Model-Driven Design and Deployment of Service-Enabled Web Applications

Ioana Manolescu[1]
INRIA Futurs – LRI, France

Marco Brambilla, Stefano Ceri, Sara Comai and Piero Fraternali
Politecnico di Milano, Italy

Significant effort is currently invested in application integration, enabling business processes of different companies to interact and form complex multi-party processes. Web service standards, based on WSDL, have been adopted as process-to-process communication paradigms. However, the conceptual modeling of applications using Web services has not yet been addressed. Interaction with Web services is often specified at the level of the source code; thus, Web service interfaces are buried within a programmatic specification.

In this paper, we argue that Web services should be considered as first-class citizens in the specification of Web applications. Thus, service-enabled Web applications should benefit from the high-level modeling and automatic code generation techniques that have long been advocated for Web application design and implementation. To this end, we extend a declarative model for specifying data-intensive Web applications in two directions: (*i*) high-level modeling of Web services and their interactions with the Web applications which use them (*ii*) modeling and specification of Web applications implementing new, complex Web services.

Our approach is fully implemented within a CASE tool allowing the high- level modeling and automatic deployment of service-enabled Web applications.

Categories and Subject Descriptors: D.2.2 [**Design Tools and Techniques**]: Computer-aided software engineering (CASE); D.2.6 [**Programming Environments**]: Graphical environments; D.2.1 [**Requirements/Specifications**]: Methodologies; D.2.10 [**Design**]: Methodologies, representation

General Terms: Algorithms, Design, Human Factors, Languages

Additional Key Words and Phrases: Web application, Web services, modeling, WebML, UML

[1]Part of this work was performed while Ioana Manolescu was visiting Politecnico di Milano. This work was partially funded by the EU Fifth Framework *WebSI* project, and by the *MIUR MAIS* project. Stefano Ceri is partially supported by *CNR-IEIIT*.

Ioana Manolescu's address: INRIA Futurs, Gemo group, 4, rue Jacques Monod, 91893 Orsay Cedex, France.
Marco Brambilla, Stefano Ceri, Sara Comai, and Piero Fraternali's address: Dipartimento di Elettronica e Informazione, Politecnico di Milano, Via Ponzio 34/5, 20133 Milano, Italy.

1. INTRODUCTION

Web Services have emerged as essential ingredients of modern Web applications; they are used in a variety of important contexts, including Web portals for collecting information from worldwide providers or business-to-business applications for the integration of enterprise procedures. The success of Web services stems from two simple facts: they rely on simple and widespread standards, and are pushed by the leading companies in the IT industry. While these two factors are sufficient to guarantee the "technological" success of Web services, they are still insufficient to guarantee the seamless integration of Web services into the best software development practices, as there exist very few formal models, methods, and tools specifically designed for Web services. As a consequence, Web service specification requires the mastering of verbose XML interfaces, and the integration of Web services within applications occurs at a very low programming level.

In this paper, we present a high-level language and methodology for designing and deploying Web applications using Web services. We use a hypertext model for describing Web interactions, and define specific concepts in the model (with an associated iconic representation) to represent Web service calls. Thus, Web service invocation is captured by a visual representation language, capturing the relationship between the invocations and the data units which provide their inputs, respectively, capture their outputs.

These specifications extend WebML [Ceri et al. 2000; Ceri et al. 2002], a high-level model of Web applications, and make Web services first-class citizens of the model. The WebML approach for designing Web applications is validated by the automatic deployment of such applications into low-level code, running on arbitrary platforms [Ceri et al. 2003]. However, the extensions we present can be adapted to other hypertext models recently proposed in the literature [Atzeni et al. 1998; Fernandez et al. 1998; Florescu et al. 1999] etc.

The benefits of automatic generation of WebML-designed applications carry over to the Web service extensions presented in this paper. Indeed, service-enabled Web applications can be automatically derived from WebML diagrams, and run on any platform providing the communication support required for Web service interactions.

In this paper, we discuss how WebML is extended to support Web services. We adopt the WSDL standard for describing message exchanges, and show the new units required to support WSDL within WebML. We also introduce some new WebML operation units, required to convert and materialise XML content, and describe how to simplify the required translation efforts by introducing a standard XML representation for local and global data resources. The proposed specification language supports the exchange of messages with Web services in both a synchronous and asynchronous manner, considered from the perspective of the end-user: the former is currently the most used, but the latter is the most promising in terms of future development of service-enabled Web applications.

The specification language also supports "duality", i.e., the ability to represent not only the application calls to Web services, but also the deployment of applicative functions in the form of Web services. With the growth of asynchronous communications, we expect that service-enabled applications will progressively adopt

peer-to-peer models, since they will at the same time invoke services and be exposed as services.

Finally, we describe how WebRatio, a commercial CASE tool supporting the WebML design process, is extended to specify and automatically implement service-enabled Web applications running in top of a standard SOAP-based communication layer. The WebRatio design environment is equipped with a code generator that produces the code of the specified application and of the Web services in the J2EE or Microsoft .NET platforms, including data extraction queries, Web service calls, data marshalling and un-marshalling logics, page templates, and WSDL service descriptors.

We stress that our approach focuses on, and fits best in Web service composition scenarios that involve human interaction on the Web. More specifically, we provide a declarative framework for specifying hypertext-based platforms where humans may provide input, make decisions, interact with existing business processes, and in short, enact a well-defined role in an inter-organizational business process.

This paper is organized as follows. Section 2 discusses related work. Section 3 introduces minimal WebML background required to make the paper self-contained. Section 4 progressively presents the WebML units specifically designed to model Web services; Section 5 describes how Web service calls can be composed into conversations. Section 6 describes how it is possible to publish a portion of Web application as a Web service, to be used by a remote peer. Then, Section 7 describes the implementation of Web services in WebRatio, the commercial tool that supports WebML, and reports our experiences with the proposed approach. Section 8 compares the Web service primitives offered by WebML with the popular Web Service standard BPEL4WS [Andrews et al. 2002]. Finally, Section 9 outlines ongoing and future work.

2. RELATED WORK

In this section, we present a broad review of the research work relevant to Web service composition and orchestration, and to the model-driven design of Web applications; these works form the background of the proposal presented in this paper.

2.1 Declarative specification of data-intensive Web sites

Several platforms and languages similar to WebML have been developed for describing data-intensive Web applications (see [Atzeni et al. 1998; Fernandez et al. 1998; Florescu et al. 1999], etc.; for a survey, see [Fraternali 1999]). Among more recent projects, Weave [Yagoub et al. 2000] focuses on Web site performance, and Qursed [Papakonstantinou et al. 2002] is an all-XML project for visualizing and querying XML data. Neither of these proposals consider interaction with remote processes. The ActiveXML system [Abiteboul et al. 2003] manages XML applications and document composition including calls to services, but does not consider Web interfaces and user interaction. The well-known BEA Workshop [BEA 2004] tool allows developers to specify Web applications and Web services, but with a limited high-level view of the application, because it concentrates on the implementation tasks. From a methodological perspective, the WebML approach is close to conceptual methodologies like W2000 [Baresi et al. 2000] and OO-HMETHOD [Gomez et al. 2001] (based on UML interaction diagrams.)

2.2 Basic Web service standard: WSDL

The basic language for describing Web services, WSDL, is currently being standardized by the W3C [WSDL 2002]. A WSDL document contains a *service type description*, as well as a set of services conforming to this description. A service type consists of a set of port types, each of which contains several operation types. An *operation* is the basic unit of interaction with a service, and is performed by exchanging messages; for any message input to or output by the service, the WSDL service type definition provides a type description in XML Schema [XSchema 2001]. A mapping from the above abstract model to any particular messaging protocol (e.g., SOAP [SOAP 2001]) is called a *binding*. A port type, a binding, and a network address, together, specify a port. Finally, a *service*, in WSDL sense, is a set of ports. We further detail the features of WSDL operations in Section 3, where we introduce the WebML extensions for modeling them.

2.3 Web service conversations and coordination

The WSDL standard describes Web service properties and syntax. However, the full potential of Web service-based development resides in the possibility of combining several services into more complex, meaningful conversation. To that purpose, several proposals have been made recently; they can be seen as being at the top of the so-called "Web service standard stack". Among the most recent Web service coordination and orchestration proposals, we mention BPEL4WS [Andrews et al. 2002], WSCI [WSCI 2002], and WSCL [WSCL 2002]. Several XML-based languages for encoding workflows have been proposed (see [Christophides et al. 2001] for a discussion of some of them). Languages like BPEL4WS [Andrews et al. 2002] specify workflows exclusively consisting of Web services; BPEL4WS includes an XML Schema for encoding complex workflow- style interactions between several Web services, based on such composition primitives as sequence, test, split, join etc. Such XML process descriptions are consumed by a workflow management system (e.g., IBM MQSeries), responsible for implementing the workflow. XL [Florescu et al. 2002] is an XML-based programming language that allows both defining and combining services. A recent discussion of BPEL Services interaction is given in [Fu et al. 2004]. Among other Web service composition proposals, [Casati and Shan 2001] describes E–Flow, developed at HP.

Languages like BPEL, WSCI, XL etc. allow the high-level specification of a complex process consisting of interactions between individual Web services; they are neither concerned with the modeling of hypertext-based implementation of Web services, nor with the automatic deployment of such implementations. Furthermore, they are centred on a process model, not on an application data model, as is the case in WebML. In our work, data remains central, following the original WebML philosophy. We show that, in addition to data and hypertext modeling, WebML can capture interesting interaction scenarios involving human users, Web applications, and Web services.

Section 8 will provide a thorough comparison of the expressive power of WebML and BPEL4WS, following the "Workflow patterns" paradigm of [van der Aalst et al. 2002]. The comparison is deferred after the complete discussion of the modeling primitives proposed in our approach, for the sake of readability.

2.4 Semantics of Web services and conversations

The goal of Web service models is to establish common frameworks for describing service semantics, together with several formal properties like service guarantees, availability etc. Such a framework would ease the dynamic identification of possible partners, the choice of the most suitable service, ad hoc process establishment etc.

An example of a formalisation of Web services is offered by Lenzerini et al. [Berardi et al. 2003]: this work describes a framework allowing to define a finite set of elementary actions, and a set of Web services whose implementations rely only on these actions. Web services are defined in terms of (infinite) execution trees of elementary actions, and a restricted subset thereof can be expressed in terms of finite-state machines. In such cases, complex services can be automatically composed by combining the elementary actions of individual Web services. Under the above-mentioned restrictions, this approach is a promissing attempt at achieving automatic composition. Hull, Benedikt, Christophides, and Su provide in [Hull et al. 2003] a broader view on the Web services scenario, by using contributions from the theory of computation to asses the Web service properties that can be automatically inferred.

Conversations must obey coordination protocols, i.e. the sequence of operation calls involving a set of Web services must be a legal one; several works provide formalisms and verification procedures for ensuring that a specific conversation complies with a given coordination protocol [Abiteboul et al. 1998; Bultan et al. 2003]. A recent example of this is the work by Deutsch, Sui, and Vianu [Deutsch et al. 2004], where the authors propose to verify Web applications, designed using a subset of WebML, with respect to relevant properties specified by means of linear and branching-time temporal logic. The application's constraints are written using temporal logic, and the WebML specification is checked to see if it satisfies the constraint formulas. These works can be considered as a first step towards the verification of WebML applications interacting with Web services.

3. BACKGROUND ON WEBML

WebML [Ceri et al. 2002] is a conceptual model for specifying a data-intensive Web application, i.e. a Web site whose primary goal is publishing or manipulating content stored in one or more data sources. A WebML conceptual specification consists of a data schema and one or more hypertext schemas, expressing the Web interface used to publish and manipulate the application data. In this section, we introduce a running example that will be used throughout the paper, and summarise the main WebML concepts using this example.

Writing conventions. Throughout this paper, we make the following writing conventions. The names of various WebML hypertext primitives, consisting of one or several space-separated words, are shown in italic fonts, e.g. *Home* page. Names of Web services are similarly written. Names of individual Web service operations are shown in regular fonts, and always consist of just one name token, e.g. NewsSubscription.

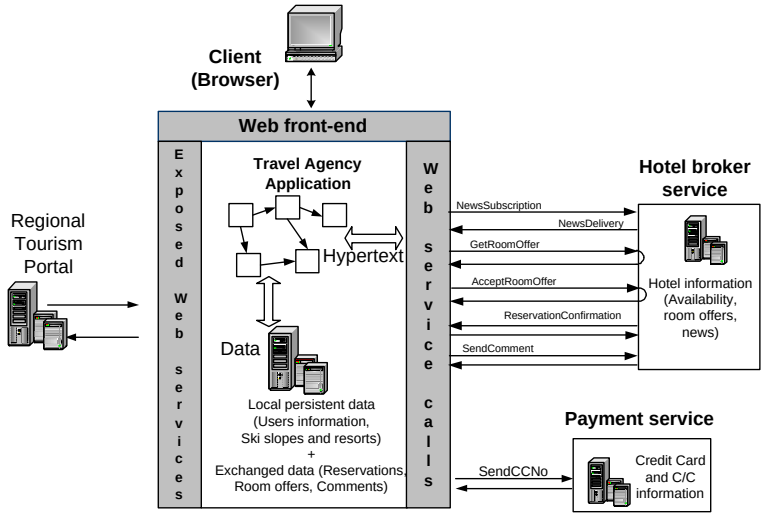

Fig. 1. The actors involved in the Travel Agency running example.

