Bulletin of the Technical Committee on

Data Engineering

December 2002 Vol. 25 No. 4 **IEEE Computer Society**

Letters

Special Issue on Web Services

Conference and Journal Notices

The Bulletin of the Technical Committee on Data Engineering is published quarterly and is distributed to all TC members. Its scope includes the design, implementation, modelling, theory and application of database systems and their technology.

Letters, conference information, and news should be sent to the Editor-in-Chief. Papers for each issue are solicited by and should be sent to the Associate Editor responsible for the issue.

Opinions expressed in contributions are those of the authors and do not necessarily reflect the positions of the TC on Data Engineering, the IEEE Computer Society, or the authors' organizations.

Membership in the TC on Data Engineering is open to all current members of the IEEE Computer Society who are interested in database systems.

The Data Engineering Bulletin web page is http://www.research.microsoft.com/research/db/debull.

Letter from the Editor-in-Chief

Ten Years as Editor-in-Chief

In 1992, Rakesh Agrawal, then chair of the TC on Data Engineering, asked me to be the editor-in-chief of the Data Engineering Bulletin. I paused before accepting this offer. I knew that this was a substantial undertaking. But, if I can be permitted to ”toot my own horn” a bit, I think that the effort have been largely successful.

A good way of judging the credibility of the Bulletin is to see who has contributed toward its success. This space is too short to list them, but I am very gratified by the success of our editors in enlisting authors from the best academic and industrial settings of our field, leading practitioners all. However, the space is sufficient to list the associate editors who have served the Bulletin so well during my tenure as editor-in-chief. It is these editors who, issue after issue, plan and solicit the technical content. I am proud to have recruited these outstanding people to serve as editors: Daniel Barbara, Surajit Chaudhuri, Umeshwar Dayal, Amr El Abbadi, Ahmed Elmagarmid, Michael Franklin, Shahram Ghandeharizadeh, Johannes Gehrke, Goetz Grafe, Luis Gravano, Alon Halevy, Joe Hellerstein, Meichun Hsu, Yannis Ioannidis, Christian Jensen, Donald Kossmann, Renee Miller, Eliot Moss, Elke Rundensteiner, Betty Salzberg, Sunita Sarawagi, Gerhard Weikum, Kyu-Young Whang, and Jennifer Widom. In addition, Divyakant Agrawal, Fabio Casati, Sharma Chakravarthy, and Ron Obermarck served as single issue editors. Thank you all for consistently doing a fine job.

Finally, I want to thank people who have contributed in other ways. Mark Tuttle twice produced latex style files and prototype issue frameworks that enable us to generate the Bulletin both electronically and in hardcopy. The Bulletin could not be published in its current form without Mark's efforts. Rakesh Agrawal, Betty Salzberg, and Erich Neuhold, as TCDE chairs, provided unwavering support, and have worked to assure that the Bulletin can continue to be successfully published. The IEEE Computer Society staff has been helpful in many respects, especially in providing membership lists and coordinating the hardcopy publication of the Bulletin.

The Current Issue

Web services are rapidly becoming a part of our collective landscape. We needn't wonder whether we can purchase something on-line. The answer is clearly “yes” almost all the time. B2C e-commerce is florishing. Certain forms of B2B e-commerce are likewise commonplace. Less common, however, is the integration of web services across enterprises, programs interacting with programs as opposed to programs interacting with people. Further, there are many potential ”gotcha's” to deal with, security, privacy, reliability, availability, etc.. Also, robust web services have been notoriously difficult to implement, with problems including the preceding ”ities”, simplifying application programming, providing standards for metadata and protocols, and integrating specific applications with generic web services provided by other vendors.

This issue discusses many of the problems associated with web services and their implementation and exploitation. As is characteristic of articles by technical professionals, be they product developers or researchers, the emphasis is on frameworks, architectures and generic approaches as opposed to specific web service instances. This should not be a surprise given that the authors are from universities or software providers, not application specialists. And, indeed, the real technical leverage is frequently via such a generic focus. Nonetheless, the current issue ends with examples of useful web services. Fabio Casati, as guest editor, and Umesh Dayal have thus captured a broad snapshot of the current state of web services. Bringing this information together in one place and in a timely fashion is a significant contribution to the dissemination of this technology. So I want to thank Fabio and Umesh for their thorough and timely editorial work. This area, an evolution of transaction processing, can be expected to remain important for years. The current issue should thus be a worthwhile source of information for a long time.

David Lomet
Microsoft Corporation

Letter from the Special Issue Editors

Web services are emerging as the paradigm and the technology of choice for integrating applications on the Web, both within and across enterprises. They represent the natural evolution of integration technologies, from being focused on interoperability among objects residing on the same LAN to supporting interaction of components at coarser granularity and accessed through a WAN, possibly across firewalls. The unprecedented agreements of virtually all major software vendors have made it possible to standardize at least the basics aspects of service description, discovery, and interaction, so that now the majority of application servers support the same concepts and the same basic standards. This appears to pave the way for a successful evolution of this still relatively young area. Indeed, most industry analyst and IT professional believe that Web services will be a driving force in e-business, although their adoption rate is slower than what most people initially predicted.

Besides their appeal from a business perspective, Web services raise a number of interesting challenges from a research point of view. It is not by chance that many academics are also very active in this area. This special issue includes several papers, from both the industry and the academia, that frame the notion of Web services and discuss problems, architectures, and applications of this novel technology. Rather than presenting solutions, the goal of the special issue is to provide an overview of ongoing research projects in this very dynamic area.

The issue starts with a paper by Gustavo Alonso, providing a critical discussion on what is the truth behind common beliefs about Web services. Then, Asuman Dogac et al. address an important and largely unsolved issue: that of describing the semantics of Web services.

The next set of papers focus on languages and infrastructures for Web services. They present the perspectives of major Web services software vendors, e-commerce enablers, and academic research groups. This central part of the special issue starts with a paper by Mike Carey, Michael Blevins, and Pal Takacsi-Nagy of BEA, describing the capabilities of WebLogic 8.0. Next, David Burdett, Qiming Chen, and Meichun Hsu of CommerceOne show how Conductor extends current Web service technology with features supporting secure interaction and collaborative process execution. Roger Barga of Microsoft presents the Phoenix project, whose goal is that of enabling transparent application recovery, freeing developers from the need of writing code for this purpose. The series of industrial infrastructure papers is concluded by a paper describing work done at HP labs on applying business intelligence techniques to managing e-business operations.

These industrial papers are followed by four contributions coming from the academia, all addressing the problem of service composition, although from different angles and in different contexts. First, Enzo Colombo et al. present VISPO, a research project addressing the dynamic composition of services and its applicability in the context of cooperative processes. The paper by Mike Papazoglou et al. combines Web services and artificial intelligence techniques into a framework for specifying services of interest, possibly to be executed by aggregating other services. Next, the contribution by Boualem Benatallah et al. introduces SELF-SERV, a platform for the specification and execution of composite services in a centralized or peer to peer fashion. Finally, Marco Brambilla et al. show a framework for designing and extending Web applications with support for Web services and service composition.

The special issues concludes with two papers describing interesting applications of Web service technology: the first one, by Jens Graupmann and Gerhard Weikum of the University of the Saarland, describes how Web services can be used to support information retrieval that goes beyond portal boundaries. The second, by Mariano Cilia and Alex Buchmann of TU-Darmstadt, shows how Web and active database technology can be used for providing services to car drivers.

We hope that the reader will enjoy this special issue, and we thank all the authors for their excellent contributions.

Fabio Casati and Umeshwar Dayal
Hewlett-Packard
Palo Alto, CA, USA

Myths around Web Services

Gustavo Alonso
Department of Computer Science
Swiss Federal Institute of Technology (ETHZ), Switzerland
www.inf.ethz.ch/department/IS/iks/

1 Introduction

Web services and the technology surrounding them have become the dominant trend in the electronic commerce arena. XML, SOAP, UDDI, and WSDL, as the foundation of Web services, are all attracting considerable attention as potential bridges between heterogeneous systems distributed across the Internet. The assumption seems to be that soon most applications will speak and understand XML, that all systems will support SOAP, that everybody will advertise their services in UDDI registers, and that all services will be described in WSDL. Once that stage is reached, application integration and business to business (B2B) e-commerce will be straightforward.

