protocol template to compose the parameters of the HTTP request method with the values extracted from the problem specification. The generated resource protocol contains its URL, the HTTP request method, and a set of request parameter-value pairs. The *TaMeX Client* is a generic HTTP client built on the *HTTP Client* class library [23]. It is responsible for communicating with the application server. It invokes HTTP requests, handles cookies and receives responses. The *TaMeX Tidy* is based on the *HTMLTidy* [39]. It transforms HTML pages to well-formed documents by inserting missing closing tags, quoting HTML attribute values, and removing invalid tags. The output of the *TaMeX Tidy* is a well-formed HTML document conforming to the XML standards. The *TaMeX Parser* is a generic parser based on the SAXON library [37]. It applies the extraction rules to interpret the response document by extracting all instances of the concept constituents based on the extraction grammar provided. Finally, the *TaMeX Combiner* is a piece of software that reconstructs the instances of the concept from the extracted instances of its constituents based on the given domain constraints. Often the instances of the concept constituents are more easily located than those of the concept itself. This happens when there is a mismatch between the structures of the target concept and the HTML document, such as missing intermediate concepts or having non-encapsulated concepts. In such cases, the instances of the constituents are used by the Combiner to reconstruct the instances of the concept based on the domain ontology and its constraints. The output of the Combiner is an XML document conforming to the domain model and containing all the instances of the target concept extracted from the HTML response to the problem specification from the mediator.

4.3 The Run-time Behavior

In this section, we illustrate the run-time behavior of the travel-assistant with a simple scenario. Suppose the user wants to check the airfare from Edmonton to Victoria departing on June 18th, 2000 and returning on July 19, 2000. Let us walk through the process of accomplishing this task to show how the task structure drives the behavior of the travel-assistant mediator, which integrates *www.itn.net*, *www.lowestfare.com*, and *www.airportcitycodes.com/aaa/ctydef* as indicated previously.

The mediator's task structure is specified in XML. By applying a generic XSL Stylesheet specifically for presenting the hierarchical structure of the task, the menu shown in Figure 4.5(a) is produced on the user's XML-enabled browser. The user of the travel-assistant application interacts with the mediator through this menu by browsing and selecting the task he wants to perform. He selects *Airfare Checking* by clicking on the link. The menu expands and displays three subtasks, *Enter Inputs*, *Execute Query* and *Show Air Tickets*, which lead the user to find solutions to his



Origin

Where are you leaving from? Enter one or more of the followings.

Origin

Input Data

origin	[illegible]	[illegible]	return date	departure flight	return flight	air fare
• city name : EDMONTON	• city name : VICTORIA	• month : JUNE • day : 18 • year : 2000	• month : JULY • day : 19 • year : 2000			

Airfare Search Result

Option #	Source	Price	Departure Time	Departure Airport	Arrival Time	Arrival Airport
4	www.lowestfare.com	$196.10	07:45AM	YEG	12:35PM	YYJ
6	www.lowestfare.com	$196.10	07:00AM	YEG	08:35AM	YYJ
7	www.lowestfare.com	$196.10	07:00AM	YEG	08:35AM	YYJ
8	www.lowestfare.com	$196.10	07:00AM	YEG	08:35AM	YYJ
9	www.lowestfare.com	$196.10	07:00AM	YEG	08:35AM	YYJ
1	www.itn.com	$198.32	7:00 AM	YEG	8:35 AM	YYJ

Show Air Tickets

Here is a list of the flights meet your request. You can modify the input data by going back to step one or select another task for travel planning.

Option 1: Cost $198.32 by www.itn.com

Flight	From	To	Leave		Arrive
CANADIAN 641BOEING 737	YEG	YVR	JUN 18	7:00 AM	7:32 AM
CANADIAN 7503	YVR	YYJ	JUN 18	8:10 AM	8:35 AM
CANADIAN 6130	YWH	YVR	JUL 19	7:00 AM	7:30 AM
CANADIAN 7870BRITISH AEROSPACE 146-200	YVR	YEG	JUL 19	9:00 AM	11:35 AM

Option 2: Cost $198.32 by www.itn.com

Flight	From	To	Leave		Arrive
CANADIAN 641BOEING 737	YEG	YVR	JUN 18	7:00 AM	7:32 AM
CANADIAN 7503	YVR	YYJ	JUN	8:10	8:35

Figure 4.5: Snapshots of the Travel-assistant User Interface

problem.