3.1 Running example

The running example considers an online Travel Agency,[2] illustrated in Figure 1.

Users can access the Web front-end of the Travel Agency to look for available hotel rooms meeting their search criteria, make reservations, pay for booked services, and exchange comments with specific hotel managers, e.g. users may inquire for special accommodation conditions, extra services, etc. The Travel Agency Web application exploits a *local database* storing the customer details and some tourism information (e.g., data on the ski slopes in mountain resorts). Moreover, the Travel Agency Web application takes advantage of two external *Web services*: the *Hotel Broker* Web service, which delivers room availability data, publishes a news bulletin, accepts room reservations, and answers users' comments; and an independent *Online Payment* Web service, which collects room fees.

The Travel Agency not only consumes Web services, but also exposes part of its functionality as a Web service. In particular, it offers a broadcasting service whereby remote peers (for instance, a *Regional Tourism Portal* peer, shown in Figure 1) can subscribe for and then receive hotel information. Thus, the Travel Agency uses external services, and in turn provides an HTML-based interface for human users, and a WSDL-based interface for peers, who can further use and compose the exposed services.

3.2 Data model

The WebML data model is the standard Entity-Relationship (E-R) model, widely used in general-purpose design tools. Entities are connected by named relationships, and may be organized into generalisation hierarchies, connected by *isA* re-

[2]The running example is inspired by application scenarios discussed with Tiscover, an Austrian Tourism Service Broker partner of the WebSI project.

lationships. Named relationships are annotated with the minimum and maximum cardinalities of participation of each entity, and may have named relationship *roles*, denoting a specific direction in which the relationship is considered. Each entity instance is uniquely characterised by an implicit *object identifier* (OID), and relationships between entities and generalisation hierarchies are mappings between OIDs.[3]

Figure 2 shows the E-R schema describing the local database of the Travel Agency. The Resort, Hotel, and Slope entities model the central concepts of the Travel Agency database. The Hotel_Resort relationship[4] specifies that in one resort there are zero or more hotels, and each hotel is in one resort. The Resort_Slope relationship specifies that in one resort there are one or more ski slopes, and that each slope belongs to one resort. The schema also describes the users of the Web applications, who are specialized in Tourists and application Managers. The RoomOffer entity, connected to the Hotel entity by relationship Hotel_RoomOffer, encodes the conditions under which the hotel rents its rooms. The Reservation entity, associated to the Hotel and Tourist entities, denotes a reservation made by a specific user for a specific room offer. The Subscription entity and its relationship with the Tourist entity denote that users have subscribed to the news channel about a selected topic; the NewsUpdate entity models the news issued in connection with a specific subscription. Finally, the Comment entity, connected to the Tourist entity, represents comments that the tourists may exchange with the hotel in which they rent rooms.

3.3 Hypertext model

The WebML hypertext model is used to specify the application's front-end. Each application consists of one or more *site views*; each site view is a specific hypertext designed to meet the requirements of a certain class of users (e.g., end-user, content managers, etc.), or of a certain access device (e.g., users connected with a PDA, a smart phone, etc). Site views consist of *pages*, where each page can be seen as a container of several smaller data fragments. Within the site view, pages are connected by *links*, which express navigation and parameter passing. One page of each site view is denoted *home page*, and is served by default to the user, in response to her initial connection to the Web application. The content of pages is specified in terms of content *units*, which are the elementary components for information publishing. WebML provides a set of predefined content units, some of which are illustrated in the sequel, and offers to the designers an extension mechanism for wrapping their own data publishing components as WebML content units.

As an initial example of hypertext modeling, consider the following specification of a Web interaction:

[3]E-R schemas are mapped to data sources using standard database techniques. Such mappings can be done for *pre-existing databases*, in which entities and relationships are associated with existing schema elements. Alternatively, the E-R schema can be used to *generate* the relational schema of a new database hosting the application content. Instances of the schema data can be inserted in this database by loading it from other data sources, or via content management applications. See [Ceri et al. 2002] for details.

[4]In the sequel, we will use the naming convention that assigns to the relationship between an entity A and an entity B the name "A_B".

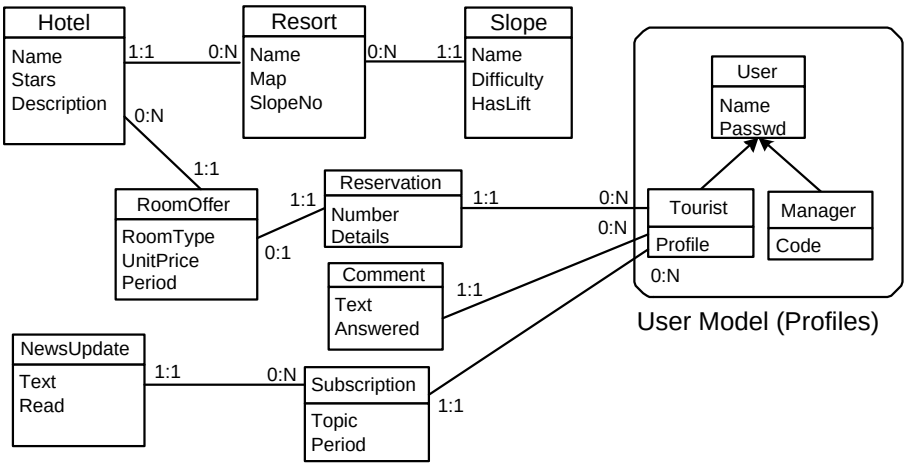

Fig. 2. The data model for the Travel Agency Web application.

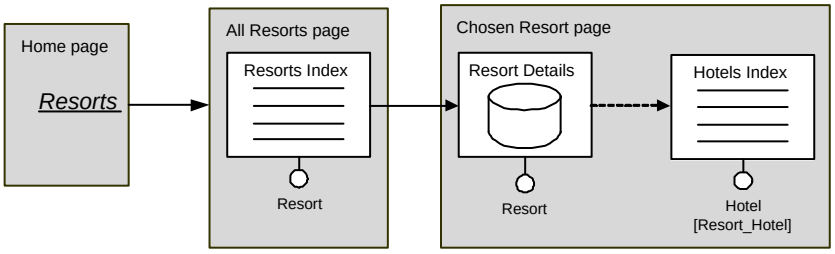

Fig. 3. Specification of a simple hypertext for browsing resort information.

> "Tourists enter the Travel Agency Web application using its *Home* page, from where they can navigate to a page showing an index of all the available resorts; after choosing one resort, tourists are led to a new page with the resort details and the list of hotels at the resort."

The hypertext model illustrated in Figure 3 fulfils the interaction described above. The hypertext includes three pages; the *Home* page contains static, un-modelled content; a link, labeled *Resorts*, from the *Home* page leads to the *All Resorts* page, containing an index of all resorts; and the selection of a resort from the index leads to the *Chosen Resort* page, showing the resort details and the index of hotels located at the selected resort. We now discuss the main elements of the hypertext specification in more detail.

Content units. Content units are components for publishing data into pages in various ways, and for accepting user input, e.g., by means of entry forms. WebML offers seven predefined content units [Ceri et al. 2002], but for simplicity in this paper we will use only four of them:

(1) *Data units* publish the data of a single entity instance.

(2) *Index* and *multidata units* publish the data pertaining to a set of entity instances: the former permits the user to select an individual object among those published, whereas a multidata unit provides in output the whole set of al the displayed objects.

(3) An *entry unit* allows users to feed data values to the Web applications. Entry units will typically be rendered as Web forms.

The hypertext of Figure 3 shows the use of data and index units. Entry units are illustrated in the hypertext of Figure 4, and multidata units are illustrated in the example of Figure 20.

In Figure 3, the *All Resorts* page contains an index unit named *Resort Index*, which is defined on top of the Resort entity, as denoted by the label at the bottom of the unit's icon. This index unit in effect publishes the list of all the resorts, each resort instance being visually identified by a subset of the its attributes (e.g., the resort name).

The *Chosen Resort* page contains two units: a data unit named *Resort Details* displays the attributes of a single resort (e.g. a photo, some text, and a map); an index unit (*Hotel Index*) displays a list of selected hotels, specifically those located at the resort displayed in the data unit. As this example demonstrates, the content of a unit may depend on *parameters* associated with another unit; such parameter passing is enabled by links and selectors, discussed next. The meaning of the page is that when the user navigates from the *All Resorts* page to the *Chosen Resort* page by selecting an item in the index of resorts, the *Chosen Resort* page opens and shows the details of the selected resorts and, at the same time, the index of all the hotels in that resort.

Links. The connections between pages and units, together forming a hypertext structure, are captured by links. WebML links can be classified as:

—*Contextual links* carry context information, in the form of one or more parameters, from their source unit to their target unit.

—*Non-contextual links* connect whole pages, and correspond to plain HTML anchors without parameters.

For example, the link from the *Home* page to page All resorts is non-contextual, because it connects two pages. On the contrary, the link from the *Resorts Index* unit to the *Resort Details* unit in Figure 3 is contextual, because it transports the OID of the resort chosen in the index unit, which determines the Resort instance displayed in the data unit. Note that the parameters carried by a contextual link are not always shown in a WebML diagram, because in most cases they can be inferred from the specification: for example, a link going out from an index unit always carries the identifier of the chosen object, a link going out from a data unit always carries the identifier of the object displayed by the unit etc. [Ceri et al. 2002]. Thus, the links exiting from the *Resorts Index* and *Resort Details* units in Figure 3 implicitly carry as parameter the OID of a Resort instance.

Both the non-contextual link from the *Home* page to page *All Resorts* and the contextual link from the *Resorts Index* unit to the *Resort Details* data unit enable the user to navigate from one page to another one, and therefore are rendered as anchors. Interestingly, WebML includes also non-navigable links, called *transport*

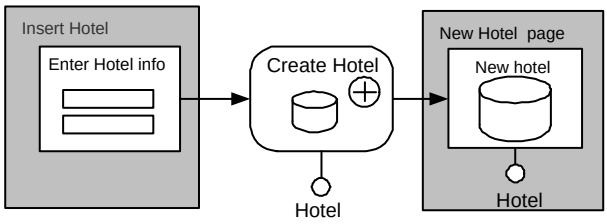

Fig. 4. WebML specification for inserting a new hotel instance.

links, drawn in the hypertext diagram as dashed arrows, which are not rendered as anchors. For example, the link from the *Resort Details* data unit to the *Hotels Index* unit is specified as a transport link, denoting that when the user enters the *Chosen Resort* page, the *Resort Details* unit is displayed, and then the content of the *Hotels Index* unit is computed and displayed at the same time (without requiring the user's intervention). In this case, an anchor is not required, thus the link is a transport link.

Selectors. The content of a unit may depend on selectors, which are (possibly parametric) predicates, filtering the instances of the entity on which the unit is defined: only the instances obtained by this filtering will contribute to the unit's contents. For example, in Figure 3, the resort OID transported from the *Resort Details* unit to the *Hotels Index* unit is used to select the hotels associated with the resort by the relationship Resort_Hotel. This filtering is expressed by the selector [Resort_Hotel][5], built over the relationship Resort_Hotel: this selector ensures that only the Hotel instances connected to the chosen resort are displayed in the index. In general, selectors can be assembled using complex conjunctive logical conditions [Ceri et al. 2002].

Operation units. WebML includes another category of components, called *operation units*, denoting business actions callable by the user during the navigation of the hypertext (such as updating data, checking the user's credentials, sending e-mail, evaluating a logical condition etc).

WebML predefines a few operation units for updating E-R data: the creation, modification and deletion of instances of an entity, respectively called *create*, *modify*, and *delete* operation, and the creation and deletion of instances of a relationship, respectively called *connect* and *disconnect* operation. Given that operation units do not display data but only execute a business action, they are not included in pages. However, operation units can be linked, both to content units embedded inside pages, or with other operation units, forming complex chains of business actions. An example of an operation unit is illustrated in Figure 4, where a simple hypertext allows the user to create instances of an entity. The *Insert Hotel* page contains an entry unit (*Enter Hotel Info*), denoting a form for collecting attributes of a new hotel from the user. The *Enter Hotel Info* is connected to a create operation unit,

[5]The selector notation [Resort_Hotel] is a shorthand for the complete notation [Resort_Hotel(OID)], which highlights the presence of a parameter in the condition, namely the OID of the resort implicitly associated with the input link of the Hotels index unit.