Unfortunately, Web services are only the next step in the natural evolution of middleware. Therefore, by design, Web services are evolutionary rather than revolutionary. The most basic form of middleware are RPC engines. When such engines become transactional, they become TP-Monitors. Once object orientation aspects are included, TP-Monitors evolve into Object Monitors. Message oriented middleware (MOM) also originated from TP-Monitors since persistent queuing was a feature of many TP-Monitors until they became systems on their own. In fact, many MOM platforms are RPC based. Web services are, primarily, an extension to middleware platforms to allow them to interact across the Internet. Only from this perspective do many of the developments in the Web services arena make sense, e.g., that one of the first protocols to be wrapped as SOAP messages was RPC (and, at the time of writing, almost the only one to have been completely specified).

Of course, it is possible that Web services will trigger a radical change in the way we think about middleware, application integration or the way we use the Internet. In that case, Web services will evolve into something yet unforeseen. At this stage, however, this has yet to happen. In reality, and precisely because they were created with that purpose in mind, Web services are used today almost exclusively for conventional enterprise application integration (which may or may not happen in a B2B setting). It is this experience as an extension to middleware platforms that will define and shape Web services in the short and medium term.

There are nevertheless many proposals that take Web services well beyond their current capabilities: semantic web, dynamic marketplaces, automatic generation of B2B applications, seamless integration of IT infrastructures from different corporations, etc. These proposals are the basis for presenting Web services as revolutionary rather than evolutionary. Such speculations are the province of long term research but they tend to ignore the exact nature of Web services and the underlying technology. Many of these ideas also ignore the current limitations of existing middleware platforms although most of these limitations appear again at the Web service level. In this regard, Web services have to a certain extent become an outlet for ideas that proved impractical in the

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

past. The result is a series of myths around Web services that make Web services, already quite a complex set of ideas *per se*, even more difficult to comprehend, understand, and analyze.

In this paper we discuss some of these myths and try to clarify how much truth there is in them. The goal is to bring to the fore a few fundamental aspects of Web services and discuss in detail their practical implications. These considerations can be seen as both a warning against expecting too much from Web services as well as directions for future research since they are all open problems that remain to be solved before Web services can be used in their full generality. The overall message is that, with the exception of standardization efforts (already a significant step forward), today's Web services may have little to offer over conventional middleware.

2 A quick overview of Web services

Web services are a series of specifications around a very generic architecture [Kr10]. This architecture has three components: the service requester, the service provider, and the service registry, thereby closely following a client/server model with an explicit name and directory service (the service registry). Albeit simple, such an architecture illustrates quite well the original purpose of UDDI, WSDL and SOAP. In all cases, the information managed by these specifications is in the form of XML documents.

UDDI The service registry is based on the UDDI specification (Universal Description, Discovery, and Integration). The specification defines how to interact with a registry and what the entries on that registry look like. Interactions are of two types: registration and lookup. Registration is the procedure whereby new service descriptions are added to the registry. Lookup corresponds to queries sent by service requesters in search for the right services. The entries contain three types of information: white, yellow and green pages. The white pages contain generic information about the service provider (e.g., address, contact person, etc.). The yellow pages include categorization information that allows the registry to classify the service (e.g., flight reservation, search engine, or bookstore). The green pages contain information about the services interface and pointers to the service provider (where the actual WSDL interface definition can be found).

There are already several UDDI registries maintained by software vendors. These public registries are meant as low level, generic systems supporting only the most basic of interactions. The underlying idea is that more sophisticated repositories (e.g., with advanced query capabilities) will be built on top of UDDI repositories. Such *service databases* are, however, not part of the specification. UDDI also describes how to interact with a repository using SOAP. Such support is intended not so much for dynamic binding to services (in the middleware sense) as for developers building advanced service databases and other applications on top of UDDI repositories. Finally, there are two types of UDDI registries: public and private. Public ones are accessible to everyone and play the role of open search engines for Web services. Private ones are those that companies or group of companies create for their own use. For obvious reasons, industrial strength Web service implementations are likely to be based on private repositories rather than on public ones. It remains to be seen to what extent private repositories use UDDI as much of its functionality is not needed for private use.

WSDL The interface to a Web service is defined using WSDL (Web Services Definition Language). By using WSDL, designers specify the *type* system used in the description, the *messages* necessary to invoke an operation of the service (and their format), the *operation* protocol (whether it returns a response, etc.), the *port type* or set of operations that conform an instance of a service, and the binding or actual protocol to be used to invoke the operations of an instance of a service (e.g., HTTP). Note that what is known as *service* is a logical unit encompassing all port types mapped to the same logical service (e.g., flight reservations through RPC or through e-mail, each one of them being a port type of the flight reservation service).

SOAP Interaction between requester, provider and registry happen through SOAP (Simple Object Access Protocol). SOAP specifies messages as documents encoded in XML divided into two parts: header and body. Both the header and the body can be subdivided into blocks. Header blocks carry information pertaining the interaction: e.g., security, authentication, transactional context, etc. Body blocks store the data used in the interaction, e.g., which procedure is being called, each individual parameter, etc. SOAP also defines bindings to actual transport protocols. A binding specifies how a SOAP message is transmitted using, e.g., HTTP.

SOAP can be best understood when it is considered as the specification of a protocol wrapper rather than a communication protocol itself. The main point of SOAP is to provide a standardized way to transform different protocols and interaction mechanisms into XML documents. As such, each concrete protocol needs a SOAP specification. An example is the specification of how to use RPC over HTTP. The specification describes how to encode an RPC invocation into an XML document and how to transmit the XML document using HTTP.

The syntax for these specifications is based on XML. Nevertheless, WSDL, and SOAP support alternative type systems. In all cases, WSDL and SOAP should be seen as templates rather than strict standards. The fact that two Web services use WSDL and SOAP does not immediately make them compatible. One of them could be an RPC service accessible through HTTP. The other could be a batch service accessible through e-mail and using EDIFACT. Both are Web services compliant with accepted specifications and yet perfectly incompatible.

3 Myth I: Web services and standards

The hype around UDDI, WSDL and SOAP has eclipsed many parallel (and previous) efforts along the same direction. As a result, there is an obvious trend towards systems that are UDDI, WSDL, and SOAP specific. Such trend thrives on the myth that Web services are an accepted and dominant standard. However, it is by no means clear that Web services will displace existing technologies. Christoph Bussler has pointed out the fact that *standard* no longer means *globally unique* in the B2B world. He predicts that, in addition to UDDI, WSDL and SOAP, up to a few hundred competing B2B standards may coexist [Bu01]. Examples of such established standards that will not simple go away are the Electronic Data Interchange (EDI) [EDI], used in manufacturing, and SWIFT [SWI], used in the financial world. Consequently, Bussler recommends developers to be agnostic towards B2B standards and has also shown how the architecture of such generic systems should look like [Bu02].

Generality is certainly a solution to the lack of standardization. If no standard dominates, a generic architecture can be easily adapted to whatever specification comes along. Unfortunately, generality comes at a price and undermines the standardization efforts. The reason is that, in practice, Web services are not being built from scratch. They are being built on top of existing multi-tier systems, systems that are all but general. Hence, many Web services are biased from the start towards specific protocols, representations, and standards, i.e., those already supported by the underlying middleware. The necessary generality will only be achieved, if at all, by yet additional software layers. Even the WSDL specification has allowed for such generalization by providing alternative entry points to a given Web service (the *port types* in the WSDL jargon). What is then wrong with this picture? Michael Stonebraker argues that there is already too much middleware with competing and overlapping functionality [St02]. He defends the need for integrating middleware functionality in just a few system types, namely data federation systems, enterprise application integration (EAI) and application servers. Otherwise, the sheer complexity of the IT infrastructure becomes unmanageable. This is an argument often repeated in the middleware arena [Be96, AM97], where real convergence was starting to take place. Web services, however, add new layers to the already overly complex multi-tier architecture typical of B2B interactions. Aiming for generic systems will make matters even worse. Translation to and from XML, tunneling of RPC through SOAP, clients embedded in Web servers, alternative port types, and many of the technologies typical of Web services do not come for free. They add significant performance overheads and increase the already extreme complexity of developing, tuning, maintaining and evolving multi-tier systems.