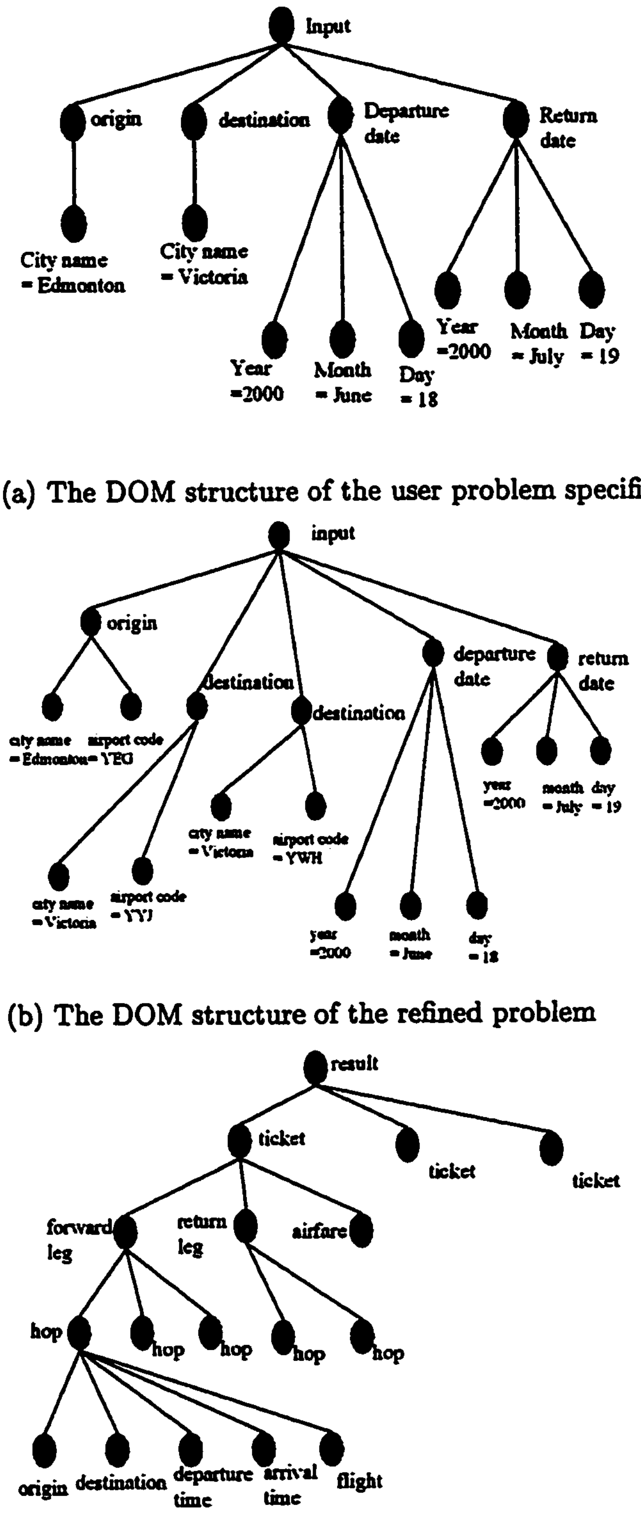
1. **Specifying the Problem**

   The first step in the process is to specify the problem inputs, i.e., the attributes of the air ticket the user is interested in. This first subtask (*Enter Inputs* in Figure 4.3) is decomposed in a set of user-interaction tasks, each of which corresponds to the specification of one of the ticket attributes, including the origin and destination locations, and the departure and return dates. When the user selects one of these tasks, an HTML request form is displayed asking the user for the input. Each form corresponds to this concept in the domain model, and is generated by the XSL Stylesheet specifying the presentation of the form. Figure 4.5(b) shows the interface generated by applying to the domain a Stylesheet developed especially for the concept *origin*. The Stylesheet for *origin* is designed to accommodate the different levels at which a location can be specified, such as country, state, city, and airport code. The user specifies the origin of the ticket, Edmonton, at the *city name* field, and submits the form by clicking *Enter the Data* button. The user's input is sent to the mediator server as the parameter of the $HTTP :: POST(param_i*)$ request. Based on the caller name of the HTTP request, the mediator identifies the corresponding task, which is *enter origin*. It updates the input DOM, which is initially empty, with the data submitted as the request parameter. Similarly, the user submits the information about the destination, the departure date and the return date. And the mediator updates the input DOM as shown in Figure 4.6(a).

2. **Querying Web Resources**

   After the desired ticket attributes have been specified, the user activates the next task, *Execute Query*, by sending another HTTP request to the mediator. The mediator identifies the task and proceeds to accomplish the task, which is in turn decomposed into a sequence of subtasks.