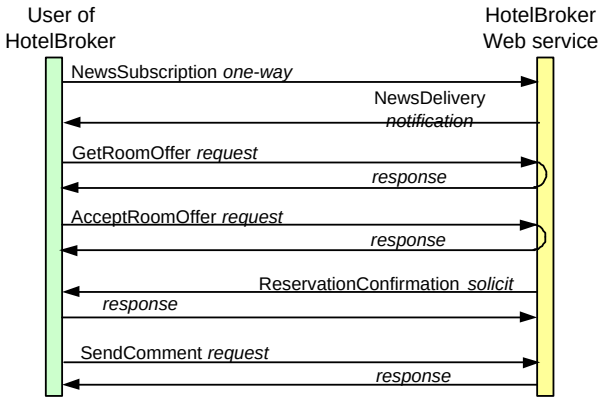

Fig. 5. Message flows between a remote peer and the *Hotel Broker* Web service.

called *Create Hotel*, depicted as a rounded-corners box. When the user submits the data by navigating the outgoing link of the entry unit, the data is passed to the create unit, which inserts a new hotel instance in the database. Subsequently, the user is led to the *New Hotel* page, where the details of the newly created hotel are displayed.

WebML includes several other content and operation units, and an extension mechanism through which application builders can design customized content and operation units [Ceri et al. 2000; Ceri et al. 2002].

4. WEBML UNITS FOR MODELING WEB SERVICES

This section presents the main ingredients for modeling Web services in WebML. We start by recalling some basic aspects of WSDL, providing the foundation of our proposed WebML extensions. A WSDL operation is the basic unit of interaction with a service, and is performed by exchanging messages; messages sent by the client to the service are called *input messages*, and messages sent by the service *output messages*.

Two categories of operations involve one message:

—A *one-way* operation is initiated by the client of the service and consists of an input message.

—A *notification* is initiated by the service and consists of an output message sent to the client.

Two other operation categories involve a message exchange:

—A *request-response* operation is initiated by the client and has one input message, followed by one output message.

—A *solicit-response* operation is initiated by the service and has one output message directed to a client, followed by one input message returned from the client.

In the remainder of this section, we introduce WebML units for one-way, notification, request-response, and solicit-response operations, and show their usage

within the WebML hypertext specification. All the icons obey two simple graphical conventions: (*i*) two-messages operations (request-response, solicit-response) are associated with round-trip arrows; (*ii*) arrows from left to right correspond to input messages as specified in the WSDL definition of the service[6]. Throughout the section, the icons associated to Web service invocation units will be graphically annotated with the name of the involved Web service operation only, for readability. This notation can be easily expanded by prefixing the operation name with the name of the Web service it belongs to.

Example. Figure 5 illustrates some WSDL messages of the*Hotel Broker* Web service. The server hosting the hotel broker presents a single Web service, called*Hotel Broker* Web service, containing a single port with the following operations:

—NewsSubscription is a one-way operation whereby a remote peer can communicate the details of a subscription to a news delivery service.

—NewsDelivery is a notification operation whereby the*Hotel Broker* Web service sends to the subscriber the updates relative to his subscription.

—GetRoomOffer is a request-response operation invoked to obtain a list of available rooms meeting search criteria provided in input to the service. The service responds with the list of rooms meeting the given selection criteria.

—AcceptRoomOffer is a request-response operation invoked to reserve a room at the conditions of a specific offer. The service responds with the details and number of the reservation.

—ReservationConfirmation is a solicit-response operation through which the *Hotel Broker* may contact a remote peer (e.g., the Travel Agency Web application) to solicit the confirmation of a pending reservation, created by means of a previous call to the *Accept Room Offer* operation.

—SendComment is a request-response operation whereby a tourist can send a comment to the *Hotel Broker*, e.g., about a hotel, and receive a response at some later time.

A user can be involved in many conversations (e.g. for reserving rooms or subscribing to news) at the same time; a conversation can be associated to several users. More details on conversation management are provided in Section 5.

4.1 One-way operation

The simplest WebML primitive modeling interaction with a Web service is the one-way operation, an example of which is visible in Figure 6 (left). The icon of the operation unit is labeled with the Web service operation name and includes a simple, right-pointing arrow that represents the message sent by the application to the Web service. The icon is further annotated at the bottom with a symbolic conversation name, which is a visual clue for the hypertext designer, showing the conversation the operation belongs to. For the sake of progressive explanation, we do not address conversations in Section 4, but defer their treatment to Section 5, after introducing all the Web service primitives.

[6]We stress that the terms "input" and "output" are interpreted from the point of view of the service, i.e. the program that implements the Web service operations, not of the Web application.

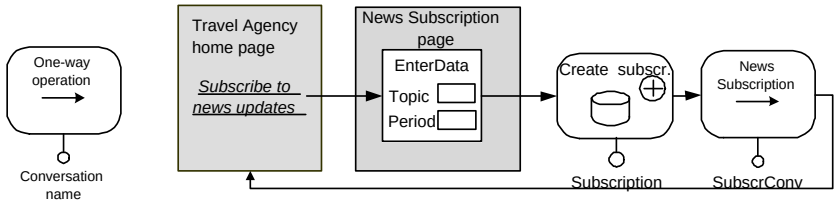

Fig. 6. One-way operation (left) and sample usage in a WebML hypertext (right).

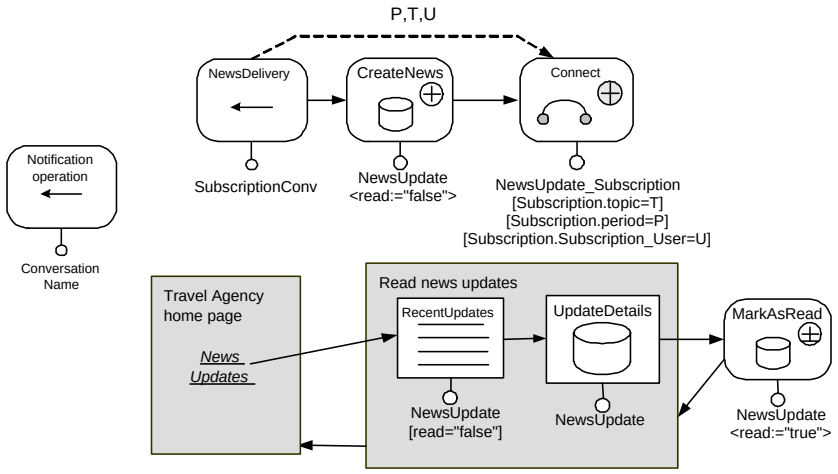

Fig. 7. Notification operation (left) and sample usage in a WebML hypertext (right).

In Figure 6 (right), we show a sample hypertext specification including a one-way operation. From the *Home* page of the Travel Agency application, users can navigate through a non-contextual link to the *News Subscription* page, where they can subscribe to the news delivery service offered by the *Hotel Broker* server. To create such a subscription, the user must enter a topic and the period of interest in the fields of the *Enter Data* entry unit. The navigation on the output link of the entry unit triggers a chain of two operations: first, the new subscription is recorded in the local database of the Travel Agency application, and second, the NewSubscription operation of the*Hotel Broker* Web service (see Figure 5) is called, to actually register the subscription at the service provider. After the one-way message is sent, the user is returned to the Travel Agency *Home* page.

4.2 Notification operation

The WebML unit for notification is shown in Figure 7 (left). Similarly to one-way operations, the icon includes a single arrow, which represents the message being sent; the arrow goes in the opposite direction than the one featured in one-way operations, because in the case of notification, it is the Web service that sends the message, and the Web application receives it.

The right part of Figure 7 illustrates the usage of the notification operation, exploiting the NewsDelivery operation of the*Hotel Broker* Web service (see Figure 5). The hypertext diagram of Figure 7 is divided into two parts.

At the top of Figure 7, we have depicted the specification of the actions taken by the Travel Agency Web application, upon the arrival of a news update from the *Hotel Broker* Web service. The diagram contains a notification operation *NewsDelivery* representing the reception of a message containing a news update relative to a subscription. The notification operation is linked to a chain of two operations. First, a create unit inserts the piece of news into the local database of the Travel Agency Web application, setting the corresponding read flag to "false". Second, a connect operation associates the News Update instance with the subscription it refers to, identified by a parametric selector condition on the Subscription entity with three parameters (period, topic, and user OID) extracted from the XML message of the notification operation and associated with the transport link to the connect operation.

At the bottom, a hypertext fragment lets the users of the Travel Agency Web application inspect the news updates relative to their subscriptions. This hypertext is necessary because the arrival of a news update, expressed by the notification operation, is totally independent of the browsing activity of the user, who may be offline when the update arrives. However, at any time the user can navigate from the Travel Agency *Home* page to the *Read News Updates* page, in which an index unit (*RecentUpdates*) displays all updates that the user has not read so far. The selection of the proper pieces of news is specified by the selector condition:

[read="false"] AND [NewsUpdate_Subscription.Subscription_Tourist(CurrentUser)]

which selects the news update instances that have the "read" flag equal to "false" and can be reached from the CurrentUser object by traversing the relationship NewsUpdate_Subscription and Subscription_Tourist[7]. Finally, in the *Read News Updates* page the user can choose a specific news update from the index, read it (thanks to the *Update Details* data unit), and subsequently mark it as "read", by navigating the link from the *Update Details* data unit to the *Mark As Read* modify operation.

4.3 Synchronous vs. asynchronous message exchange

The remaining WebML units involve two messages and correspond to the request-response and solicit-response operations of WSDL. In this section, we explain a subtle modeling issue, which leads to the introduction of two WebML units for each of the request-response and solicit-response operations.

From a Web application perspective, the interaction with a two-message Web service operation can take place in two different ways:

—*Synchronous:* no action performed by the user can take place between the two messages exchanged in the Web service operation. Automatic actions may or

[7]For simplicity, we have omitted from Figure 7 the specification of the login of the user, which has the effect of storing the user's OID in the predefined session parameter CurrentUser, exploited in the selector condition of the *Recent Updates* index unit. The interested reader is referred to [Ceri et al. 2002] for details on the login operation and on session parameters storing the user's OID.

may not be also taken by the Web application.

—*Asynchronous:* some action taken by the user may occur between the two messages exchanged in the Web service operation. This does not rule out automatic actions performed by the Web application.

This distinction between synchronous and asynchronous execution is relevant to the Web application designer, who chooses the most appropriate design pattern based on the expected interaction between the human users and the Web application. Such choice is normally independent of the technical features of the transport protocol used for the Web service messages, and is based on the application requirements; however, some extra effort may be needed for implementing asynchronous communication patterns on top of a synchronous protocol (e.g., if HTTP is used as a transport protocol, the asynchronous exchange of messages may require the use of ad hoc reply-to mechanisms).

To see the issues involved in making this decision, consider a request-response operation, the most frequently used two-way interaction between an application and a Web service. When making his choice, the designer relies on his knowledge about the expected time between the request and its response. If the time is short, e.g. one or two seconds, the application is built assuming that users will stay online waiting for the response; once the response is received, users will browse additional information about the response that will determine their next action in the hypertext. If instead the response time is larger than a few seconds, the preferred interaction pattern may be asynchronous: the user is not forced to wait for the response but can perform other navigation actions after submitting the request. Later on, he will come back to the hypertext and look for the response, using the usual "pull" approach of Web applications.

In contrast, in solicit-response operations the Web service solicits some action from the Web application. Again, the construction of the response to the Web service may require the user's intervention, in which case the message exchange is necessarily asynchronous; otherwise the Web application may automatically build the response to the solicitation, e.g., in the form of an acknowledgment, in which case the message exchange takes place synchronously.

Note that for synchronous solicit-response operations, there is no user interface at all from the Web application standpoint; however, WebML can still be employed to describe the sequence of actions to be taken automatically by the application in response to the solicitation, as in the case of the notification operation (discussed in Section 4.2).

Furthermore, WebML can also be used to specify the data content of the message to be sent back to the soliciting Web service: if the chain of actions triggered by the solicit response operation ends with a link pointing back to the solicit response unit, the output parameters associated with such link can be used to build the SOAP message returned to the Web service. We illustrate this in Section 4.7.

4.4 Synchronous request-response operation

Figure 8 illustrates synchronous request-response operations. As usual, the WebML icon of the operation unit appears on left, and a usage example on the right.

Synchronous request-response operations are triggered when the user navigates

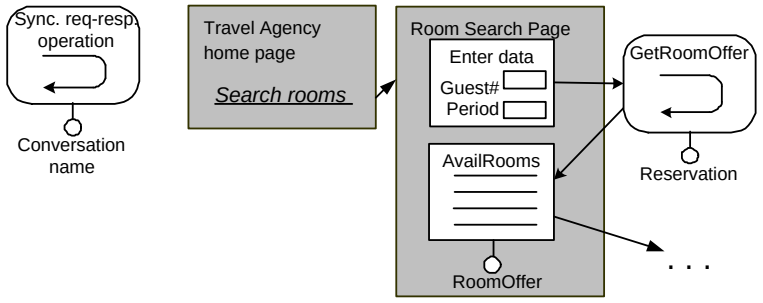

Fig. 8. WebML icon for synchronous request-response operation (left) and sample usage in a WebML hypertext (right).

one of their input links. The context parameters associated with the input links of the Web service operation are used to build the input message and the user waits until the arrival of the response message from the invoked service; she then can resume navigation from the page reached by the output link of the Web service operation unit.

In the hypertext in Figure 8, the Travel Agency user can browse to the *Room Search* page, in which an entry unit (Enter data) permits the input of search criteria, such as the number of guests and the desired reservation period. From this information, a request message is composed and sent to the GetRoomOffer operation of the*Hotel Broker* Web service (see Figure 5). The user then waits for the response message, containing a list of available room options (e.g., larger or smaller rooms, with shower or bath etc.). From these options, a set of instances of the RoomOffer entity are created, and displayed to the user by means of the *AvailRooms* index unit; the user may continue browsing from the *Room Search* page, e.g., by choosing one of the displayed offers and looking at its details.