Taken together, these two concerns configure a difficult dilemma for Web services. The proliferation of competing standards, whether based on the same syntax (XML) or not, will require additional software layers to address interoperability problems. Even in those cases where a single set of standards can be used, web services are being almost universally built as additional tiers over existing middleware platforms. Unfortunately, multi-tier architectures are already too complex and cumbersome. Adding more layers will not make them any better and the sheer complexity and cost of such systems may prevent the widespread use of the technology. Without widespread use, standards will fragment even further, thereby making it almost impossible to produce generic enough platforms which, in turn, increases again the development and maintenance costs. The resulting vicious circle can be the Achilles' heel of Web services as there is no obvious way out of it.

4 Myth II: Web services in conventional applications

One of the drawbacks of Web technology is that it is still too tightly related to humans and browsers. Web services have computers as their main users and are not based on browsers at all. Nevertheless, many of us still think about Web services in the same terms we think about a Web browser: our first image of a web service is that of an interactive one. Maybe with the execution driven by a computer instead of a human but interactive nonetheless. The examples available in the literature, and not only in the research literature, corroborate this bias. We have all seen many different variations of the traveling planner service, which has been misused so often that it should become a standard on its own. Flight reservations, car rental and hotel booking, or buying a travel guide, are all examples of interactive services. Moreover, all these services are typical Business to Consumer (B2C) interactions, rather than B2B exchanges. This is an interesting development since Web services are being pursued because of their potential impact on B2B not on B2C.

There are of course practical advantages in using Web services interactively and on-line. One example often mentioned are applications that embed a search engine by using Web services [Sh02]. Other examples are applications or operating systems that send periodic bug reports to the software vendor using a Web service, applications that automatically download and install patches, or systems that use a remote service to provide functionality that cannot be provided locally (e.g., access to a very large database that is not locally available). These are all very appealing scenarios but it is not immediately obvious that Web services are the best way to implement them. In some cases (e.g., information flow from the application to a server), this functionality is already being provided without Web services and it is not clear that switching to Web services will bring any significant advantages. In other cases, it does not seem reasonable to bloat the application with the whole machinery of Web services to implement just a fancy feature. If the operating system eventually provides support for accessing Web services to all applications, then this may make sense but we are quite far from that stage. Perhaps an even more decisive factor is that many of the features of Web services are irrelevant in these settings. For instance, application specific information does not need to be sent as an XML document. Likewise, interfaces used internally by a software vendor do not need to be described using WSDL (and certainly do not need advertising using UDDI).

From a practical perspective, it is also not clear how to build applications that rely on Web services for part of their functionality. The problem has been recently analyzed by Clay Shirky [Sh02]. He points out that Web services are still *plumbing for the exchange of XML documents using SOAP*. For interactive and on-line use within applications, he identifies several crucial issues that remain unsolved. One of them is trust: how far can the application trust and rely on external Web services which it does not control? Another one is the fact that we do not yet understand the impact of Web services on software design as many of the techniques for component based software development do not work with Web services. Answers to these questions are needed before Web services are widely used as extensions to conventional applications. As Shirky also points out, it will not be a trivial endeavor. It will require a whole new software engineering philosophy and tools that will not be available any time soon.

5 Myth III: Direct connectivity across corporate boundaries

Another myth resulting from ignoring the complexities of application integration and software design at a large scale is the claim that Web services provide a direct link between middleware platforms of different corporations. Most conventional middleware platforms are implemented on top of RPC: TP-Monitors, Object Monitors, CORBA implementations, and even message oriented middleware. Because of its pervasiveness, RPC over HTTP was one of the first interaction mechanisms specified using SOAP. By doing so, a Web service becomes an extension of existing multi-tier architectures but with the client residing now at the other side of the firewall and behind a Web server. Since B2B services are implemented using multi-tier systems, being able to use RPC through SOAP is seen by many as a gateway to interconnect the IT infrastructure of different companies.

There are several problems with such an interpretation. One is that RPC results in a tight integration that makes components dependent on each other. This is unacceptable in any industrial strength setting, specially if the components belong to different companies. Not only would the complexity of the resulting system increase exponentially, the mere act of maintaining the system would become a coordination nightmare with tremendous costs. This is why the vast majority of B2B interactions happen asynchronously and in batch mode, not interactively. Rather than direct invocations, requests are batched and routed through queues. Responses are treated in the same way. The actual elements of the interaction (client and server, to simplify things) are kept as decoupled as possible so that they can be designed, maintained, and evolved independently of each other. Systems based on EDI and SWIFT are, again, good examples of the typical loosely coupled architectures of B2B systems.

Proof of this is the strong trend towards asynchronous SOAP. The fact that the most widespread use of SOAP is to tunnel RPC does not contradict this statement. Many queuing systems are implemented on top of RPC. A message is placed on a queue and a daemon makes an RPC call to another remote daemon that takes the message and places it on the receiving queue. Technically this is not only possible, it is a reasonable way of implementing B2B interactions. From the point of view of Web services, however, it means that the Web service description will be far more complex than an RPC invocation encoded as an XML message. The description may have more to do with the interaction mechanism (the queues) than with the service interface itself. In fact, in many cases, the actual service interface will not necessarily be made explicit. For instance, a service may simply indicate that it is a queue that accepts EDIFACT purchase order messages without describing such messages (since their format is already known to those using them).

6 Myth IV: UDDI and dynamic binding

An UDDI registry is conceptually similar to a name and directory server. There are, however, significant practical differences between the two, differences that tend to be ignored and lead to the assumption that an UDDI registry has the same purpose as a name and directory server. The result is the widespread assumption that dynamic binding will be a common way of working with Web services. This is far from being the case and there are two very strong arguments against this assumption.

From the point of view of functionality, UDDI registries have been created as standardized catalogues of Web services. The information they contain is intended for humans, not for computers. First, there is the problem of the semantic interpretation of the parameters and operations defined by the interface. These parameters indicate the expected type but not what the parameter actually means (e.g., a price is given as an integer but there might not be any indication of the currency used). There is also the issue of how to deal with exceptions and how to link them to the internal business processes. The service might also provide different ways to proceed depending on the outcome of intermediate operations. Only a person can make sense of this information while using it will require careful analysis and a significant design effort. Second, interactions between different companies are regulated by contracts and business agreements. Without a proper contract, not many companies will interact with each other. To think that companies will (or can) invoke the first Web service they find on the network is unrealistic. Web service based B2B systems will be built through specialists who locate the necessary services, identify the interfaces, draw the necessary business agreement, and then design and build the actual application with the Web service either *hardwired* into the application code or defined as a deployment parameter.

From the software engineering point of view, dynamic binding is a double edge sword. If dynamic binding is used simply to determine the location of a well defined service, it is indeed an useful feature. Any other form of dynamic binding makes it almost impossible to develop real applications. CORBA already provided designers with very fancy dynamic binding capabilities. An object could actually query for a service it had never heard of and build on the fly a call to that service. Such level of dynamism makes sense only (if at all) in very concrete, low level scenarios that appear almost exclusively when writing system software. Application designers have no use for such dynamic binding capabilities. How can one write a solid application without knowing what components will be called? It is nearly impossible to write sensible, reliable application logic without knowing what exceptions might be raised, what components will be used, what parameters these components take, etc. In its full generality, dynamic binding does not make sense at the application level and this also holds for Web services. In regard to dynamic binding as a fault tolerance and load balancing mechanism, in the context of Web services, the UDDI registry is simply the wrong place for it. UDDI has been designed neither with the response time capabilities, not the facilities necessary to support such dynamic binding. Moreover, the UDDI registry cannot do any load balancing nor any automatic fail over to a different URI in case of failures. It is simply not designed to do that. Such problems are to be solved at the level of individual Web service provider using known techniques like replication, server clustering, and hot-backup techniques.

UDDI registries will, thus, be used by programs only to the extent that service publishing will be automatic in many systems and search over an UDDI registry will happen through specialized added-value tools built on top the UDDI registries.