   The first subtask is to refine the attributes specified at high levels of abstraction to their most specific domain levels (task *refine inputs* in Figure 4.3). This is an internal task; that is, it is further decomposed in a set



(a) The DOM structure of the user problem specification

(b) The DOM structure of the refined problem

(c) The DOM structure of the wrapper response

Figure 4.6: The Mediator's DOMs at Different Stages

of tasks[3], each of which is tailored to refining some type of information with a domain-level attribute, e.g., locations. The location-refinement task is an information-collection task, that is, a task accomplished by invoking a request to some of the wrapped resources. When an information-collection task is reached, the mediator sends a request to the wrappers of the resources producing the desired output given a particular set of inputs. For the location-refinement task, the mediator queries the wrapper of the airport database at *www.airportcitycodes.com/aaa/ctydef*, which is static. The wrapper finds all the airports in Edmonton, which is YEG, and in Victoria, which are YYJ and YWH. The mediator receives the airports and updates the DOM. As a result, the origin and destination are defined at the lowest, most precise domain level for locations. The refined input DOM is as seen in Figure 4.6(b).

Next, the mediator proceeds to the task of finding possible tickets for the refined problem (task *get tickets* in Figure 4.3). This is yet another internal task, and decomposed to two information-collection subtasks. One is *get tickets from ITN*, and the other is *get tickets from lowestfare*. The task, *get tickets from ITN*, is performed by the wrapper for the Web site, *www.itn.net*. When the ITN wrapper receives the problem, i.e., the input DOM, from the mediator, it generates two problems with the combination of different origin and destination airports. Then, for each problem, it does the following. First, the *Request Generator* maps the problem specification to the appropriate GET request based on the protocol template of *www.itn.net*. The request includes all the required values of the parameters, such as the origin, destination, departure and return date, and is then taken by the *TaMeX Client* to invoke the ITN Web site. After the HTML response is returned from the site, it is cleaned by the *HTML Tidy* to produce a well-formed HTML document. Next, since ITN response documents do not contain concepts for *forward leg* and *return leg* (see 5.1 for a detailed discussion), the *TaMeX Parser* extracts the instances of all the constituents, namely *hop* and *fare*, according to the extraction rules specified in the domain-model extension. Then the *TaMeX Combiner* reconstructs a set of instances of *forward leg*, *return leg* and *air ticket* from the extracted instances of *hop* to conform to the domain model schema. Finally, it returns

[3] Although there is currently only one element in this set, the idea is that additional refinement, such as time refinement, can be added.

all the instances of *air ticket* in the form of a DOM as shown in Figure 4.6(c) to the mediator. Similarly, the mediator receives the other collection of tickets as the response from the wrapper of *www.lowestfare.com*, and combines to produce a DOM of all the tickets.

The last subtask of *execute query* is *infer attributes*. This is an internal task, and thus is performed by one of the internal processes of the mediator itself. The mediator derives additional attributes, such as the number of hops in either leg, for each ticket in the collection.

3. **Displaying Results**

The ticket results are displayed by two different XSL stylesheets.

As the response to the user's request for the task of *execute query*, the summary of the air ticket collection is displayed in the browser through a stylesheet as a table capable of being sorted by the different ticket attributes. This functionality enables the user to compare the tickets based on his own preferences, such as price, departure/arrival airports, and so on. Figure 4.5(c) shows the mediator's response including the problem specification on the top and a sortable table of the result summary.

Finally, the user selects *Display Tickets* to view the detail of each ticket. The mediator proceeds to another user-interaction task. The same collection of air tickets is presented by being applied by another stylesheet to show the full ticket details as in Figure 4.5(d).

Chapter 5

Conclusions

Incompatible interaction models and differences in terminology prevent users from combining the services of different Web sites. These incompatibilities are, in some cases, intentional and designed to prevent service and price comparisons for business competition reasons. However, even in cases where the different Web sites offer complementary services, such as air-travel reservations and weather forecasting, the integration of information is often difficult and tedious. In general, the effort does not transfer well to additional sites of the same domain. Nevertheless, users of the World Wide Web would be provided with more and better services if some interoperability were possible.

In this thesis, we designed and developed a task-structure based mediation environment for establishing interoperation of Web-based applications that is extensible and can be, at least partially, automated. The objective of the TaMeX environment is to support task-specific interoperation of a collection of existing applications, as opposed to transforming these applications into generally interoperable components. Applications are wrapped to interoperate for supporting a particular user task. As a result, the domain model of the mediator is designed to express the types of information that are necessary for the accomplishment of the task, and not necessarily all types of information that the underlying applications can process. Furthermore, the application wrappers make available to the mediator not all the applications' services, only the ones relevant to the mediator's task. Finally, the mediator's task structure controls the locus of information exchange to deliver the solution to the user's task.

5.1 Evaluation and Reflection

5.1.1 The Mediation Architecture

In TaMeX, the mediation architecture is task-structure based. The main properties of this design are that the implementation is explicitly separate from the specification, and the aggregate system has a highly flexible integration architecture.