4.5 Asynchronous request-response operation

Figure 9 shows the WebML unit for asynchronous request-response operations, and an example.

The upper half of an asynchronous request-response is triggered when the user navigates one of its input links; from the context parameters associated with these links, a message is composed and sent. The user resumes navigation immediately after message delivery, by following the output link of the upper half of the operation. The lower half of the operation is triggered upon the receipt of the response message coming from the previously invoked Web service.

The hypertext in Figure 9[8] describes an exchange of comments between the tourist and the*Hotel Broker* Web service. From the *Home* page of the Travel Agency a tourist can go to the *Send Comments* page, or to the *Read Answers to Comments* page. In the former case, the user can enter a comment's text in the *Edit Comment* entry unit; this input is used to create a new instance of the Comment entity in

[8]As done in Figure 7, we simplify the specification by omitting the login of the user, which is necessary for determining the content of the CurrentUser session parameter.

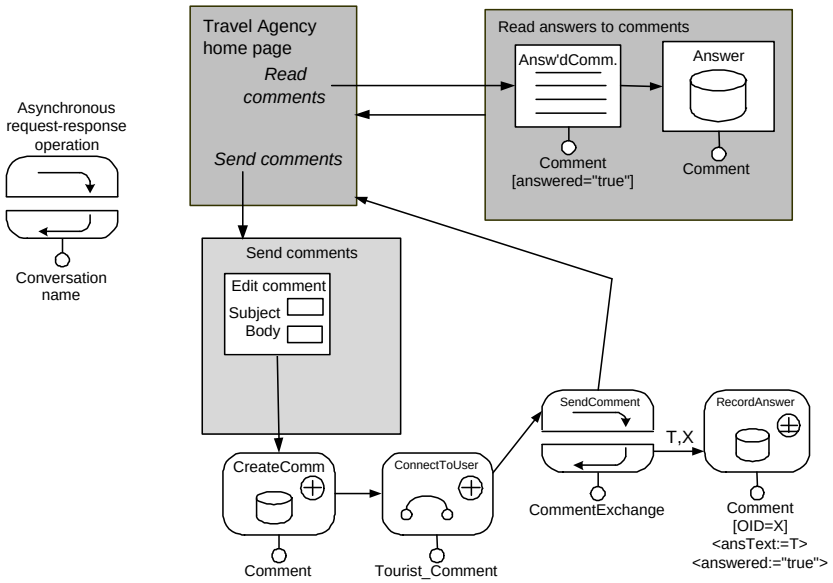

Fig. 9. WebML icon for asynchronous request-response operation (left) and sample usage in a WebML hypertext (right).

the local database of the Travel Agency application, and then to build the input message for the SendComment request-response operation.

Since the comment answer is not guaranteed to come immediately, the SendComment operation is used asynchronously: after sending the comment, the user is led back to the *Home* page, as modeled by the output link from the upper half of the *Send Comment* operation, which points to the Travel Agency *Home* page. When the response message arrives (lower part of the *Send Comment* operation), the text of the answer and the identifier of the comment (respectively corresponding to the parameter T and X associated with the output link of the *Send Comment* operation) are extracted from the message and used to modify the appropriate Comment instance. Later, if the user browses to the page named *Read Answers to Comments*, an index unit displays all his comments for which there is an answer.

The comments and answers exchanged in this hypertext belong to the *Comment Exchange* conversation, as denoted by the subscript underneath the asynchronous request-response WebML unit.

4.6 Asynchronous solicit-response operation

Figure 10 presents the WebML unit for asynchronous solicit-response operations and a sample WebML hypertext using it.

The upper half of the asynchronous solicit-response is triggered not by the user but by the reception of a message coming from an external Web service. Upon the receipt of the message, an arbitrary chain of WebML operations may be executed, typically storing in the Web application database the relevant information encapsulated in the message. The lower half of the asynchronous solicit-response operation

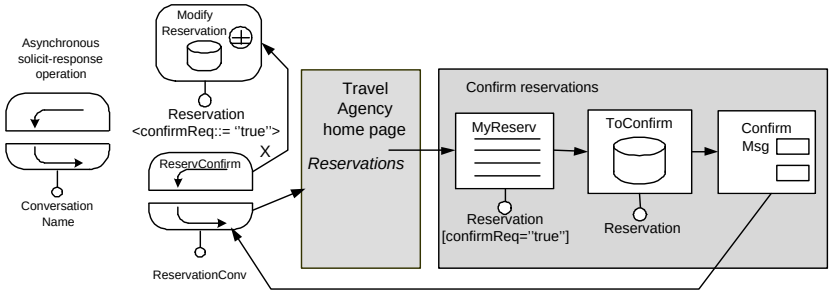

Fig. 10. WebML icon for asynchronous solicit-response operation (left) and sample usage in a WebML hypertext (right).

is triggered when the user navigates one of its input links; as usual, a message is composed from the context parameters associated with the input links, and then sent back to the Web service. Then, the user resumes navigation, following the link outgoing from the lower half of the unit.

In Figure 10, we give an example of hypertext incorporating the asynchronous use of the *Confirm Reservation* solicit-response operation of the *Hotel Broker* Web service (see Figure 5). This operation of the *Hotel Broker* service solicits tourists to confirm the details of a given reservation, that is, agree on the terms and conditions of the hotel etc.

The*Hotel Broker* service starts the conversation by sending a solicit message to the Web application; the reservation information is extracted from this message (as the value of the X parameter), and the appropriate instance of the Reservation entity is marked to record that confirmation is pending. At a later moment, when the user browses to the *Confirm Reservations* page, he is shown his pending reservations, and may pick one and confirm it by navigating the output link of the *Confirm* entry unit. This action triggers the bottom part of the solicit-response operation, which sends back the*Hotel Broker* Web service the confirmation data input by the user.

4.7 Synchronous solicit-response operation

Figure 11 presents the WebML unit for synchronous solicit-response operations and a sample WebML hypertext using it.

The synchronous solicit-response is not triggered by the user, but by the receipt of a message coming from an external Web service. Upon the receipt of the message, an arbitrary chain of WebML operations may be executed, typically storing in the Web application database the relevant information encapsulated in the message. The operation chain ends up with a link pointing back to the solicit-response unit, and the parameters associated with this link represent the data usable to build the response message to be sent back to the Web service in response to the solicitation.

Figure 11 illustrates an example of hypertext incorporating the synchronous use of the solicit-response operation; to do so, we propose a variation of the example of Figure 7, by adding an acknowledge message to the solicit operation whereby the*Hotel Broker* Web service sends news updates to the Travel Agency Web ap-

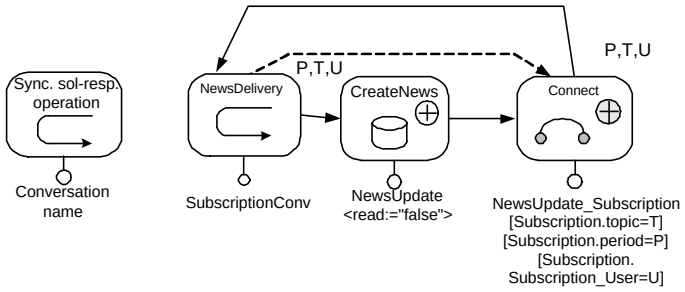

Fig. 11. WebML icon for synchronous solicit-response operation (left) and sample usage in a WebML hypertext (right).

plication. Specifically, the acknowledge message contains the data characterizing the subscription (User, Period, and Topic). A more sophisticated example of an acknowledgement message, involving the extraction of information from the database, will be illustrated in Figure 15.

The WebML diagram is similar to the one in Figure 7: the Web service operation initiating the interaction is the synchronous solicit-response NewsDelivery operation, which denotes the reception of the message coming from the*Hotel Broker* Web service. Upon the arrival of the message, the sequence of a create operation and a connect operation is executed, to store the novel news update and connect it to the proper subscription. In contrast with Figure 7, the operation chain now terminates with a link pointing back to the solicit-response operation. Such link carries as parameters the user OID (U), the topic name (T), and the period (P), which can be used to build the response message for the invoking Web service. The subsequent browsing of news can take place as described in Figure 7.

4.8 Additional WebML primitives for data marshalling and unmarshalling

To represent the application content, WebML relies on the E-R model, while Web service messages follow the XML Schema standard. Using a Web service operation (e.g. request-response) in a WebML hypertext requires the ability to:

—Construct an XML-structured message (e.g. request) from E-R data, extracted from the repository and/or passed as parameters from other WebML units;

—Decompose an XML-structured message (e.g. response) into E-R data to be stored in the repository and/or passed as parameters to various WebML units.