7 Myth V: all data will be in XML

XML is a blessing as a syntax standard. It allows to build generic parsers that can be used in multitude of applications, thereby ensuring robustness and low cost for the technology. Unfortunately, this significant advantage does not compensate for the fact that XML is a performance nightmare. There are also many data types that do not get along well with XML. e.g., anything that is binary or nested XML documents [Sa02]. In many cases, even if it is possible, there is no point in formating the application data as an XML document. We have already mentioned an example: a Web service implemented as a queue expecting EDIFACT e-mail messages does not gain much by having the message encoded in XML. In fact, it only loses performance and introduces unnecessary software layers.

XML encoding makes sense when linking completely heterogeneous systems or passing data around that cannot be immediately interpreted. It also makes sense when there is no other standard syntax and designers must choose one. When Web services are built based on already agreed upon data formats, then the role of XML is reduced to be the syntax of the SOAP messages involved. This is why there is such a strong demand for SOAP to support a binary or *blob* type. There are several ways of doing this [Sa02]: using URLs as pointers, as an attachment or with the recently proposed Direct Internet Message Encapsulation (DIME) protocol [Ni02]. Whatever mechanism becomes the norm, expect an increasing amount of Web service traffic to contain binary rather than XML data.

The use of binary rather than XML for formatting application data has a wide range of implications for Web services. First, it will provide a vehicle for vertical B2B standards to survive even if Web service related specifications become dominant. This is directly related to the discussion above on Myth I. In practice, Web services become just a mechanism to tunnel interactions through the Internet, their original intended goal. The actual interaction semantics will be supported by other standards, those use to encode the data in binary (e.g., once more, EDI or SWIFT). The question will then be whether Web services provide enough added value to justify the overhead. Second, related to Myth III, Web services implemented over binary data will describe only the interaction. They cannot specify the actual programmatic interface of the service as this is hidden in the binary document and, therefore, cannot be controlled by the Web services infrastructure. This will reduce even further the chances of having tightly coupled architectures built around Web services. Finally, related to Myth IV, Web services based on binary formats will increase the dependency on humans for binding to services as much of the information needed to bind to a service might be external to the Web service specification.

8 Conclusions

After the burst of the Internet bubble, several critical voices have been raised against the hype around e-commerce, e.g., [Co02]. Web services are, to a great extent, still riding that hype and generating their own. To keep things in the proper perspective, it is useful to understand the original purpose of a given technology. This does not prevent the technology from becoming revolutionary, but it helps to identify those that are merely evolutionary. Web services are, at the current stage, only a natural evolutionary step from conventional application integration platforms. In spite of the many grand ideas being proposed in both industry and academia, there is a fair chance that market forces and sheer practicality will keep the role of Web services to, indeed, mere plumbing for B2B exchanges.

Acknowledgments

Part of this work is supported by grants from the Hasler Foundation (DISC Project No. 1820) and the Swiss Federal Office for Education and Science (ADAPT, BBW Project No. 02.0254 / EU IST-2001-37126).

References

[AM97] G. Alonso, C. Mohan. WFMS: The Next generation of Distributed processing Tools. In: Advanced Transaction Models and Architectures. S. Jajodia and L. Kerschberg (Eds.). Kluwer Academic Publishers, 1997.

[Be96] Philip A. Bernstein. Middleware: A Model for Distributed System Services. CACM, Vol. 39 No. 2, Feb 1996.

[Bu01] C. Bussler. B2B Protocol Standards and their Role in Semantic B2B Integration Engines. IEEE Data Engineering Bulletin, Vol 24. No. 1, March 2001.

[Bu02] C. Bussler. The Role of B2B Engines in B2B Integration Architectures. Sigmod Record, Vol. 31 No. 1, March 2002

[Co02] T. Coltman et al. Keeping E-business in perspective. CACM, Vol. 45, No. 8, August 2002.

[EDI] United Nations Directories for Electronic Data Interchange for Administration, Commerce and Transport. http://www.unece.org/trade/untdid/welcome.htm

[Kr10] H. Kreger. Web Services Conceptual Architecture (WSCA 1.0). IBM. Available from: *http://www-4.ibm.com/software/solutions/webservices/pdf/WSCA.pdf*

[Ni02] Nielsen H. F. et al. Direct Internet Message Encapsulation (DIME). Internet draft, draft-nielsen-dime-02, June, 2002.

[Sa02] R. Salz. Transporting Binary Data in SOAP. Published on XML.com http://www.xml.com/pub/a/2002/08/28/endpoints.html

[Sh02] C. Shirky. Web Services and Context Horizons. IEEE Computer, Vol. 35 No. 9, September 2002.

[St02] Michael Stonebraker. Too Much Middleware. ACM Sigmod Record, Vol. 31 No. 1, March 2002

[SWI] S.W.I.F.T. SCRL. http://www.swift.com/

Exploiting Web Service Semantics: Taxonomies vs. Ontologies

Asuman Dogac, Gokce Laleci, Yildiray Kabak, Ibrahim Cingil

Software Research and Development Center
Middle East Technical University (METU)
06531 Ankara Turkiye
email: asuman@srdc.metu.edu.tr

Abstract

Comprehensive semantic descriptions of Web services are essential to exploit them in their full potential, that is, discovering them dynamically, and enabling automated service negotiation, composition and monitoring. The semantic mechanisms currently available in service registries which are based on taxonomies fail to provide the means to achieve this. Although the terms "taxonomy" and "ontology" are sometimes used interchangably there is a critical difference. A taxonomy indicates only class/subclass relationship whereas an ontology describes a domain completely. The essential mechanisms that ontology languages provide include their formal specification (which allows them to be queried) and their ability to define properties of classes. Through properties very accurate descriptions of services can be defined and services can be related to other services or resources.

In this paper, we discuss the advantages of describing service semantics through ontology languages and describe how to relate the semantics defined with the services advertised in service registries like UDDI and ebXML.

1 Introduction

When looking towards the future of web-services, it is predicted that the breakthrough will come when the software agents start using web-services rather than the users who need to browse to discover the services. Currently well accepted standards like Web Services Description Language [WSDL] and Simple Object Access Protocol [SOAP] make it possible only to "dynamically access" to Web services in an application. That is, when the service to be used is known, its WSDL description can be accessed by a program which uses the information in the WSDL description like the interface, binding and operations to dynamically access the service. However to dynamically *discover* services, say through software agents require detailed semantic information about the services to be available.

Currently, a number of taxonomies are being used to discover services in service registries like [UDDI] or [ebXML]. The most widely used taxonomies are North American Industrial Classification Scheme [NAICS] for associating services with "industry" semantics; Universal Standard Products and Services Classification [UNSPSC] for classifying product/services and [ISO 3166] for locale.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

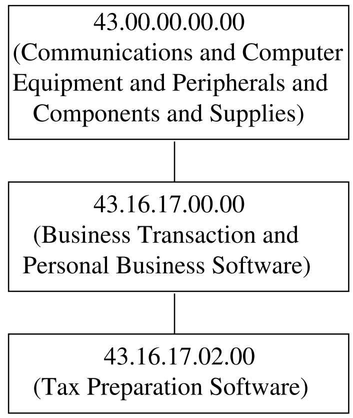

Figure 1: A part of an UNSPSC Taxonomy

A taxonomy is a hierarchy and a unique code is usually assigned to each node of the hierarchy. This code also encodes its path. For example, as shown in Figure 1, UNSPSC code for *Communications and Computer Equipment and Peripherals and Components and Supplies* is "43.00.00.00.00". One of the classes under this class is *Business transaction and personal business software* whose code is "43.16.17.00.00". Going one more level deep,"43.16.17.02.00" is the code for *Tax preparation software*.