Separation of Specification from Implementation

In TaMeX, the task structure of the aggregate system is not only a specification of the behavior designed in the mediator, but also a roadmap for the developer designing and integrating different components of the aggregate system. High-level tasks correspond to a menu-like interface allowing the user to select the services provided by the mediator; information-tasks correspond to a set of wrappers through which the mediator interacts with the underlying resources; user-interaction tasks correspond to user interfaces through which the mediator exchanges information with the user; and internal tasks correspond to additional methods or procedures for further domain-specific reasoning. However, the implementation is absolutely hidden from this specification. It depends on the user's browser for the interaction tasks, on the wrapped applications for the information-collection tasks, and on the mediator's implementation for the internal tasks. The separation of the system implementation from the specification provides an abstract view of the aggregate system and allows the flexibility of implementing each component of the system independently.

Declarative modeling languages

Another two important elements of the mediator's design in TaMeX are the domain model and task structure, both expressed in XML. The declarative task-structure language allows the lightweight and seamless integration of new resources. The canonical domain model language, supported by XML Schema, provides a uniform and semantic data schema for the domain-specific conversations among heterogeneous applications. This is a knowledge engineering effort, which could benefit from research on ontology engineering [1]. In fact, various business and technology sectors have embarked on the specification of standard XML schemas for exchanging information in their domains of interest. As such standard domain descriptions become more common, the design of TaMeX-style mediators, and correspondingly, the integration of existing Web applications, will become more straightforward.

5.1.2 The Wrapper Construction Algorithm

The wrapper construction algorithm developed in TaMeX is based on the hierarchical structure of HTML documents. The representation languages for documents and grammar are general. HTML documents and the extracted information are both represented in Document Object Model (DOM). The extraction grammar is specified in XPath expressions. In fact, we have constructed wrappers by hand that conform to the same representation and behave similarly at run-time.

The grammar-learning algorithm has been evaluated using both generated test data and actual HTML pages from travel planning Web applications. The generated test data consisted of encapsulated and contiguous target concepts in a variety of HTML structures including lists, tables, and paragraphs. The algorithm was able to successfully learn the grammar for all six sets of test data when given a small set of exclusive annotated examples for each set. The learned grammar was then used to extract all instances of the encapsulated concept from the generated pages.

The algorithm was also used to wrap two popular travel planning Web sites: *www.expedia.com* and *www.itn.net*. The target concept was encapsulated in the first and contiguous but not encapsulated in the second. In the effort of wrapping these sites, we encountered some problems that show the limitations of the current algorithm in facing the following situations.

Missing intermediate concepts

Although all elementary leaf entries of the target concept exist in the responses from ITN and Expedia, some intermediate constituents are not encapsulated. For example, in Expedia, the ticket concept is encapsulated, but the forward and reverse legs are not. This means that the forward and reverse legs are not delimited by HTML tags in the application responses. In fact, the ticket simply consists of a set of hops, some of which constitute the forward leg and some the reverse. In this case, the algorithm fails to identify the XPath expressions for the forward and reverse legs. In fact, this problem is due to a mismatch between the structure of the concept in the domain model and the underlying database schema of the Web application: the first assumes a more complex structure than the second does. At this point, to address this problem, we present the algorithm with the examples of the target concept assuming different variants of the canonical domain model produced by removing the intermediate subconcepts. As soon as the examples of one variant are identified, the grammar for this variant is produced. During run-time, the instances conforming to the variants will be extracted and used to compose the instances of the concept conforming to the domain model.

Non-encapsulated and non-contiguous concepts

A further complication occurs when the ticket concept itself is not encapsulated in the application's response, as is the case of ITN. In this case, the algorithm fails to identify the examples of all concept variants. Currently, the algorithm hypothesizes that the concept is not encapsulated. It then attempts to identify instances of its constituents starting from the more complex toward the simpler ones. Again, once a grammar is formulated, it is used to produce a wrapper for this application that will construct instances of the target concept, as defined in the canonical model, based on the constraints of this concept. The wrapper construction algorithm is still limited in that it does not address the problem of learning non-contiguous concepts. The algorithm needs to be extended in this dimension and we believe that the constraints of the domain model will be extremely useful.

"Unified" constituents

The algorithm also faces problems if the HTML response of the application has more than one constituent of the target concept in the same HTML node, such as the *origin* and *destination* presented within the same <TD> tag and separated by a <br/> tag. An extension is needed to further enrich the language expressions for the extraction grammars. The learned XPath rules can be extended with expressions of semantic delimiters in addition to HTML tags.

"Hidden" constituents

The current algorithm is only able to "extract what can be seen", i.e., the information which is the value of an element node in the HTML document. Therefore, the information that is presented as an attribute value, such as a link presented as the value of <A href>, will be overlooked. To solve this problem, the rule expressions need to be further extended to include HTML attributes.