For its generality and conceptual cleanness, and for backward compatibility with the implementation of WebML in the WebRatio tool suite (described in Section 7), we have decided to keep the E-R model as the unifying conceptual data model for all data handled by the application (whether meta data, application data, or Web service messages). However, for facilitating the transformation of data from E-R to XML format and vice versa, we have also defined a canonical XML format, which corresponds to the E-R Model, with XML elements representing entities, relationships, and their attributes.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
   <entity id="Resort">
<instance>
   <attribute id="OID">res1</attribute>
   <attribute id="name">Kitzbuhel</attribute>
</instance>
<instance>
   <attribute id="OID">res2</attribute>
   <attribute id="name">Cortina</attribute>
</instance>
   </entity>
   <entity id="Slope">
<instance>
   <attribute id="OID">slo1</attribute>
   <attribute id="name">Streif</attribute>
</instance>
<instance>
   <attribute id="OID">slo2</attribute>
   <attribute id="name">Beginners</attribute>
</instance>
<instance>
   <attribute id="OID">slo3</attribute>
   <attribute id="name">Tofane</attribute>
</instance>
   </entity>
   <relationship id="Resort_Slope">
  <instance><source-oid>res1</source-oid> <target-oid>slo1</target- oid></instance>
  <instance><source-oid>res1</source-oid> <target-oid>slo2</target-oid></instance>
  <instance><source-oid>res2</source-oid> <target-oid>slo3</target- oid></instance>
   </relationship>
</root>
```

Fig. 12. Example of canonical XML representing a set of resorts, and the related slopes.

Figure 12 shows an example of the canonical XML representation of a piece of E- R content. Elements <entity> include, for each entity, the set of instances with respective attributes, while elements <relationship> include all the instances of the E-R relationship, described by means of the source and destination object identifiers.

The role of the canonical XML is that of an intermediate format between the E-R data representation of a WebML application and the arbitrary data representation assumed by the XML schema of the messages exchanged with external Web services. The benefits of such an intermediate encoding are twofold:

—It standardizes the conversion of XML data into E-R data stored in the native format of WebML. As we will see, two abstract WebML operations (called XML-in and XML-out) wrap the technical details for inserting a piece of canonical XML into the E-R data repository underlying all WebML units, and for extracting

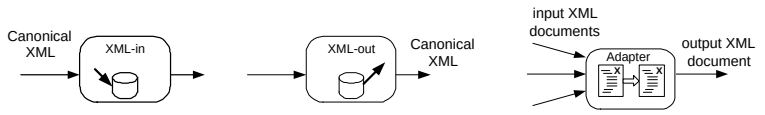

Fig. 13. XML-in, XML-out and Adapter units.

XML content from such an E-R repository.

—It facilitates the construction and decoding of the input and output messages necessary to interact with Web services. An incoming XML message, which conforms to its own service-dependent XML Schema needs only be converted into the canonical XML format to be automatically inserted (thanks to the XML-in operation) into the E-R data repository; dually, the E-R content of the data repository can be automatically extracted as a piece of canonical XML (thanks to the XML-out operation), which simplifies its translation into the XML Schema required by the Web service.

The transformation of E-R content into XML and vice versa is specified with the help of three additional WebML units, illustrated in Figure 13.

The **XML-in** unit takes as input a canonical XML fragment and stores in the underlying E-R repository the value of a specific property of the unit. From the physical viewpoint, the storage can be done in main memory (in which case the lifespan of the stored information is the user session), or peristently in the database.

The **XML-out** unit allows the selection of a set of objects belonging to an E-R sub-schema from the data repository and outputs their content as an XML fragment conforming to the canonical XML schema.

The **Adapter** unit takes in input n links carrying XML fragments, and emits along an output link another XML fragment with the desired schema and content; transformations are currently expressed using XSLT, but in principle any XML query language could be used, including XQuery or visual XML-to-XML mapping languages, like e.g. [Alto 2004; CLIO 2002]. Section 6.4 describes our current work on a visual mapping interface developed as an extension of the WebRatio design tool.

The XML-in, XML-out, and adapter units are necessary to express the data transformation tasks required by the exchange of messages with Web services. In the examples of sections 4.1 through 4.7 we have omitted these units the WebML diagram for the sake of readability. We now show a complete specification example, including both the Web service and the data transformation operations.

Figure 14 expands the diagram of Figure 8 showing the usage of adapters and XML-in units when making a synchronous request-response Web service call. From the *Room Search* page, the user may specify the number of guests and the desired stay period. When this information is submitted, the *Room Adapter* unit converts its input parameters into the XML format required for the input message of the GetRoomOffer operation of the *Hotel Broker* Web service. The response returned by the service contains a set of available room offers matching the user's request, formatted according to the XML Schema defined by the service provider. Such XML content is transformed into the canonical XML by the *Canonical Adapter*

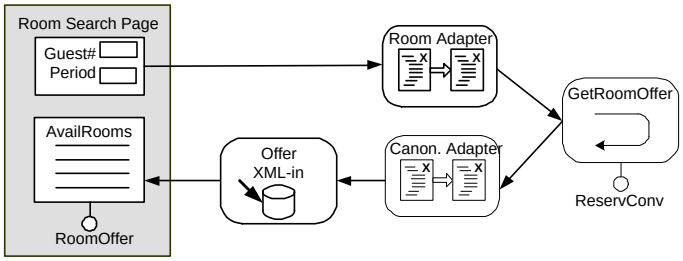

Fig. 14. Sample WebML hypertext using XML-in and Adapter units.

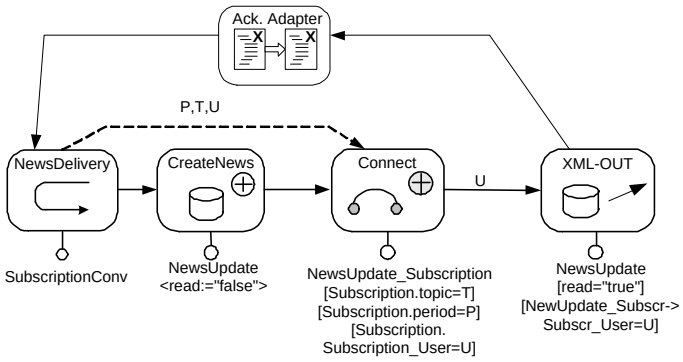

Fig. 15. Sample WebML hypertext using XML-out and Adapter units.

unit, and then stored in the data repository by the *Offer* XML-in unit, so that the *AvailRooms* index unit can present it to the user.

As an example of an XML-out unit, Figure 15 shows a variation of the solicit-response example of Figure 11, in which a more complex acknowledge message is sent back to the soliciting Web service. The response message contains the list of the news update read so far by the user to whom the solicitation refers; its content is extracted by an XML-out unit placed at the end of the operation chain, which builds a piece of canonical XML data containing the relevant news updates. This XML is given as input to an adapter unit, which converts it into the format accepted by the external Web service.

Within WebML hypertext, the adapter, XML-in and XML-out units can be considered at a lower level than other WebML units and links. This is because they serve as transformers on the data to be manipulated by other units, but do not have a direct impact on the interface shown to the user, nor do they enable interactions, as is the case for example of entry units. Therefore, their visualization can be optionally omitted from the WebML hypertext diagrams, to let designers concentrate only on the "conceptual view" of the application. Moreover, given that Web service units always require some XML transformation before and after the calls, a shorthand notation is introduced: each Web service unit is associated with two properties (input adapter and output adapter), which permits the designer to

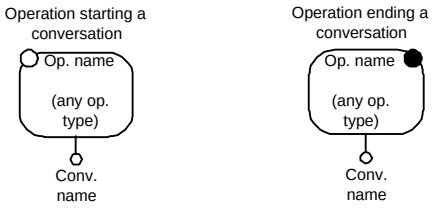

Fig. 16. WebML notation for starting and ending conversations.

specify the required XML-to-XML transformation necessary for invoking the unit without explicitly inserting into the WebML diagram the corresponding adapter units.

5. MANAGEMENT OF CONVERSATIONS

A WebML conversation is a collection of Web service operations, belonging to one or more Web services, used together to achieve a given application goal, and orchestrated by a Web hypertext. We graphically distinguish WebML operations that start/end a conversation using a white (resp. black) circular marker, as depicted in Figure 16. As discussed in the previous sections, each operation call takes place in the context of a given conversation and the conversation name appears below the operation's icon.

In this section, we discuss how WebML supports the notion of conversation. In Section 5.1, we present the metadata that describe the execution of WSDL operations belonging to a conversation; in Section 5.2 we discuss the effect of executing Web service operations and conversion start/termination on Web service metadata. Section 5.3 provides a full-fledged example of conversation.

5.1 Metadata for describing Web services

Metadata describe generic, application-independent data, whose purpose is making an information system self-descriptive; for instance, a classical use of metadata is found in relational catalogs, which specify the schema elements of one or more databases. In our context, metadata express the actual execution of Web service operations and their relationship to application data; more specifically, they correlate operations belonging to the same conversations, associate pairs of messages belonging to the same operation, and connect messages to the relevant users, either on-line or off-line.

Web service metadata, shown in Figure 17, describe conversations, operations, and messages. The Conversation entity is characterised by a conversation name and a unique conversation identifier, created at the start of the conversation. The Operation entity is characterised by an operation name and an operation identifier, created when the operation is initiated; a one-to-many relationship connects each operation instance to a given conversation, while each conversation instance is associated with multiple operations. Finally, the Message entity is characterised by a unique message identifier, and connected to a given operation, as either the input or the output message; it may optionally include the message's content, which is a

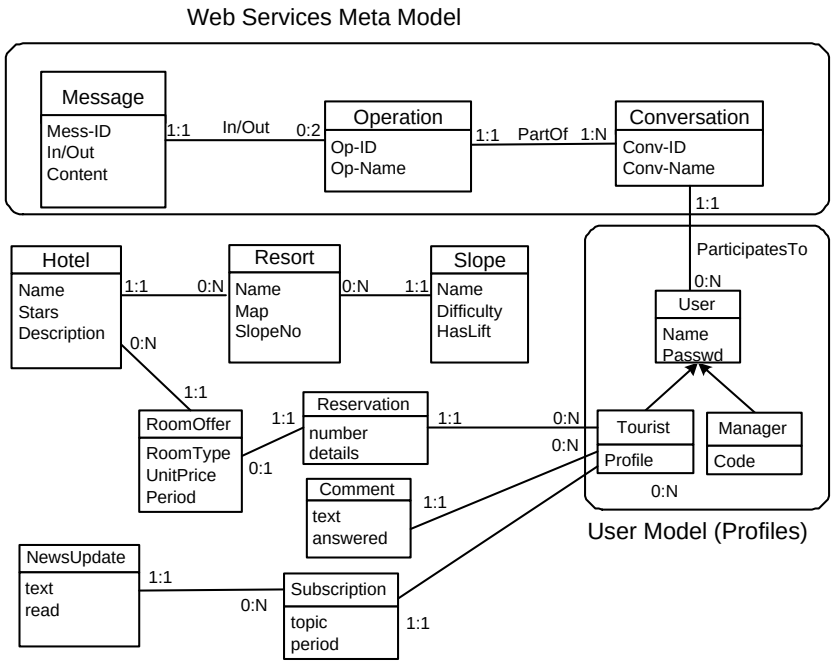

Fig. 17. Complete E-R diagram of the Travel Agency Web application including meta-data.

chunk of XML data. Operation instances are connected to either one or two message instances (more precisely, request-responses and solicit-responses have both input/output messages, while one-way and notification operations have a single message).

Web service metadata are connected to the rest of the schema, to enable the mapping of service calls to the appropriate users. The easiest way to perform this task is to associate conversations to users via a relationship, as described in Figure 17; the relationship between the user and a conversation is established by the first Web service operation triggered by the user's action[9]. Operations and messages can also be associated with application data (e.g., messages concerning hotels to the related hotel instance); this is not needed in the running examples. Note that the Web service meta-schema is identical in each application, but the actual connection of its entities to the rest of the application data schema is application-specific.

5.2 Metadata management

The management of instance-level metadata (creating and updating instances of the meta-level entities and relationships of Figure 17) is automatically performed when WebML primitives are translated into WSDL calls.

A conversation instance is created when an operation marked as "start conversation" is executed. Such an operation:

[9]As already stated in the Introduction, we are concerned with Web service conversations that always include at least one operation involving the user's interaction.

—Automatically assigns an ID to the newly created conversation instance;
—Connects it to the current user (represented by a username or, if missing, by a session identifier).

Whenever an operation (or its upper half, in case of asynchronous operations) is triggered, a new Operation instance is created with the proper parameters, and connected to the appropriate Conversation instance by the PartOf relationship.

Whenever a message is sent or received, Message instances are created and connected to the relative Operation instance.

A conversation ends when an operation marked as "endConversation" is executed. Note that in the case of two-messages operations, this means after the second message of the operation has been sent/received. Conversations may end also on a timeout, whose value depends on the application (e.g., a conversation for booking a room should not take more than 20 minutes)[10].

5.3 A complete example of conversation

In this section, for the purpose of illustration, we present a more complex conversation example, involving several Web service operations.

Figure 18 depicts the WebML hypertext for the complete room reservation process. The specification combines and extends various fragments presented in Section 4.4 and Section 4.6.

At the top of Figure 18, users create room reservations as explained before; notice that the *GetRoomOffer* operation starts the *ReservConv* conversation. When the user chooses one offer to accept, this triggers another synchronous request-response operation, AcceptOffer; this operation is also part of *ReservConv*. After accepting the offer, the tourist is led back to the Travel Agency *Home* page.