By relating a service with the codes in such taxonomies, it is possible to give the service a certain amount of semantics. For example, when a service uses the code "43.16.17.02.00" to describe its semantics, we understand that the service is about "tax preparation software". Therefore, a user looking for a service related with a "tax preparation software" can search the service registries with the corresponding UNSPSC code to obtain all services that have declared themselves to be related with this UNSPSC code. Note however that these services could be anything; they may be "selling" such software; they may be providing it as a service. Furthermore the service may have any number of properties like certain service qualities, say, minimum and maximum service delivery times; may require certain payment obligations, say, advance credit card payment. Additionally a service may be available at a discount price when it is used in aggregation with the other services that the company provides. There can be several such properties of a service and hence the user has to go through the services found to manually pick the service that satisfies her requirements. Notice for example that, in UDDI registries, this information can only be made available informally (i.e., with no formal semantics) through "OverviewDocs" or "OverviewURLs" of the tModels associated with the services. In short, taxonomies do not help in this respect.

It follows that to exploit the Web services to their full potential we need more powerful tools, that is, ontologies to describe their semantics. In fact, currently, describing the semantic of Web in general [Berners-Lee], and semantic of Web services in particular are very active research areas. There are a number of efforts for describing the semantics of Web services such as [McIlraith a, McIlraith b, Denker]. Among these [DAML-S] defines an upper ontology, that is, a generic "Service" class. In order to make use of DAML-S upper ontology, the lower levels of the ontology need to be defined. To provide interoperability, application domains must share such specifications. In fact, an ontology describes *consensual knowledge*, that is, it describes meaning which has been accepted by a group not by a single individual. Standard bodies need to define domain specific ontologies.

In this paper, we discuss the advantages of defining service semantics through ontology languages. We further note that, once the semantic is defined, it is also necessary to relate the defined semantics with the services advertised in the service registry. Therefore we describe how this can be achieved in UDDI and ebXML registries.

The paper is organized as follows: Section 2 briefly introduces DAML-S upper ontology and describes the advantages of ontology languages over taxonomies. Section 3 describes mechanisms provided by UDDI and ebXML registries for associating semantics with the services advertised.

2 Service Semantics through Ontology Languages

Web services, like their real life counterparts, may have many properties. The aim of this section is to demonstrate that all the necessary properties of services can easily be defined through an ontology language. While developing domain specific ontologies, it is a good idea to ground them in upper ontologies since in this way they are more consistent and it becomes easier to integrate them within distributed heterogeneous systems.

DAML-S provides such an upper ontology, that is, it defines a top level "Service" class with some generic properties common to most of the services. The "Service" class has the following three properties:

- *presents:* The range of this property is *ServiceProfile* class. That is, the class Service *presents* a *ServiceProfile* to specify what the service provides for its users as well as what the service requires from its users.

- *describedBy:* The range of this property is *ServiceModel* class. That is, the class Service is *describedBy* a *ServiceModel* to specify how it works.

- *supports:* The range of this property is *ServiceGrounding*. That is, the class Service *supports* a *ServiceGrounding* to specify how it is used.

DAML-S is based on [DAML+OIL] and DAML+OIL allows very sophisticated ontologies to be defined and queried through, for example, DAML APIs. The queries on ontologies usually access the properties of classes or traverse the ontology. Hence it is possible to standardize queries to facilitate their use in an automated way.

In the following we provide examples for some of the properties of DAML-S Service Profile:

- *serviceParameters:* In DAML-S, service parameters denote an expandable list of RDF properties that may accompany a profile description. The range of each property is unconstrained, i.e. no range restrictions are placed on the service parameters as shown in the following:

```
<daml:Property rdf:ID="serviceParameter">
 <daml:domain rdf:resource="&service;#ServiceProfile"/>
 <daml:range rdf:resource="http://www.daml.org/2001/03/daml+oil#Thing"/>
</daml:Property>
```

- *degreeOfQuality:* This property of Service Profiles provide qualifications about the service.

```
<daml:Property rdf:ID="degreeOfQuality">
 <daml:domain rdf:resource="&service;#ServiceProfile"/>
 <daml:range rdf:resource="http://www.daml.org/2001/03/daml+oil#Thing"/>
</daml:Property>
```

2.1 An Example Service Ontology

In this section we define an example ontology grounded in DAML-S for tax services for the sole purpose of describing the power of ontology languages. A detailed description of several properties of services including the methods of charging and payment, the channels by which the service is requested and provided, constraints on temporal and spatial availaility, service quality, security, trust and rights attached to a service, is given in [O'Sullivan]. Through the example ontology we show how some of these properties can be defined.

As shown in Figure 2, the top level class of this ontology is "TaxServices" which inherits from DAML-S "Service" class. In this way, several properties of the "TaxServices" class are conveniently defined by inheriting from the properties of the DAML-S "Service" class. For example, "paymentMethod" is defined as a subproperty

```
<!DOCTYPE uridef[
 <!ENTITY rdf     "http://www.w3.org/1999/02/22-rdf-syntax-ns">
 <!ENTITY rdfs    "http://www.w3.org/2000/01/rdf-schema">
 <!ENTITY xsd     "http://www.w3.org/2000/10/XMLSchema">
 <!ENTITY daml    "http://www.daml.org/2001/03/daml+oil">
 <!ENTITY profile "http://www.daml.org/services/daml-s/2001/05/Profile.daml">
 <!ENTITY service "http://www.daml.org/services/daml-s/2001/05/Service.daml">
 <!ENTITY tp      "http://www.srdc.metu.edu.tr/2002/10/TaxPayment.daml">
 <!ENTITY unspsc  "http://www.eccma.org/unspsc/browse/43.html">

<rdf:RDF
     xmlns:rdf =     "&rdf;#"
     xmlns:rdfs =    "&rdfs;#"
     xmlns:xsd =     "&xsd;#"
     xmlns:daml =    "&daml;#"
     xmlns:profile = "&profile;#"
     xmlns:service = "&service;#"
     xmlns:unspsc =  "&unspsc;#"
     xmlns:tp =      "&tp;#"

<daml:Ontology rdf:about=" ">
  <daml:imports rdf:resource="http://www.daml.org/2001/03/daml+oil"/>
  <daml:imports rdf:resource="&service;">  </daml:Ontology>

<daml:Class rdf:ID="TaxServices">
  <rdfs:subClassOf rdf:resource="&service;"/> </daml:Class>

<rdf:Property rdf:ID="paymentMethod">
   <rdfs:subPropertyOf rdf:resource=&profile;serviceParameter/>
   <rdfs:domain rdf:resource="&service;#ServiceProfile"/>
   <rdfs:range rdf:resource="&daml;#Thing"/> </rdf:Property>

<rdf:Property rdf:ID="serviceGuarantee">
   <rdfs:subPropertyOf rdf:resource=&profile;degreeOfQuality/>
   <rdfs:domain rdf:resource="&service;#ServiceProfile"/>
   <rdfs:range rdf:resource="&daml;#Thing"/> </rdf:Property>

<rdf:Property rdf:ID="requiredService">
  <rdfs:domain rdf:resource="&damls;#Service/>
  <rdfs:range rdf:resource="&damls;#Service/> </rdf:Property>

<rdf:Property rdf:ID="discountAmount">
  <rdfs:domain rdf:resource="#PromotionRequirements"/>
  <rdfs:range rdf:resource="&xsd;#nonNegativeInteger"/> </rdf:Property>

<daml:Class rdf:ID="TaxPreparationService">
  <rdfs:subClassOf rdf:resource="#TaxServices"/>
  <rdfs:label> Tax Preparation Service </rdfs:label>
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#promotion"/>
      <daml:toClass rdf:resource="#PromotionRequirements"/>
    </daml:Restriction> </rdfs:subClassOf> </daml:Class>

<rdf:Property rdf:ID="promotion">
   <rdfs:subPropertyOf rdf:resource=&profile;serviceParameter/>
   <rdfs:domain rdf:resorce="#TaxPreparationService"/>
   <rdfs:range rdf:resource="#PromotionRequirements"/> </rdf:Property>

<daml:Class rdf:ID="PromotionRequirements">
   <rdfs:subClassOf rdf:resource="#TaxServices"/>
   <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#requiredService"/>
       <daml:toClass rdf:resource="&damls;#Service"/>
    </daml:Restriction> </rdfs:subClassOf> </daml:Class>

<daml:Class rdf:ID="LegalConsulting">
  <rdfs:subClassOf rdf:resource="#TaxServices"/>
  <rdfs:label> Legal Tax Consultancy Service </rdfs:label>
</rdfs:subClassOf> </daml:Class>

<rdf:Property rdf:ID="serviceCodeUNSPSC">
  <rdfs:label> Defining a property to denote UNSPSC codes of services</rdfs:label>
  <rdfs:domain rdf:resource="&damls;#Service/>
  <rdfs:range rdf:resource="&unspsc;#UNSPSCCodes"/> </rdf:Property>
</rdf:RDF>
```