The algorithm is effective but still quite brittle and limited. As the complexity of the concepts to be extracted from the application response increase and the structure of the response HTML document becomes flatter, the algorithm becomes less effective. This is not surprising, since it corresponds to learning more from less. To our knowledge, until now, related research has focused on simpler concepts.

Finally, another challenging problem we have encountered in the process of integrating Web applications is the fact that Web-based applications frequently change their request protocol and the presentation of their responses, and consequently change the structure of the HTML documents. Currently, we recognize that the underlying wrapped application has been changed when the resource wrappers fail at run-time. A more robust technique is needed to recognize at run-time when the wrapper needs to be updated and to proceed to the learning process automatically.

5.2 Future Work

In the future, we plan to extend TaMeX mainly in two dimensions. One is to improve the complexity of the wrapper construction algorithm. The other is to improve the intelligence of the mediator. More specifically, we will focus on the following problems.

Extending the wrapper learning algorithm As we discussed in Section 5.1.2, the current wrapper learning algorithm is useful but still fairly naive. When the concept structure gets more complex, and more mismatches occur between the structure of the target concept and the schema of the underlying resource, the algorithm becomes less effective. We believe that the variants of the concept and domain constraints can be useful to solve the problem. However, a more intelligent algorithm is needed to diagnose the cause of failures, and to automatically take the correct action, such as generating the concept variants. In addition, to be able to extract more information from HTML documents, the grammar expressions need to be further enriched so that semantic delimiters and HTML attribute values are included in addition to HTML tags.

Wrapping Web resources that require multi-step interactions The current wrapper is only able to simulate single-step user actions to interact with the Web resource. However, for many existing Web sites, the user has to follow a sequence of steps in order to obtain the desired information. In such cases, the wrapper has to act on behalf of the user to have conversations with the Web resource. The difficulty is that the sequence of interactions is non-deterministic and based on the various user inputs. One possible solution is to have the learner record all the possible interactions while an expert user interacts with the Web resource. The learner would then generate a finite-state automaton modeling the resource. Each node would represent every different screen and each edge would represent the possible user action and possible user inputs. Such an automaton would be used as a "map" to guide the wrapper in completing the sequence of interactions with the Web resource.

Supporting dynamic binding The main strategy of the task-based mediator is its ability to interpret its non-deterministic task structure at run-time to dynamically combine the wrapped applications. In the current version of the prototype, the interpretation process is hard-coded. To achieve dynamic binding, during run-time, the user could activate the higher level tasks which would be sent as part of a request to the mediator. The mediator could then decompose the task based on the task structure to the point where a set of non-decomposable operators would be identified. For an operator corresponding to a task of collecting information from a Web resource, the mediator would invoke the corresponding wrapper. This dynamic run-time binding could result in highly flexible system integration.

Supporting wrapper registration One of the requirements for an aggregate system is its extendibility. Therefore, it becomes very important for such a system to be able to "employ" a new resource with the least effort from the developer and with a minimum effect on the existing components of the system. This could be done by having a wrapper registry table, which is a list of all the wrappers of the underlying resources. The table would be represented in XML and would specify all the resource-specific information, such as the location of each underlying resource, the function or services it provides, its protocol to request the services, its grammars to extract desired information, etc. The table would be expanded over time as new resources are joined in. During run-time, the mediator will look up the table for the wrappers that provide appropriate services to complete the specific tasks.

Problem Generalization One of the features of a TaMeX mediator is its ability to solve ill-defined problems. The mediator is able to refine a problem by either generalizing or specifying some of its attributes based on the domain level concept. In the current version of the TaMeX mediator, only specialization is implemented. However, generalization is extremely useful when the user's query does not give any solutions. It could provide the user with suggestions or alternatives to his original problem.

Solving the naming problem A critical problem for integration of heterogeneous applications is the naming problem. Different resources have their own naming schemas. This can cause difficulties not only during the learning phase, such as while the learner tries to match the user problem to the resource protocol, but also during run-time while the wrapper attempts to translate the extracted information to a common language. In the former case, the learner needs to have the full knowledge of all the naming schemas in order to map the sample problem specifications to the learning samples of the source protocol and extraction grammar. A solution can be, for each type of object involved in the domain, to designate a recognizer that is responsible for recognizing a specific type of object in different naming schema. In the later case, the wrapper of each individual resource has to perform the translation, so every wrapper "speaks" in the same language. A possible solution is, for each wrapper, to have a mapping table that contains the naming schema for each object in the domain. During the run-time, the wrapper looks up the table for the correct translation from the source-specific naming schema to the one agreed to by all the wrappers and the mediator. Both the mapping tables and the recognizers should be extendable as the number of integrated resources increases.