The reservation process resumes when the *Hotel Broker* service asks for the confirmation of the reservation (e.g., before a few days from the departure date), as depicted in the centre of Figure 18 (and explained in Section 4.6); the *ConfirmReservation* asynchronous solicit-response operation is still part of the same *ReservConv* conversation; when the tourist accepts the reservation, thus sending the response message to the *Hotel Broker* Web service, the reservation conversation is over. At a later time, the user may progress through the process by paying for the reserved rooms. This takes place at the bottom of Figure 18: the user navigates from the *Home* page to the *Payment* page, where he can provide his credit card number and expiration date using the *Enter Payment* entry unit. The user's input is structured into a message sent to the *Payment* Web service (see Figure 1); the service responds with a message that declares the payment either accepted or refused (e.g., due a credit card problem). The outcome of the Web service call is assessed by an IF operation unit, which makes the navigation proceed along different paths based on the outcome of a test; in the present case, the IF unit checks a Boolean value extracted from the response of the payment service. If the payment fails, the user is brought back to the *Payment* page and asked to re-submit the required information until

[10]In this paper we do not consider the management of conversations that terminate with a timeout (and in general we do not discuss how metadata relative to completed applications should be disposed of) or with an operation failure. Clearly, the conversation-related metadata should be cleaned-up according to a given garbage-collection policy.

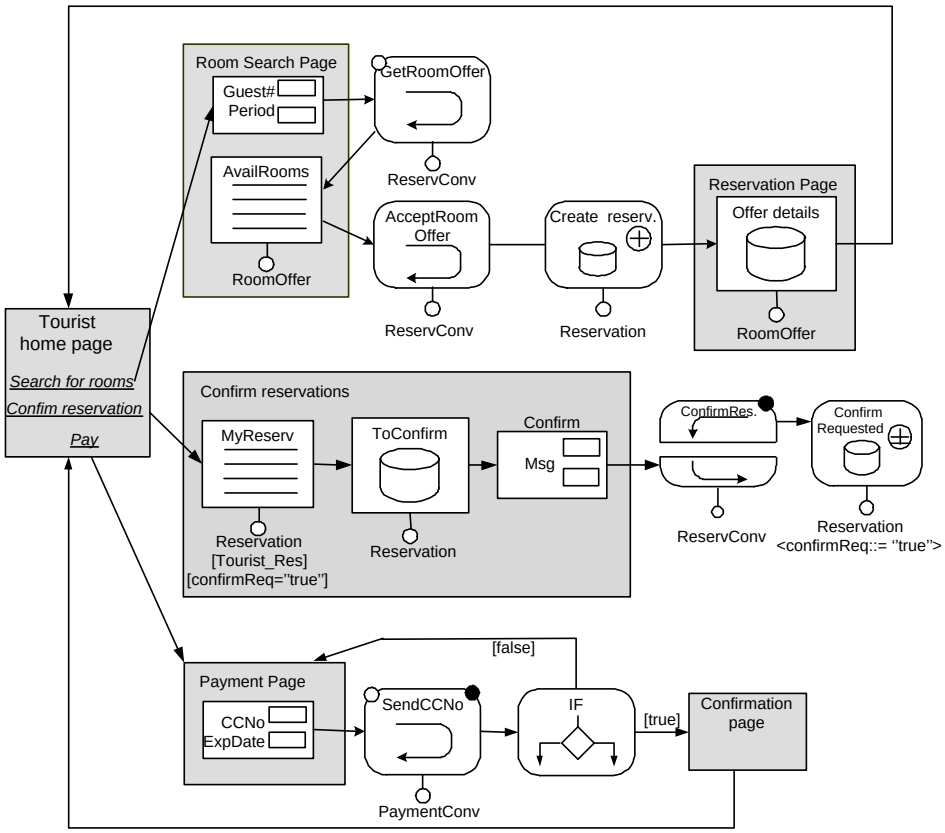

Fig. 18. Complex conversations for room reservation and payment.

the payment succeeds. Each payment attempt produces a new instantiation of the *PaymentConv* conversation. Finally, when the payment information is accepted, the user is led to the *Confirmation* page, on which a message informs him of the success of the reservation.

Notice that the grouping of Web service calls into conversations is a matter of design: the Web application designer could have chosen, for example, to collapse all Web service calls into a single conversation (including the payment process), or, on the contrary, to split the process in many small conversations. The conversations actually used in Figure 18 reflect one possible division of actions related to the reservation, from those related to payment.

6. DEPLOYING WEB SERVICES FOR REMOTE PEERS

So far, we have analyzed the interaction between a Web application and a Web service by considering the Web application as a user of an existing Web service. For example, the Travel Agency Web application uses the *Hotel Broker* Web service, calling it as a result of the human users' interaction.

However, a Web application may also expose part of its functionality as a Web

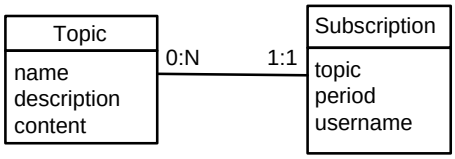

Fig. 19. Fragment of the data schema of the application implementing the NewsSubscription and NewsDelivery operations of the*Hotel Broker* Web service.

service. The inherent symmetry of WSDL requires that two parties are involved in any message exchange. Given a Web service S, the service obtained by changing the "direction" of all messages (transforming one-way operations into notifications and vice-versa, transforming request-response into solicit-response operations and vice-versa) is commonly termed "the dual of S" [Andrews et al. 2002]. For example, the Travel Agency Web application may be seen as "the dual" of the*Hotel Broker* Web service, whose WSDL operations are derived by reversing the roles of the two parties shown in Figure 5.

Therefore, the same high-level approach to the modeling and automatic implementation of Web applications using Web services can be also exploited for specifying and publishing the dual of a given Web service, by the simple symmetry of WSDL.

Thus, the Web services extensions of WebML provide a high-level, elegant model for specifying and automatically deploying Web services.

We stress that the consumer and the provider of Web services need not be both designed and implemented in WebML; each party can be either coded by hand, or deployed automatically using any platform. The advantage of the model-driven approach to the development of Web applications and services resides in its high level and declarative character.

As a next step, we show how a Web application designed using WebML can be used to specify the news delivery Web service that runs at the *Hotel Broker*. This service was presented from a consumer's perspective in Section 4.1 and 4.2, where we have shown how the Travel Agency Web application can call it. Supposing that the*Hotel Broker* manages a database with a schema like the one shown in Figure 19, we can assume that whenever the administrators of the*Hotel Broker* application updates a topic, e.g., by means of a Web-based content management application, the modified data is assembled into an XML message sent to all subscribers interested in the topic.

The hypertext in Figure 20 describes the internal functioning of the *Hotel Broker* Web service. When a message is received for the NewsSubscription operation, a new instance of the Subscription entity is created and connected to a topic specified in the subscription, and a new conversation called *InfoConv* is started (part **A** of the hypertext). In the *Update Topics* page of the*Hotel Broker* content management application (part **B**) of the hypertext), administrators can pick a topic to update in the *AllTopics* index unit, and display the current topic details and all the active subscriptions; the administrator can then update the information by inserting new data using the *Enter New Info* entry unit. Submitting the new information activates

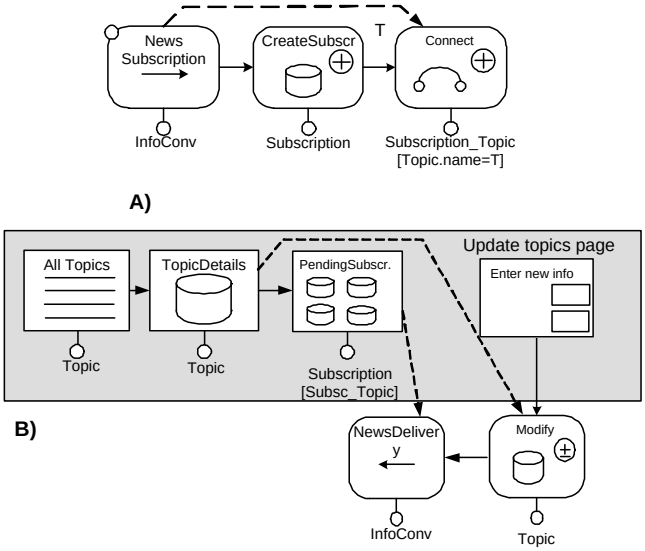

Fig. 20. WebML hypertext defining the behavior of the NewsSubscription and NewsDelivery operations of the *Hotel Broker* Web service.

a sequence of operations: the selected topic is updated in the database of the*Hotel Broker* (*Modify* unit), and a NewsDelivery message is sent to all the peers that have an active subscription relative to the updated topic. Note that for a given subscription, the NewsDelivery operation may be triggered multiple times, because the administrator can update the same topic multiple times.

7. IMPLEMENTATION

The Web service extensions of WebML illustrated in the previous sections have been implemented inside a CASE tool for Web application development, called WebRatio [WebRatio 2004]. WebRatio is an integrated environment for the visual specification of Web applications in WebML, and the automatic generation of code for the J2EE and Microsoft .NET platforms [Ceri et al. 2003].

7.1 WebRatio architecture

The architecture of WebRatio (shown in Figure 21) consists of two layers: a design layer, providing functions for the visual editing of specifications, and a runtime layer, implementing a Model–View–Controller[11] Web application framework [Ceri et al. 2003]. These layers are connected by the WebRatio code generator, which exploits XML transformations to map the visual specifications edited in the design layer (which are stored as XML) into application code executable within the runtime layer.

The design layer, code generator, and runtime layer have a plug-in architecture:

[11] MVC is a classic software engineering pattern that cleanly separates the different functions of an application (control, interface, and business logic), widely adopted by Web development platforms.

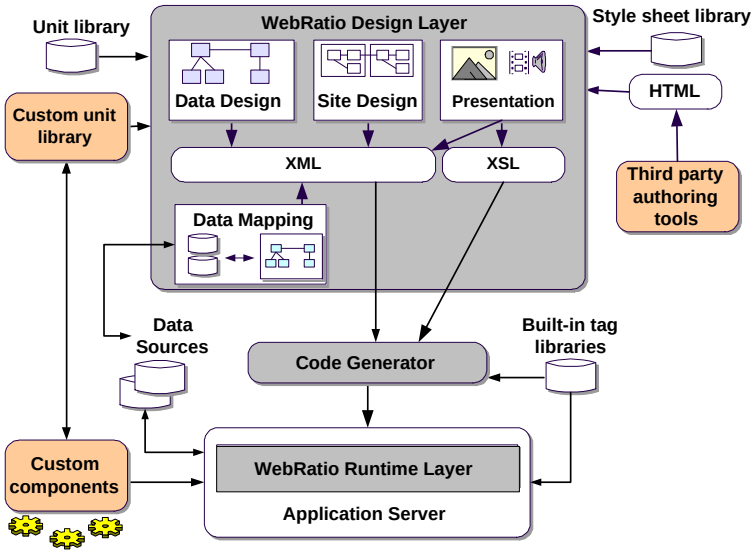

Fig. 21. Overview of the main layers of WebRatio.

new components can be described using XML descriptors and made available to the design layer as custom WebML units, the code generator can be extended with additional XSL rules to produce the code necessary for calling the plug-in components, and the components themselves can be deployed as business objects in the runtime application framework.

In the rest of this section, we briefly illustrate the features of the three layers of the WebRatio architecture and discuss the extension implemented to support Web service interaction.

7.2 Runtime layer and extensions for the interaction with Web services

The runtime layer of WebRatio follows the general organization of the MVC architecture, which comprises three main components:

(1) The *Controller*, which accepts the client's requests and dispatches them to the business object in charge of serving them. The dispatching is mediated by special-purpose classes, called actions, which decouple the Controller from the business objects.

(2) The *Model*, which contains the state objects representing the internal status of the application, and the business objects (called runtime services in the WebRatio jargon) capable of responding to the client's requests.

(3) The *View*, which exploits the state objects of the Model to build the interface to be exposed to the user. In the Web context, the View consists of page templates embedding data-display custom tags.

To support the interaction with Web services, the original runtime architecture of WebRatio has been modified, resulting in the framework shown in Figure 22.

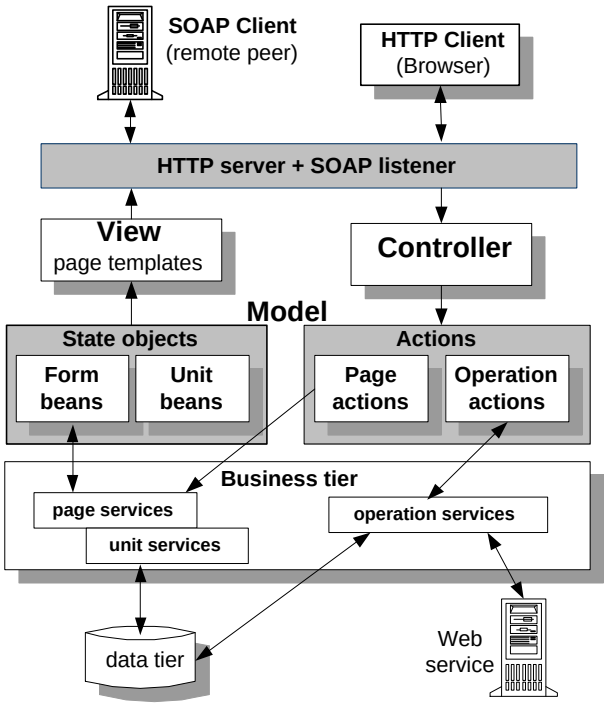

Fig. 22. Architecture of the WebRatio Runtime Layer extended to support Web service interaction.

The Web front end has been extended with a SOAP listener to support both human users posing regular HTTP requests with a browser, and remote applications sending SOAP messages. Each request is sent to the Controller, which dispatches it to the model action in charge of serving it. If the request is an HTTP request for a page, a page action is invoked, which extracts the content from the data sources and builds the state objects (called unit beans[12]) necessary for the page templates of the view to assemble the HTML response. Otherwise, if the request is a SOAP message or an HTTP request for an operation, the Controller sends it to an operation action, which performs the required function and returns a status code to the Controller, who decides what to do next. In particular, the operation action may exploit a runtime service that interacts with the remote Web service, as explained next.

7.2.1 *Runtime services for the interaction with Web Services.* The call to a Web Service has required the implementation of a new kind of runtime services (called Web Service operation services) in the business tier. A Web Service operation service is a runtime component capable of:

—decoding the information transported by the input link(s) of a WebML Web service unit;

[12]The unit beans are JavaBeans objects used to store and manage E-R content inside the WebRatio runtime framework.

—assembling the message for invoking a remote Web service from the information associated with the input link(s);

—invoking the remote Web service and collecting its XML response message;

—transforming the XML response into a set of unit beans, so that the content of the response message is seamlessly available to the other WebML units for data display and manipulation.

In the current implementation, the Web service to call can be determined both at compile time, by means of a UDDI [Belwood et al. 2002] wizard for inspecting service registries and for binding a specific service to a Web service unit in the hypertext diagram, and at run-time, by passing the endpoint URL of the service to call as a parameter of the Web service unit. At the moment, it is not possible to dynamically negotiate the actual Web service URL to call among a set of candidates; future versions may offer dynamic service negotiation at runtime, based on the specification of the required service level.

The reception of incoming messages from Web service operations has been realized by deploying a message-acceptor Web service for each expected inbound message. In this way, we have reused the same implementation technique adopted for exposing arbitrary pieces of WebML hypertext (that do not involve user interactions) as Web services.

We discuss this technique in the next section.

7.3 Code generation and its extensions for the publication of Web services

The WebRatio Code Generator transforms the WebML specification into a set of software components implementing the specified application. In the web context, the generated components include:

—The configuration file of the Controller, which contains the navigation control flow logic.

—The action classes, which are invoked by the Controller and in turn invoke the runtime services.

—The XML configuration files (called runtime descriptors) of the runtime services. In particular, every WebML unit in the diagram produces one runtime descriptor configuring the behaviour of a generic runtime service. For example, a specific index unit generates a runtime descriptor containing, among other things, the SQL code for extracting the content of the index from the data repository.

—The server-side templates for dynamically building the actual pages of the application.