Figure 2: An Example Ontology Description for Tax Payment Domain

of DAML-S “ServiceProfile serviceParameter” property and “serviceGuarantee” as a subproperty of DAML-S “ServiceProfile degreeOfQuality” property. Notice that, in this generic ontology, ranges of these properties are defined to be the most general class in a DAML+OIL ontology, that is, *daml+oil#Thing* class. This class is specialized when defining ontology instances as shown in Figure 3. This general ontology also states that a

```
<!DOCTYPE uridef[
 <!ENTITY rdf     "http://www.w3.org/1999/02/22-rdf-syntax-ns">
 <!ENTITY tp      "http://www.srdc.metu.edu.tr/2002/10/TaxPayment.daml">

<rdf:RDF
     xmlns:rdf =     "&rdf;#"
     xmlns:tp =      "&tp;#"

<tp:TaxPreparationService rdf:ID="TaxHeavenTaxPreparationService">
  <tp:serviceCodeUNSPSC>43.16.17.02.00 </tp:serviceCodeUNSPSC>
  <tp:paymentMethod> CreditCard </tp:paymentMethod>
  <tp:serviceQuarantee>24</tp:serviceQuarantee>
  <tp:promotion rdf:resource="#MyPromotionRequirements"/>
</tp:TaxPreparationService>

<tp:PromotionRequirements rdf:ID="MyPromotionRequirements">
  <tp:requiredService rdf:resource="#LegalConfort"/>
  <tp:discountAmount>10 </tp:discountAmount>
</tp:PromotionRequirements>

<tp:LegalConsulting rdf:ID="LegalConfort">
</tp:LegalConsulting>
</rdf:RDF>
```

Figure 3: Service Ontology of the company "TaxHeaven"

"promotion" property can be associated with "TaxPrepatationService" which requires another service to be used and specifies the discount amount when the two services are used in aggregation. Finally, the ontology states that the UNSPSC codes may be associated with services.

Figure 3 shows a specific instance of the generic ontology given in Figure 2, for the company "TaxHeaven". "TaxHeaven" is providing a tax preparation service. The service has the maximum delivery time of 24 hours; and accepts only credit card payment. "TaxHeaven" also provides a seperate legal consultancy service for tax payment, called "LegalConfort" and provides 50% discount for this service to the users of their tax preparation service, namely, "TaxHeavenTaxPreparationService".

To provide ease in readability, these ontologies are also shown graphically in Figure 4. The graphical representation followed is inspired by [Gonzalez-Castillo].

3 Associating Semantics with Service Registries

In this section we describe how to associate the semantic defined with services advertised by using the mechanisms provided by UDDI and ebXML registries.

3.1 UDDI Registries

The mechanism to relate semantics with services advertised in the UDDI registries are the tModel and the catagory bags of registry entries. tModels provide the ability to describe compliance with a specification, a concept, a shared design or a taxonomy. Services have category bags and any number of tModel keys can be put in these category bags.

In relating the semantics defined in DAML+OIL with the services advertised in UDDI registries, the first question to be answered is where to store the semantic descriptions. Generic descriptions can be stored by the standard bodies who define them and the server, where the service is defined, can host the semantic description of the service instance. This facilitates the maintenance of the service descriptions. However there are times, when it is necessary to query all the individual service descriptions. Therefore a combined schema per industry domain containing all the semantic descriptions of the services pertaining to this domain may be necessary to facilitate global querying.

The second issue is relating the ontology defined in DAML+OIL with the services advertised in the UDDI registry. For this purpose, similar to WSDL, DAML+OIL should be classified as "DAMLSpec" with "uddi-org:types" taxonomy. A seperate tModel of type "DAMLSpec" for the combined schema of each industry

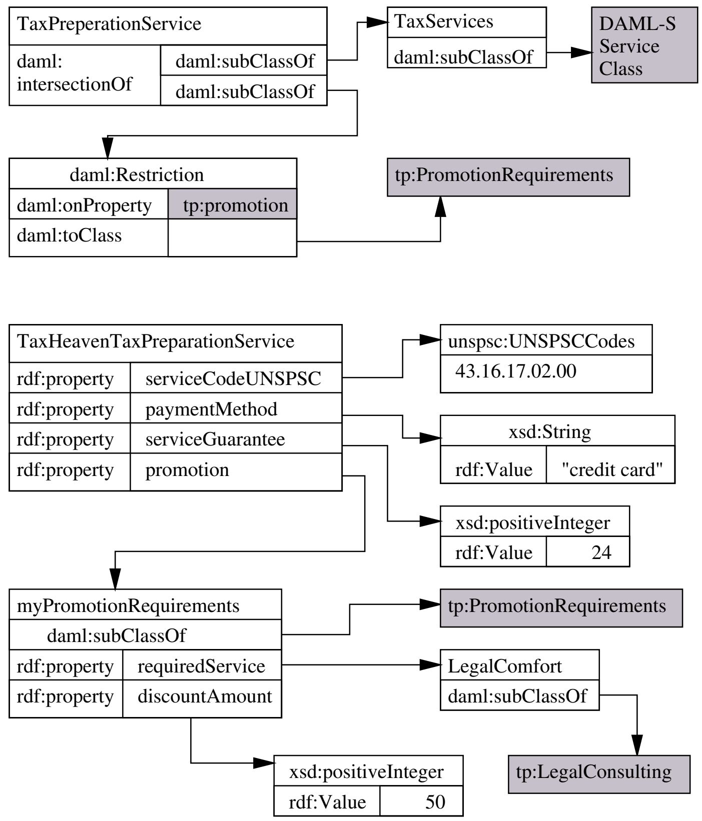

Figure 4: A pictorial description of "TaxHeaven" ontology

domain should be created. The services in an industry domain must contain the key of this tModel in their category bags. Hence, this tModel key can be used to find services in an industry domain through the UDDI registry.

There may be times where it is necessary to find all the instances of a generic class. For example, to choose a tax payment service with required functionality, it may be necessary to retrieve all semantic tax payment service descriptions to check their properties. To be able to do this, that is, in order to find instances of a generic class, it is necessary to associate a tModel key for each generic service class and store this tModel key with individual service descriptions.

Finally, there should be a tModel key for each service instance. This tModel key can be used in searching the UDDI registry to find a particular advertised service instance according to its semantic description. A more detailed treatment of this issue is given in [Dogac].

3.2 ebXML Registries

The basic mechanism in ebXML registries for associating semantics with the objects stored in the registry is the "classification" hierarchy, called *ClassificationScheme*. *ClassificationScheme* defines a hierarchy of *ClassificationNodes*. The nodes in this hierarchy are related with registry objects through *Classification* objects. A *Classification* instance classifies a RegistryObject instance by referencing a node defined within a particular classification scheme. As an example, assume that "TaxHeavenTaxPreparationService" stored in the ebXML registry and a *ClassificationScheme* exits for the "Tax Payment" industry like the one provided in Figure 2. Figure 5 demostrates how this service is associated with this classification schema by using the classification object.

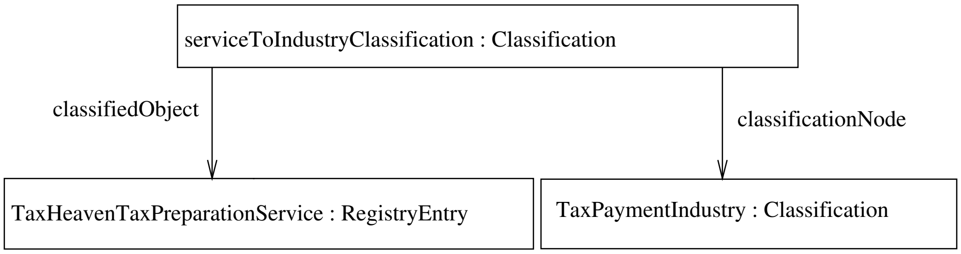

Figure 5: Associating a Service with a Classification Node in ebXML

Note that a registry object can be classified according to any number of classification schemes.