Supporting multiple users The current TaMeX mediator only deals with single users. It can be extended to handle multiple users by using HTTP cookies and a persistent XML database. Each user would be sent a cookie ID the first time using the system. Every query and its response would be stored in the database under each user based on the cookie ID. The database would also store each user's preferences.

5.3 Contributions

The central contribution of this thesis is the development a task-structure based mediation architecture for the integration of domain-specific Web applications through XML. More specifically, this thesis has made the following contributions.

- **Knowledge Representation in XML**

  TaMeX provides the specification of two languages in XML in support of the design of an aggregate system:

  - the wrapper-specification language, and
  - the mediator task-structure specification language and the associated stylesheet for its presentation to the user.

  The wrapper-specification language enables the description of the protocol for accessing the underlying Web resources and the grammar for extracting the data provided by the resources. The modeling language for the mediator task structure provides a clear view of the data flow of the application and easy identification of its sub-components. Also, because the task-structure XSL stylesheet is generic, i.e., conforming to the data schema of the task model, it not only makes the interface design simpler, but also ensures the interfaces of all TaMeX-developed applications give a consistent look and "speak" with the same terminology. These features make the aggregate applications easy to learn and to use.

- **Grammar-Learning for Complex Concepts with Multiple Instances**

  The wrapper-learning algorithm of TaMeX is also novel. It takes advantage of the hierarchical structure of HTML documents to discover the rules for extracting the desired data. It observes the regularities within the tree structures of HTML documents, and generates a grammar for extracting multiple instances of a domain concept interspersed among irrelevant HTML content without requiring any special landmarks or delimiters. Furthermore, with the support of the domain constraints, the algorithm is able to generate the most complex class of wrappers identified by Kushmerick [33, 34].

- **Generic, Reusable System Implementation**

Although the TaMeX environment was not developed as an application framework [36, 20], its design and, to some extent, the implementation of some of its components, are generic and reusable across similar applications.

The development of a mediator is based on its task structure. The task structure describes the internal processing logic of the mediator and the information flow within the aggregate application. The high level tasks correspond to a user's high level options of the tasks he can perform. The leaf tasks, either user-interaction tasks or information-collection tasks, implement the mediator's information exchange with either the user or the wrappers of the integrated resources. The internal tasks reflect the internal data processing within the mediator. The task structure enables the lightweight and seamless integration of newly developed components with the existing functionalities.

Furthermore, the generic-wrapper component, to which the wrapper protocol and response grammar are plugged, is generic. It consists of the *Request Generator* for mapping application problems to resource protocols, the *TaMeX Client* for sending requests and retrieving responses, the *TaMeX Tidy* for cleaning HTML response documents, the *TaMeX Parser* for extracting desired data in HTML documents, and finally the *TaMeX Combiner* for constructing instances of target concepts from their constituents. These components complete the entire Web resource wrapping process, and are reusable for any Web resource in any domain. The construction of a source specific wrapper requires simply specifying the domain model and learning the source description, in the XML resource-description language, by way of the wrapper construction toolkit.

- **The Travel-Planning Assistant: A Proof-of-Concept Prototype**

  The TaMeX development environment has been evaluated with the development of a proof-of-concept prototype in the travel domain. The travel-assistant mediator is connected to three wrappers for three different Web resources, which are

  - a static online airport database, *www.airportcitycodes.com/aaa/*, the wrapper for which was constructed manually, and
  - two airfare-finding services of two Web applications,

www.lowestfare.com and *www.itn.net*.

The user of the travel-assistant prototype can specify a travel problem in terms of origin and destination locations and departure and arrival times. The mediator can elaborate the problem specification to refine the locations into several airports close to the specified origin and destination and then, it proceeds to query the airfare-finding applications for specific ticket information. The results are collected, tabulated and displayed to the user, who can compare them by sorting on different attributes, such as price, departure airport, etc.

This thesis has presented TaMeX, a task-structure based mediation architecture for structurally developing new applications by integrating multiple Web resources in a specific domain. An aggregate system developed in the TaMeX environment enables the communication among heterogeneous Web applications in a common XML-defined language and the collaboration among them towards the completion of complex tasks. TaMeX also provides a semi-automatic toolkit for developing wrappers for Web applications based on the hierarchical structure of HTML documents.

Bibliography

[1] A.Farquhar, R.Fikes, W.Pratt, and J.Rice: Collaborative Ontology Construction for Information Integration. Knowledge Systems Laboratory Department of Computer Science, KSL-95-63, August 1995.

[2] A.O.Mendelzon, G,A.Mihaila, and T.Milo: Querying the World Wide Web. In the proceedings of the International Conference on Parallel and Distributed Information Systems (PDIS'96), pp.80-91, Miami, Florida, 1996.

[3] A.Sahuguet, and F.Azavant. Looking at the Web through XML glasses. CoopIs'99, 1999.