Code generation starts from the WebML model of the application, which is encoded as an XML file[13], and proceeds along the steps illustrated in Figure 23. The code generator is implemented as a set of XSLT rules, executed by a standard XSLT processor.

In the first phase, a fixed set of XSLT rules (called model expansion rules) annotate the WebML model with redundant information that make the specification

[13]The XML DTD used for representing WebML specifications is available at http://www.webml.org.

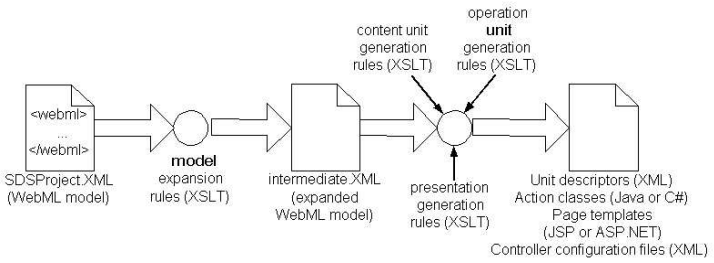

Fig. 23. Main steps of the WebRatio code generation process.

more explicit and simplify the subsequent code generation rules (e.g., data typing information is copied from the E-R specification into the description of the pages). In the second phase, a set of customizable XSLT rules apply to the pages and operation units of the specification and transform them into the source code of the application. Typically, user customization focuses on presentation generation rules, which dictate the presentation-oriented features of the generated page templates. In particular, the deployment of a piece of a WebML application as a Web service requires writing presentation rules addressing the specificity of Web service publication.

7.3.1 *Exposing parts of the Web application as Web services.* The current implementation supports the export of hypertext fragment not requiring user interaction as a Web service. This functionality allows remote peers take advantage of the information and functions offered by the Web application.

Notice that conceptually, there is no obstacle to exporting more complex hypertext fragments, *including* user interactions, as Web services; indeed, this is one of the most interesting aspects of our work, and we illustrated it in Section 6.

To achieve Web service publication of simple hypertext fragments, the WebRatio code generator has been extended with novel presentation rules producing the following pieces of code:

—The WSDL description of the service, which specifies the messages exchanged by the automatically generated Web service.

—A SOAP message template, which produces the XML response message to be returned to the client. The SOAP message template fulfils a similar role to that played by page templates in a Web application, because it may use the model objects to build the message returned to the client application. This message can be simply a static acknowledge of the invocation received from the SOAP client (e.g., for one-way operations) or may contain arbitrary information dynamically computed by the WebML fragment (e.g., for solicit- response operations).

7.4 The Design Layer and its extension for Web services

WebRatio includes a GUI for designing Web applications in WebML. Figure 24 shows a screenshot representing a portion of Web site that includes Web service operations. The GUI contains a Project Tree, with all the items of the current project (top left), a work area displaying the WebML diagrams and a tool palette

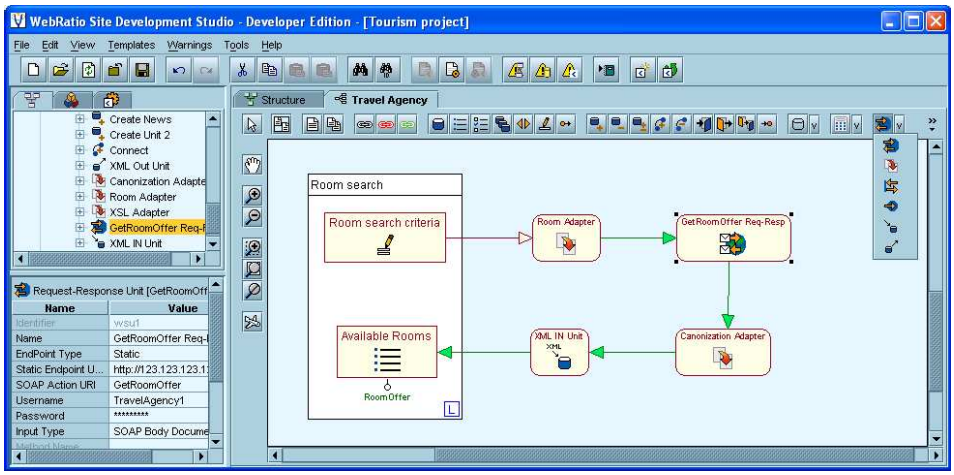

Fig. 24. WebRatio used for modeling Web services invocation.

listing the available WebML constructs (links, pages, and units) (right), and a property panel for editing the currently selected object (bottom left). The screenshot in Figure 24 shows a request-response Web service operation, selected in the work area and displayed in the property panel. On the right corner of the tool palette, a drop down list contains the Web service units, which have been added to WebRatio as a custom unit package. The depicted hypertext is the one illustrated in Figure 14.

Figure 25 shows the WebML model of a published Web service. A solicit response unit is exploited to implement the hypertext presented in Figure 11 (and Figure 15): the hypertext fragment shown describes the reception of a News update at the Travel Agency Web application, the update of the local database and the provision of an acknowledge message to the *Hotel Broker*. Indeed, once a piece of news is received, *Create News* and *Connect* units update the Travel Agency database, then the XML-out and Adapter units build up a new message to be sent back as acknowledgement.

7.5 Visual design of XSL transformations

Since the XSL writing required by adapter units is a repetitive and low- level task, the GUI has been extended with a wizard for avoiding the manual coding of transformations. In particular, we studied a visual representation of transformations from generic XML documents to the canonical E-R representation discussed in Section 4.

Our focus is slightly different from other document transformation tools (e.g., Mapforce by Altova [Alto 2004]), because we concentrate on a particular case of transformation, in which the destination document schema is known, and corresponds to the E-R canonical format. Therefore, we do not cover the full expressive power of XPath / XSLT, but offer a simple and intuitive, yet formal, visual representation of the typical transformations from generic XML to E-R content. The supported transformations typically include:

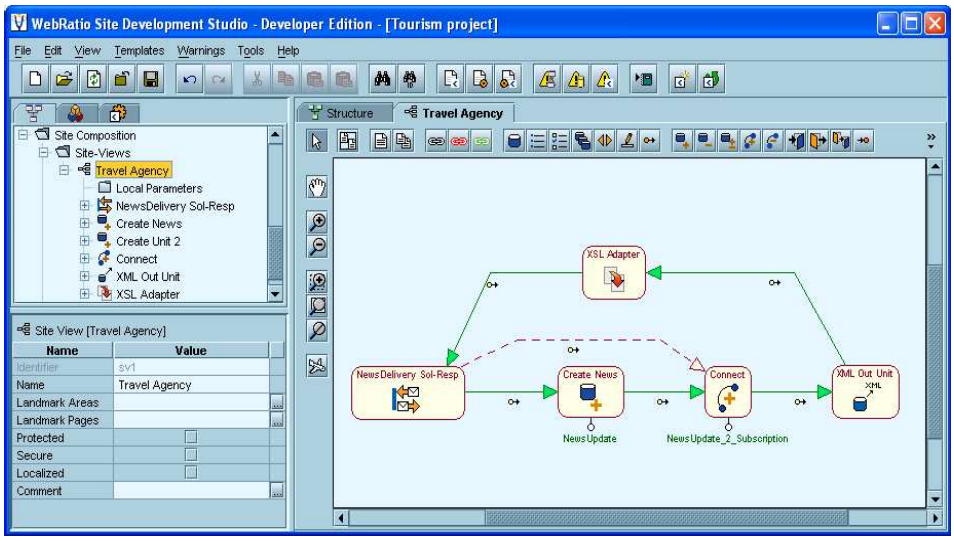

Fig. 25. Snapshot of the WebRatio tool, used for modeling Web services publication.

—The mapping of an XML element and its sub-elements into a single entity. The mapping can be either immediate, when the entity attributes correspond to the element attributes (e.g., the Slope entity and Slope element in Figure 26), or slightly more complex, when an entity maps to one XML element with multiple nested sub-elements (e.g., if we map the Hotel entity to the Hotel XML element and to its sub-element Address, which in turn may have sub-elements Street, Number and ZIPcode, not shown in Figure 26).

—The mapping of elements and sub-elements to multiple entities, connected through a relationship. This occurs with multi-valued XML properties or sub- elements, which needs to be mapped onto a pair of related entities (e.g., the Slope XML element is mapped over the Slope and Resort entities in Figure 26).

—The mapping of sub-elements belonging to several distinct super-elements to a set of interrelated entities.

—Simple value transformations, like trimming and concatenation of element and attribute data, which are then mapped onto entity attributes.

These transformations are performed using the visual interface illustrated in Figure 26, which shows, on the left side, a tree representing the source XML document, and on the right side, the entities and relationships of the target E-R schema. Transformations can be designed by drawing lines from XML elements to E-R attributes, relationships and entities.

For complex transformations involving ID references between XML elements, intermediate boxes are used for connecting the referenced elements (see Figure 26).

In the example of Figure 26, instances of the XML element City are mapped into instances of E-R entity Resort, and instances of element Slope are transformed into instances of entity Slope, with the respective attributes. The "REL IDREF" box allows one to specify the E-R relationship between Resort and Slope, based on the

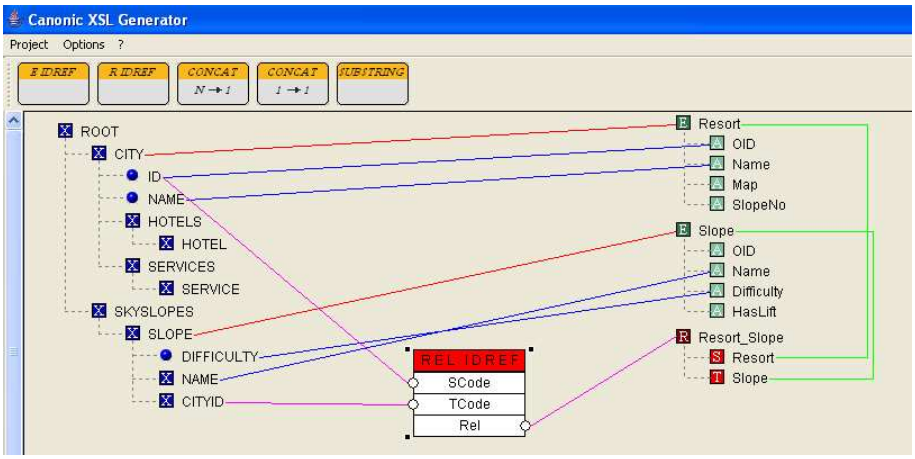

Fig. 26. Snapshot of the XML-ER mapping tool.

XML IDREFS attributes connecting the City and Slope XML elements.

The mapping drawn by the designer is stored into an XML file, which can be used to automatically generate the appropriate XSL files to be associated to the adapter units performing the needed document transformation.

7.6 Experience and online demo

The Web Service extensions of WebML implemented in the WebRatio tool suite have been already exploited in the construction of several industrial applications. In this section we briefly overview the major achievements and present some of the results of this experience.

The most complex application is the MetalC Platform, described at www.metalc.it (in Italian), which aims at providing different e-services to the small and medium entreprises (*SME*s) of the mechanical district of the Como province. The ultimate goal of the project is to make SMEs restructure their core business process, by using Web technologies and interoperating in an open environment based on Web services. The MetalC platforms comprises three main applications:

(1) A Web-enabled **product configurator**, whereby users of a SME can specify the bill of materials of their products, the data entry workflow that customers must follow to order a specific configuration, and the computation rules that determine the quantitative parameters emitted in output by the configurator (e.g., price, lead time, payment options, discount level, etc). This application is deployed both as a Web front-end, embedded within the company's institutional Web site, and as a Web service, callable by other applications, including the other modules of the MetalC platform.

(2) A Web-enabled **concurrent engineering environment**, where technicians can share, redline and version CAD files representing product blueprints and technical drawings. Again, the application has a double interface, because it is open to both humans and remote applications (for example, the product

configurator can fetch the latest technical drawings of a configuration from the engineering environment).

(3) A **document exchange application**, which connects to the enterprise information systems of a partner, fetches business documents (e.g., offers, orders, production plans) and allows them to be transmitted to a remote partner, and inserted into the latter's enterprise information system. The proprietary information systems of the participating SMEs are wrapped by means of data extraction and insertion Web services, and the MetalC platform acts as a sophisticated orchestrator of the workflow needed for checking the documents produced by the enterprise information system, possibly enriching them (e.g., with configurations and technical drawings produced by the other MetalC modules) and dispatching them to the destination legacy system.

Nineteen enterprises currently use the services of the MetalC platform. The configurator and the concurrent engineering environment are fully operational, the document exchange application is finished, and the software houses of the various SMEs are completing the data insertion and extraction Web services. Over fifty different Web services have been specified and deployed using WebML and WebRatio, and the Web front-ends for executing the various functions contain over ninety Web service calls.

From the viewpoint of the application's developers and maintainers, working with a unified conceptual model of the application Web interfaces, of the deployed Web services, and of the Web service calls has been a great benefit, due to the ability to access a visual representation of all system facets. This representation has helped especially in turning functions initially conceived for Web usage by humans, into Web services callable from other applications. For instance, both the configurator and the concurrent engineering environment were initially conceived as standalone applications, and only later in the project the requirement arose for accessing part of their functionality as a service from within another application. This requirement has been fulfilled by publishing request-response and solicit-response operations publishing selected functions of the application (e.g., the retrieval of technical documents for which a user has the proper access rights).

A second application is the Web portal www.pattichiari.it of the Italian Banks' Association, which offers to the users a number of services related to the choice of debit/credit accounts and to the purchase of financial products. Some of the offered functions require the use of remote Web services. For example, the user can ask for a location-based map showing the ATMs close to a given address. The Pattichiari portal was a good benchmark of the performance of the implementation, because it features more than 30.000 unique visitors per day.

8. EXPRESSIVE POWER COMPARISON BETWEEN WEBML AND BPEL WEB SERVICE COMPOSITION

This section is devoted to the evaluation of the expressive power of WebML with respect to BPEL4WS. For this comparison, we take advantage of a codified set of workflow patterns, proposed by [van der Aalst et al. 2002; Wohed et al. 2003], and evaluate to what extent such patterns are covered by both languages. Table 1 summarizes the results of the comparison: if a pattern is supported directly by

PATTERN	BPEL4WS	WebML
Sequence	+	+
Parallel Split (AND-split)	+	P
Synchronization (AND-join)	+	+
Exclusive Choice (OR-split)	+	+
Simple Merge (OR-join)	+	+
Multi Choice	+	P
Synchronizing Merge	+	+
Multi Merge	−	+
Discriminator	−	+
Arbitrary Cycles	−	P
Implicit Termination	+	+/−
MI without Synchronization	+	P
MI with a Priori Design Time Knowledge	+	+
MI with a Priori Runtime Knowledge	−	+/−
MI without a Priori Runtime Knowledge	−	+
Deferred Choice	+	P
Interleaved Parallel	+/−	+
Routing		
Milestone	−	P
Cancel Activity	+	−
Cancel Case	+	−

Table I. BPEL and WebML expressive power comparison.