However classification structure provided by ebXML is not adequate to store DAML+OIL ontologies and need to be extended to be used for this purpose. Also ebXML registry interface needs to be extended to query DAML+OIL ontologies.

References

[Berners-Lee] Berners-Lee, T., Hendler, J., Lassila, O., "The Semantic Web", Scientific American, May 2001.

[DAML+OIL] DAML+OIL, http://www.w3.org/2001/10/daml+oil

[DAML-S] DAML Services Coalition (A. Ankolekar, M. Burstein, J. Hobbs, O. Lassila, D. Martin, S. McIlraith, S. Narayanan, M. Paolucci, T. Payne, K. Sycara, H. Zeng), DAML-S: Semantic Markup for Web Services, in Proceedings of the International Semantic Web Working Symposium (SWWS), July 2001.

[Denker] Denker, G., Hobbs, J. R., Narayan, S., Waldinger, R., "Accessing Information and Services on DAML-Enabled Web, Semantic Web Workshop, Hong Kong, China, 2001.

[Dogac] Dogac, A., Cingil, I., Laleci, G. B., Kabak, Y., "Improving the Functionality of UDDI Registries through Web Service Semantics", 3rd VLDB Workshop on Technologies for E-Services (TES-02), Hong Kong, China, August 23-24, 2002.

[ebXML] ebXML, http://www.ebxml.org/

[Gonzalez-Castillo] Gonzalez-Castillo, J., Trastour, D., Bartolini, C., "Description Logics for Matchmaking of Services", Technical Report, HP Labs, Bristol, UK.

[ISO 3166] ISO 3166, http://www.iso.ch/iso/en/prods-services/iso3166ma/index.html

[McIlraith a] McIlraith, S. A., Son, T. C., Zeng, H., "Semantic Web Services", IEEE Intelligent Systems, March/April 2001, pp. 46-53.

[McIlraith b] McIlraith, S. A., Son, T. C., Zeng, H., "Mobilizing the Semantic Web with DAML-Enaled Web Services", Semantic Web Workshop 2001, Hongkong, China.

[NAICS] North American Industrial Classification Scheme (NAICS) codes http://www.naics. com.

[SOAP] Simple Object Access Protocol (SOAP) http://www.w3.org/TR/SOAP/

[O'Sullivan] O'Sullivan, J., Edmond, D., Hofstede, A., "What's in a Service? Towards Accurate Description of Non-Functional Service Properties", in the Journal of Distributed and Parallel Databases, Vol. 12, No. 2/3, Sept./Nov. 2002.

[UDDI] Universal Description, Discovery and Integration (UDDI), http://www.uddi.org/.

[UNSPSC] Universal Standard Products and Services Classification (UNSPSC) http://eccma.org/unspsc

[WSDL] Web Service Description Language (WSDL), http://www.w3.org/TR/wsdl

Integration, Web Services Style

Michael Carey, Michael Blevins, Pal Takacsi-Nagy
BEA Systems, Inc.
{mcarey, mblevins, pal}@bea.com

Abstract

In mid-2002, BEA released WebLogic Workshop, an integrated development framework that dramatically simplifies the task of building enterprise class web services and deploying them in the J2EE environment. In this article, we describe where BEA is going next in terms of web services. In particular, we provide an overview of the next generation of BEA's WebLogic Integration product, which is currently undergoing a major web services inspired overhaul. We explain how web services and application integration relate in the next release of WebLogic Integration; we describe how business processes (a.k.a. workflows) can be used to construct applications by orchestrating web services and how the emerging XML Query standard provides the XML data manipulation capabilities required for web service-oriented integration. We also briefly describe how BEA's new Liquid Data for WebLogic product complements WebLogic Integration by adding web service-based data integration capabilities to the BEA Platform.

1 Introduction

The integration of heterogeneous applications and data has been a long-standing problem in both the enterprise computing industry and computer science research. As evidence, it has been twenty years since the Multibase project at the Computer Corporate of America first tried to provide a common data model and query interface to a collection of disparate database systems [5], it has been over ten years since the OMG CORBA 1.0 specification was published in an attempt to provide a standard way to interconnect distributed heterogeneous applications [7], and it has been ten years since the founders of Tibco (then Teknekron Information Systems) tried to bring order to application integration by developing an event-oriented (*a.k.a.* publish/subscribe) "information bus" with application adaptors [6].

With the advent of XML and its accompanying technologies and standards, it appears that there is finally real hope on the integration horizon. In particular, web services offer many of the basic ingredients (in what have become widely accepted forms) required to crack the integration problem in a standards-based way [4] - including a simple and flexible physical data format (XML), a logical data description mechanism (XML Schema), a loosely-coupled remote invocation facility (SOAP), a service interface description mechanism (WSDL), and a declarative query language (XQuery). Work is in progress on standards for some of the "holes" in today's web service technology, such as XML web service orchestration protocols, security, and transactions. As a result, it is our belief that web services will finally succeed where various previous architectures and standards have fallen short.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

Earlier this year, BEA released a major new product called WebLogic Workshop [BEA02a]. BEA WebLogic Workshop provides a unified platform consisting of an integrated development environment and a runtime container [BEA02b] that make developing and deploying J2EE-based enterprise applications an activity that all corporate application developers (not just Java/J2EE experts) can undertake successfully. Release 1.0 specifically focuses on the task of implementing, composing, and deploying industrial-strength web services. To make it easy to use enterprise resources such as databases, JMS queues, existing EJBs, and other web services to build a new web service, WebLogic Workshop provides a controls framework - developers can set properties, call methods, and handle events using controls that provide simplified interfaces to the resources, shielding them from the low-level details and complexities required to use such resources directly. Web service invocations can be transported over HTTP or JMS, can be stateless or stateful, and can be either synchronous or asynchronous. WebLogic Workshop generates the low-level J2EE artifacts (stateless session beans, entity beans, message-driven beans, JDBC calls and result set processing, etc.) needed to support the applications' choice of controls and control annotations.

2 Web Services and Application Integration

In the past, application integration technologies have suffered from drawbacks such as insufficiently loose interface bindings (e.g., CORBA [7]) or use of proprietary object models (e.g., first-generation EAI products such as [6]). XML and web services offer dramatic improvements in these respects. The next generation of BEA WebLogic Integration (WLI 8.0) is based solidly on XML and web services standards, particularly in its new business process management capabilities. On the "backend" side, it utilizes the WebLogic Workshop web service control paradigm to make it simple to construct business processes that integrate applications that are exposed through web service interfaces. Many packaged application vendors, such as SAP, Siebel, and PeopleSoft, are moving towards providing web service APIs; others are building tools that export legacy mainframe applications in web service form, while still others are making important functionality for business partners available through web services (e.g., the Amazon.com Web Service). In addition, BEA WLI 8.0 (like WLI 7.0) includes a set of J2EE-CA adaptors that provide XML "application views" (accessible via WebLogic Workshop's application view control) that expose XML interfaces to a wide variety of non-web-service-enabled systems. Within a business process, WLI 8.0 provides built-in support for XML workflow variables, optionally typed by XML Schema types, as well as for Java workflow variables. XML variables are used for storing and manipulating the XML documents and data that flow through integration-oriented applications; their XML content is accessible via Xquery [8] statements. In addition, WLI 8.0 provides web service invocation as one of the primary means for starting or continuing interactions with business processes - yielding a paradigm where business processes *are* web services.