[4] A.Sahuguet and F.Azavant: Building Intelligent Web Applications Using Lightweight Wrappers. Data and Knowledge Engineering (to appear), 2000.

[5] A.Tomasic, L.Raschid, and P.Valduriez. Scaling Heterogeneous databases and the Design of DISCO. In the proceedings of ICDCS, 1996.

[6] B.Chandrasekaran: Task Structures, Knowledge Acquisition and Machine Learning. Machine Learning 4:341-347, 1989.

[7] B.Chidlovskii, J.Ragetli and M.Rijke: Wrapper Generation via Grammar Induction. In the proceedings of 11th European Conference in Machine Learning (ECML'2000), Barcelona, Catalonia, Spain, 30 May - 2 June 2000.

[8] B.Ludascher, and A.Gupta: Modeling Interactive Web Sources for Information Mediation, Intl. Workshop on the World-Wide Web and Conceptual Modeling (WWWCM'99), Paris, France, LNCS, Springer, 1999.

[9] B.Ludascher, Y.Papakonstantinou, and P.Velikhov: A Framework for Navigation-Driven Lazy Mediators, ACM Workshop on the Web and Databases (WebDB'99), Philadelphia, USA, 1999.

[10] C.A.Knoblock, S.Minton, J.L.Ambite, N.Ashish, P.J.Modi, I.Muslea, A.G.Philpot, and S.Tejada. Modeling Web Sources for Information Integration, Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98), Madison, WI, USA, 1998.

[11] C.Bontempo. Datajoiner for AIX. IBM Corporation, 1995.

[12] C.Hsu and M.Dung: Generating Finite-state Transducers for Semi-structured Data Extraction from the Web. Information System, Vol. 23, No. 8, pp. 521-538, 1998.

[13] D.Konopnicki and O.Shmueli: WWW Information Gathering: The W3QL Query Language and the W3QS System. In ACM Transactions on Database Systems, September 1998.

[14] Document Object Model (DOM) Level 2 Specification. $http : //www.w3.org/TR/1999/CR - DOM - Level - 2 - 19991210/$

[15] E.Stroulia and A.K.Goel: A Model-Based Approach to Blame Assignment: Revising the Reasoning Steps of Problem Solvers. In the Proceedings of the 13th Annual Conference on Artificial Intelligence, pp. 959-965, AAAI Press, 1996.

[16] E.Stroulia and A.K.Goel: Redesigning a Problem Solver's Operators to Improve Solution Quality. In the Proceedings of the 15th International Joint Conference on Artificial Intelligence, pp. 562-567, 1997.

[17] E.Stroulia, J.Thomson, Q.Situ: Constructing XML-speaking wrappers for Web Applications: Towards an Interoperating Web. In the Proceedings of the 7th Working Conference on Reverse Engineering (WCRE'2000) 23-25 November 2000, Brisbane, Queensland, Australia, IEEE Computer Society.

[18] Extensible Markup Language, *http : //www.w3c.org/XML*

[19] Extensible Stylesheet Language (XSL) Version 1.0 W3C Working Draft 27 March 2000. *http : //www.w3.org/TR/xsl*

[20] G.Froehlich, H.J.Hoover, L.Liu and P.Sorenson: Hooking into Object-Oriented Application Frameworks. In the Proceedings of the 1997 International Conference on Software Engineering. Boston, May 1997.

[21] G.Wiederhold and M.Genesereth: The Conceptual Basis for Mediation Services, IEEE Expert, 1997, pp. 38-47.

[22] H.Marais and T.Rodeheffer: Automating the Web with WebL. Dr.Dobb's Journal, January, 1999.

[23] HTTPClient Library. *http : //www.innovation.ch/java/HTTPClient/*

[24] I.Muslea, S.Minton and C.Knoblock: A Hierarchical Approach to Wrapper Induction. 3rd Conference on Autonomous Agents 1999.

[25] I.Muslea, S.Minton and C.Knoblock: Hierarchical Wrapper Induction for Semistructured Information Sources. (To appear) In the Journal of Autonomous Agents and Multi-Agent Systems (special issue on "Best of Agents'99").

[26] J.L.Ambite, C.A.Knoblock, I.Muslea, and A.Philpot: Compiling Source Descriptions for Efficient and Flexible Information Integration, Technical Report, Information Sciences Institute, University of Southern California, 1998.

[27] J.Lu, J.Mylopoulos, and J.Ho: Towards Extensible Information Brokers Based on XML, 12th Conference on Advanced Information Systems Engineering (CAiSE*00), 7-9 June 2000, Stockholm, Sweden.

[28] K.Syscara, J.Lu, M.Klusch, and S.Widoff: Matchmaking among Heterogeneous Agents on the Internet, Proceedings of the 1999 AAAI Symposium on the Intelligent Agents in Cyberspace, 22-24 March, 1999, Stanford University, USA.