a language construct, the rating is "+"; if the pattern is supported indirectly, the rating is "+/-"; if support of a pattern is limited, the rating is "P" (partial support); if the pattern is not supported at all, or is supported in an unnatural way, the rating is "-".

The features of BPEL are described in [van der Aalst et al. 2002], while the features of WebML covering the workflow-oriented patterns are illustrated in [Brambilla et al. 2002]. WebML directly supports most patterns of Table 1, if the WebML workflow extensions presented in [Brambilla et al. 2002] are considered; however workflow patterns can be modeled also without the above-mentioned extensions, although indirectly and in a less elegant and concise way (these alternatives were discussed in [Ceri and Manolescu 2003]).

WebML supports the sequence constructor and the AND/OR split/join primitives, thanks to the use of conditional operation units (IF and SWITCH), and of session-based parameters. The support of the *AND-split*, *Multichoice* and *Multiple Instances without Synchronization* patterns is rated "partial", because WebML cannot express the cases in which the automatic start of parallel execution threads is required. Indeed, every event in WebML (user's click or Web service notification) can activate only one activity, e.g., a sequence of operations or the computation of a page. As a consequence, the *Parallel Split* pattern is supported, but with the limitation that only one of the branches is automatically executed. The *Arbitrary Cycles* pattern is only partially supported for the same reason: activity cycles cannot contain tasks automatically launched in parallel. *Implicit Termination* (i.e.,

the automatic recognition of the correct termination of the process even if no terminal point has been explicitly reached) is not directly supported either, in the sense that WebML can express the interface for letting the user terminate a process, but cannot express an autonomous activity for terminating a process. The *Multiple Instances* patterns are in general supported by means of the sequential creation or selection from a work queue of an arbitrary number of data items, each of which is associated with a different instance of some workflow task. The *Deferred Choice* and *Milestone* patterns are only partially supported because WebML lacks a model of condition-based or temporal events. Therefore, activities cannot be enabled/disabled based on time deadlines or on the occurrence of arbitrary conditions. The only classes of events handled by WebML are users' interactions and messages: a user-generated event or a message received by a Web service is the only means of triggering activity execution. *Cancel Activity* and *Cancel Case* are not supported primitively and in a general form, although they can be modeled by specifying the ad hoc interfaces and operations necessary for activity and case cancellation.

9. CONCLUSIONS AND FUTURE WORK

This paper has presented a conceptual approach to Web service design, which has been formally defined as an extension of the WebML language, and implemented in WebRatio, a CASE tool supporting the development of applications with WebML. The proposed modeling primitives rely on a Web service meta-model, and on new WebML operation units expressing the various WSDL operation types and the XML data processing necessary for integrating the messages exchanged with the Web services and the content managed by the Web application. The Web service extensions are currently being used in several industrial projects, and are packaged as a collection of plug-in components that will be soon commercialized together with WebRatio.

Several extensions to the present work are planned. A challenging research direction is the verification of consistency between hypertext and conversation specifications, which aims at demonstrating, for example, that a given Web interface is a correct implementation of the dialogue required by a Web service conversation. Verification could follow an approach like the one presented in [Deutsch et al. 2004] and express the mutual consistency properties of hypertexts and Web service conversations in temporal logic. To check the compliance of a WebML specification to a conversation model, linear temporal logic will likely be sufficient, even though decidability problems may require restrictions on the usable WebML constructs. Joint work in this direction with the authors of [Deutsch et al. 2004] is in progress.

Another line of research concerns the semantics of conversation failure (e.g., how to model conversations that do not come to a proper conclusion) and the repair of failed conversations, which constitute an important, yet rather neglected, issue. Aspects to be considered include garbage- collecting all the data of failed conversations, and expressing repair actions executable upon failure in order to resume a conversation.

Another interesting direction for future work exploits the concept of design patterns. Well-known in the area of software engineering, design patterns have proved useful also for Web application design [Rossi et al. 1999; Ceri et al. 2001]. De-

sign patterns involving Web service operations can be identified and expressed in WebML: for example, a "call-choose" pattern includes a request-response operation and a conditional unit testing the returned message; a "call-store" pattern materializes the answer received from the service; a "publish-subscribe" pattern includes a pair of one-way and notification operations, and the logics for constructing the message sent to subscribers etc. Structured Web service conversations will likely lead to more complex service-centric design patterns.

ACKNOWLEDGMENT

The authors thank Susan Davidson for her careful proofreading of this work.

REFERENCES

Abiteboul, S., Bonifati, A., Cobna, G., Manolescu, I., and Milo, T. 2003. Dynamic XML documents with distribution and replication. In *Proceedings of the 29th ACM SIGMOD International Conference on the Management of Data*. ACM, San Diego (CA), USA, 527–538.

Abiteboul, S., Vianu, V., Fordham, B., and Yesha, Y. 1998. Relational transducers for electronic commerce. In *Proceedings of the 24th ACM SIGACT-SIGMOD-SIGART Symposium on the Principles of Database Systems*. ACM, Seattle (WA), USA, 179–187.

Alto 2004. Altova's MapForce mapping tool. Available at http://www.altova.com/products_mapforce.html.

Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Leymann, J. K. F., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., and Weerawarana, S. 2002. Business process execution language for web services. Available at http://www.ibm.com/developerworks/webservices.

Atzeni, P., Mecca, G., and Merialdo, P. 1998. Design and maintenance of data-intensive web sites. In *Proceedings of the 1998 International Conference on Extending Database Technologies*. Lecture Notes in Computer Science. Springer, Valencia, Spain, 436–450.

Baresi, L., Garzotto, F., and P.Paolini. 2000. From web sites to web applications: New issues for conceptual modeling. In *Proceedings of the ER 2000 Workshops on Conceptual Modeling Approaches for E-Business and The World Wide Web and Conceptual Modeling*. Lecture Notes in Computer Science. Springer, Salt Lake City (UT), USA, 89–100.

BEA 2004. BEA Weblogic Workshop. Available at http://www.bea.com/content/products/workshop/.

Belwood, T., Clement, L., Ehnebuske, D., Hately, A., Hondo, M., Husband, Y. L., Januszewski, K., Lee, S., McKee, B., Munter, J., and von Riegen, C. 2002. Universal Description, Discovery and Integration Technical Committee Specification. Available at http://uddi.org/pubs/uddi_v3.htm.

Berardi, D., Calvanese, D., Giacomo, G. D., Lenzerini, M., and Mecella, M. 2003. Automatic composition of e-services that export their behavior. In *Proceedings of the 1st International Conference on Service Oriented Computing (ICSOC 2003)*. Springer, Trento, Italy, 43–58.

Brambilla, M., Ceri, S., Comai, S., Fraternali, P., and Manolescu, I. 2002. Specification and design of workflow-driven hypertexts. *Journal of Web Engineering 1,* 2, 163–182.

Bultan, T., Fu, X., Hull, R., , and Su, J. 2003. Conversation specification: A new approach to design and analysis of e-service composition. In *Proceedings of 12th International World Wide Web Conference*. The International World Wide Web Conference Committee, Budapest, Hungary, 403–410.

Casati, F. and Shan, M.-C. 2001. Dynamic and adaptive composition of e-services. *Information Systems 26,* 3, 163–203.

Ceri, S., Fraternali, P., and Bongio, A. 2000. Web modeling language (webml): a modeling language for designing web sites. *Computer Networks 33,* 1-6, 137–157.

Ceri, S., Fraternali, P., Bongio, A., Brambilla, M., Comai, S., and Matera, M. 2002. *Designing Data-Intensive Web Applications*. Morgan Kaufmann, Amsterdam, Holland.

Ceri, S., Fraternali, P., and Matera, M. 2001. Conceptual tools for enhancing design reuse.

Ceri, S. and Manolescu, I. 2003. Constructing and integrating data-centric web applications: Methods, tools, and techniques. In *Proceedings of the 29th International Conference on Very Large Databases.* Morgan Kaufmann, Berlin, Germany, 1151.

Ceri, S., P.Fraternali, Acerbis, R., Bongio, A., Butti, S., Ciapessoni, F., Conserva, C., Elli, R., Greppi, C., Tagliasacchi, M., and Toffetti, G. 2003. Architectural issues and solutions in the development of data-intensive web applications. In *Proceedings of the 1st Biennial Conference on Innovative Data Systems Research.* online proceedings, Asilomar (CA), USA.

Christophides, V., Hull, R., and A.Kumar. 2001. Querying and splicing of XML workflows. In *Proceedings of the International Conference on Cooperative Information Systems.* Lecture Notes in Computer Science. Springer, Trento, Italy, 386–402.

CLIO 2002. A schema mapping tool. Available at http://www.cs.toronto.edu/db/clio/.

Deutsch, A., Sui, L., and Vianu, V. 2004. Specification and verification of data-driven web services. In *Proceedings of the ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems.* ACM, Paris, France.

Fernandez, M. F., Florescu, D., Kang, J., Levy, A. Y., and Suciu, D. 1998. Catching the boat with Strudel: Experiences with a web-site management system. In *Proceedings of the 24th ACM SIGMOD International Conference on the Management of Data.* ACM, Seattle (WA), USA.

Florescu, D., A.Gruenhagen, and Kossmann, D. 2002. XL: A programming language for web service specification and composition. In *Proceedings of the 11th International World Wide Web Conference.* Lecture Notes in Computer Science. Springer, Honolulu, Hawaii, USA, 65–76.

Florescu, D., Levy, A. Y., Suciu, D., and Yagoub, K. 1999. Optimization of run-time management of data intensive web-sites. In *Proceedings of the 25th International Conference on Very Large Databases.* Morgan Kauffman, Edinburgh, Scotland, 627–638.

Fraternali, P. 1999. Tools and approaches for developing data-intensive web applications: A survey. *ACM Computing Surveys 31,* 3, 227–263.

Fu, X., Bultan, T., , and Su, J. 2004. Analysis of interacting BPEL web services. In *Proceedings of the 13th International World Wide Web Conference (WWW).* online proceedings, New York (NY), USA.

Gomez, J., Cachero, C., and Pastor, O. 2001. Conceptual modeling of device-independent Web applications. *IEEE MultiMedia 8,* 2, 26–39.

Hull, R., Benedikt, M., Christophides, V., and Su, J. 2003. E-services:a look behind the curtain. In *Proceedings of the 22nd ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS).* ACM, San Diego (CA), USA.

Papakonstantinou, Y., Petropoulos, M., , and Vassalos, V. 2002. QURSED: querying and reporting semi-structured data. In *Proceedings of the 28th ACM SIGMOD International Conference on the Management of Data.* ACM, Madison (WI), USA, 199–203.

Rossi, G., Schwabe, D., and Lyardet, F. 1999. Improving Web systems with navigational patterns. In *Proceedings of the 8th International World Wide Web Conference.* Lecture Notes in Computer Science. Springer, Amsterdam, Holland, 1667–1678.

SOAP 2001. Simple object access protocol. Available at http://www.w3.org/TR/SOAP.

van der Aalst, W., Dumas, M., Hofstede, A., , and Wohed, P. 2002. Pattern-based analysis of BPML (and WSCI). QUT Technical report, FIT-TR-2002-05.

WebRatio 2004. The webratio tool suite. Available at http://www.webratio.com.

Wohed, P., van der Aalst, W., Dumas, M., and Hofstede, A. 2003. Analysis of web services composition languages: The case of BPEL4WS. In *Proceedings of the International Conference on Conceptual Modeling.* Lecture Notes in Computer Science. Springer, Chicago (IL), USA, 200–215.

WSCI 2002. Web service choreography interface. Available at http://wwws.sun.com/software/xml/developers/wsci.

WSCL 2002. The web service conversation language. Available at http://www.w3.org/TR/2002/NOTE-wscl10-20020314.

WSDL 2002. Web service description language. Available at http://www.w3.org/2002/ws/desc.

XSchema 2001. The XML schema language. Available at: http://www.w3.org/XML/Schema.

YAGOUB, K., FLORESCU, D., ISSARNY, V., AND VALDURIEZ, P. 2000. Caching strategies for data-intensive web sites. In *Proceedings of the 26th International Conference on Very Large Databases*. Morgan Kauffman, Cairo, Egypt, 188–199.

...