[29] L.Liu, C.Pu and W.Han: XWRAP: An XML-enabled Wrapper Construction System for Web Information Sources.

[30] L.M.Haas, R.J.Miller, B.Niswonger, M.Tork Roth, P.M.Schwarz, and E.L.Wimmers. Transforming Heterogeneous data with Datacase Middleware: Beyond Integration. IEEE Data Engineering Bulletin, vol.22, no.1, pp.31-36, 1999.

[31] M.Craven, D.DiPasquo, D.Freitag, A.McCallum, T.Mitchell, K.Nigam and S.Slattery: Learning to Extract Symbolic Knowledge from the World Wide Web. Proceedings of the 15th National Conference on Artificial Intelligence (AAAI'98), pp. 509-516, Madison, WI, USA, 1998.

[32] Muffin, World Wide Web filtering system. *http : //muffin.doit.org/*

[33] N.Kushmerick, D.Weld and R.Doorenbos: Wrapper Induction for Information Extraction. In the proceeding of 15th International Joint Conference on Artificial Intelligence (IJCAI'97), Nagoya, Aichi, Japan, 23-29 August, 1997.

[34] N.Kushmerick: Wrapper induction: Efficiency and expressiveness. Artificial Intelligence 118, pp. 15-68, 2000.

[35] R.Elio, and A.Haddadi: On Abstract Task Models and Conversation Policies. Specifying and Implementing Conversation Policies for Agents, 3rd International Conference on Autonomous Agents, pp. 89-98. Seattle, WA, USA, May 1999.

[36] R.Johnson, and B.Foote: Designing Reusable Classes. Journal of Object-Oriented Programming, June/July 1988, Volume 1, Number 2, pages 22-35.

[37] SAXON. *http : //users.iclway.co.uk/mhkay/saxon/*

[38] S.Heiler: Semantic Interoperability. ACM Computing Surveys. 27(2):271-273, June 1995.

[39] The HTML Tidy utility. *http : //www.w3.org/People/Raggett/tidy/*

[40] V. S. Lakshmanan, F. Sadri, and I. N. Subramanian: A Declarative Language for Querying and Restructuring the Web. In the proceedings of 6th International Workshop on Research Issues in Data Engineering, RIDE'96, pp 12-19, February 1996.

[41] XML Path Language (XPath) Version 1.0, W3C Recommendation 16 November 1999. *http : //www.w3.org/TR/xpath*

[42] XML Schema, *http : //www.w3.org/TR/xmlschema − 1/*, *http : //www.w3.org/TR/xmlschema − 2/*.

[43] Y.Arens, C.Y.Chee, C.Hsu, and A.Knoblock: Retrieving and Integrating Data from Multiple Information Sources. International Journal of Intelligent and Cooperative Information Systems. Vol. 2, No. 2, pp. 127-159, 1993.

[44] Y.Papakonstantinou, H.Garcia-Molina, and J.Widom: Object Exchange Across Heterogeneous Information Sources. In the proceedings of IEEE Conference on Data Engineering. pp. 251-260, Taipei, Taiwan, 1995.

Appendix A

The Languages

A.1 Domain Model Language

Domain model language is supported by XML Schema specified at
http : //www.w3.org/TR/xmlschema – 1/,
http : //www.w3.org/TR/xmlschema – 2/.

A.2 Task Structure DTD

```
<!ELEMENT taskModel (task+)>
<!ELEMENT task ( input*, output*, (subtask+) | operator )>
<!ELEMENT subtask EMPTY>
<!ELEMENT operator (#PCDATA)>
<!ELEMENT input (#PCDATA)>
<!ELEMENT output (#PCDATA)>
<!ATTLIST  taskModel name CDATA #REQUIRED>
<!ATTLIST  task
           name CDATA #REQUIRED
           id ID #REQUIRED
           type CDATA #REQUIRED
           description CDATA #REQUIRED
>
<!ATTLIST  subtask
           name CDATA #REQUIRED
           id IDREF #REQUIRED
>
<!ATTLIST  operator
           xmlfilename CDATA #IMPLIED
           xslfilename CDATA #IMPLIED
           servlet CDATA #IMPLIED
>
<!ATTLIST  input required (yes | no) #REQUIRED>
```

A.3 Resource Protocol Specification DTD

```
<!ELEMENT request method (cookie+) form)>
```

```
<!ELEMENT form (parameter+)>
<!ATTLIST  method type (GET | POST) #REQUIRED
           url CDATA #REQUIRES >
<!ATTLIST  cookie name CDATA #REQUIRED
           value CDATA #REQUIRES >
<!ATTLIST  parameter
           name CDATA #REQUIRED
           value ID #REQUIRED
           input (YES | NO) "NO"
>
```