2.1 Business Processes as Web Services

Figure 1 shows a simple example of a business process for handling purchase orders. This process was created using the graphical business process editor of WLI 8.0, which supports the creation and editing of Java Work Flow (JWF) programs. Beneath this graphical view, a JWF is a Java class with annotations that describe the flow (i.e., the business process orchestration) logic; the annotations reference Java or XQuery methods within the class that implement the detailed business logic. This simple purchase order example illustrates how web services relate to business processes in WLI 8.0 - this workflow exposes a SOAP operation that accepts a purchase order asynchronously, places orders for the line items contained therein, and then responds to the requestor with a purchase order reply message by performing a SOAP callback. The process uses a JWF *foreach* loop construct to iterate over the set of line items in the purchase order. In the underlying JWF file for the business process (not shown), the incoming purchase order is stored in an XML workflow variable and an XQuery expression is used to control the looping by enumerating each line item in turn from within the XML purchase order variable. Inside the loop, a web service call is made to send the line item to a backend order management system, and the response comes back from the order management system in the form of a web service callback. JWF includes constructs to specify such flow actions as message sending and receiving, looping, conditional branching, parallel execution, waiting for one of a possible set of messages, Java method invocation, and transaction and exception handling.

It is important to note that JWF flow annotations are not intended to replace Java as a general-purpose programming language. Rather, the flow annotations augment Java's capabilities by adding support for the construction of *long-running, reliable, asynchronous* programs. For example, the line item callbacks in the purchase order example above can take a large (indefinite) amount of time to occur, perhaps hours or even days depending on the nature of the backend system. One of the major benefits of a flow language is that it enables applications like this to be easily constructed by corporate developers. The JWF runtime container uses transactions and queuing to reliably execute and sequence (and recover, if necessary) the individual Java- and/or XQuery-based workflow steps, it handles call/callback correlation, and it enables the application to be deactivated (utilizing entity beans and persistent storage under the covers) during long periods of inactivity - even in the midst of loops in the flow, etc. Note also that the flow description essentially indicates which, and when, various types of messages are expected by the workflow. The workflow container then delivers incoming messages, such as calls and callbacks, to the workflow in the order expected - which may differ from their actual order of receipt. Issues such as these cannot be solved by the Java programming language alone, and handling them manually by utilizing Java and advanced J2EE features such as entity beans and message-driven beans is a difficult task, especially for a corporate application developer.

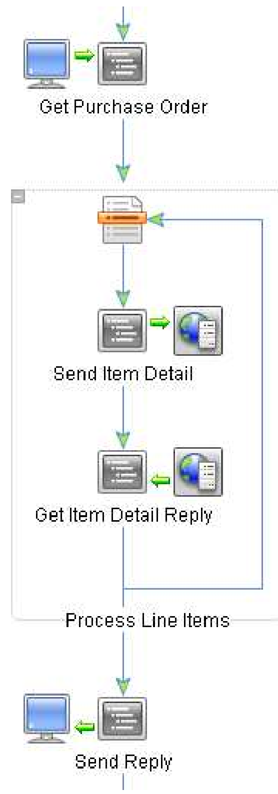

Figure 1: Purchase Order Business Process.

2.2 Slicing and Dicing XML Data

In order to handle the XML data flowing through a business process, WLI 8.0 embeds a high-performance streaming Xquery [8] execution engine. XQuery queries can be found in several roles within business processes - they can be used to transform data on the way into or out of a workflow, and they can also be used to control workflow looping and branching. WLI 8.0 includes graphical XQuery editing tools for these purposes. Figure 2 shows a screen shot from the user interface of the WLI 8.0 XML data transformation tool; a user selects the desired source and target XML Schema descriptions for a given data transformation and then interacts with the tool to describe the transformation in an intuitive graphical manner. Figure 3 shows the XQuery query that is created and (two-way) edited as the result of Figure 2's visual XML mapping specification.

In addition to XML data, the same XML Schema-centric mapping paradigm is used to specify mappings between XML and binary data formats. In the case of binary data, the data transformation developer provides an additional XML file that describes to the system how to parse the binary format of interest to/from XML; this file can be created graphically as well by using the FormatBuilder tool already included as part of WLI 7.0.

Internally, WLI 8.0 includes an XML "document store" layer that provides the required backing store for XML and binary messages and workflow variables. Such data is parsed once, on entry to the system, and then

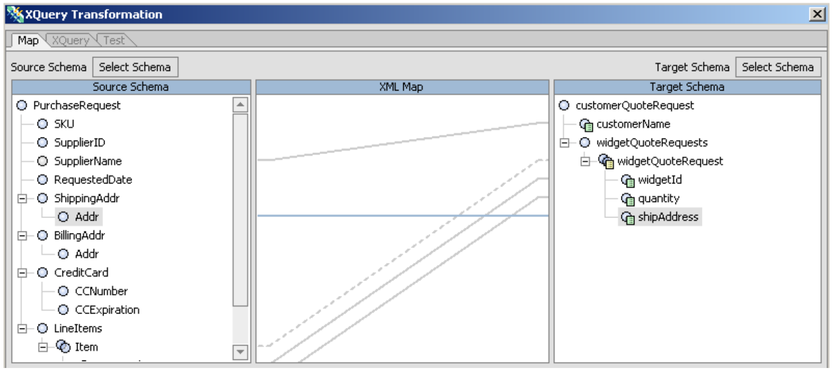

Figure 2: User Interface for XML Data Transformations.

```
<customerQuoteRequest>
    <customerName>{data($src/SupplierName)}</customerName>
    <widgetQuoteRequests>
     {
       for $Item_1 in $src/LineItems/Item return
           <widgetQuoteRequest>
               <widgetId>{data($Item_1/PartNumber)}</widgetId>
               <quantity>{data($Item_1/Quantity)}</quantity>
               <shipAddress>{data($src/ShippingAddr/Addr)}</shipAddress>
           </widgetQuoteRequest>
        }
    </widgetQuoteRequests>
</customerQuoteRequest>
```

Figure 3: Corresponding Generated XQuery.

stored in an efficient tokenized form for further consumption. Also, XML and binary data is passed by reference between workflows running within the same WebLogic Server cluster for added efficiency.

3 Web Services and Data Integration

In addition to taking an XML- and web-services-centric approach to application integration, BEA now has a newly announced mid-tier data integration offering - BEA Liquid Data for WebLogic [3] - which is built on the same standards-based foundation. Liquid Data is designed to complement BEA's WebLogic Integration product by providing a unified, real-time XML view of disparate enterprise data sources. Specifically, Liquid Data views each data source in an enterprise as being an XML data source with a given XML Schema. Sources supported today include relational databases, XML files, web services, WLI adaptor-based application views, and custom data sources (exposed by registering custom Java functions that consume and produce XML). BEA Liquid Data includes an extensive set of administration pages, delivered as an extension to the BEA WebLogic Server application server's console, that enable data sources to be added, configured, and dropped. For creating the unified XML view of the enterprise, Liquid Data includes a graphical Data View Builder tool (roughly similar in function to the WLI 8.0 data transformation tool) that enables a data architect to create XQuery-based views and parameterized stored queries. Note that, like WLI 8.0, the XML, XML Schema, and XQuery web standards lie at the heart of Liquid Data.

Web services play two roles in BEA Liquid Data for WebLogic. On the data source side, web services are an important class of data source that can be included in Liquid Data views and queries. For example, one could form a unified Customer view in Liquid Data by combining customer information from an Oracle database with order information exported by a backend order management system's web service APIs. On the data producer side, as mentioned above, a data architect can use Liquid Data to create a set of parameterized stored queries for use by enterprise application developers. These queries can be exported to the enterprise's internal developer community via web service APIs. (EJB APIs are also supported.) Note that this paradigm provides a very important form of control to the data architect - rather than exposing the enterprise's various operational data sources to ad hoc queries, the architect chooses and publishes a limited set of parameterized queries (e.g., find customer by id, find customer by name, etc.) that have been deemed acceptable to run against the operational data. In this way, Liquid Data uses web services to provide real-time data visibility and aggregation capabilities for "front-office" applications (such as customer care systems, employee self-service portals, and so on).

In terms of their relationship, BEA Liquid Data provides integrated read access to disparate enterprise data, while BEA WebLogic Integration provides the corresponding update capability required to build complete applications. For example, "view employee" is an operation easily provided by Liquid Data, while a corresponding "insert employee" operation requires a potentially complex workflow that may take several days in a large corporation (e.g., to notify the hiring manager, obtain computer and e-mail accounts, arrange office space, notify benefit providers, receive and persist acknowledgements of all these actions, etc.).

4 Conclusion

In this short paper, we have provided an overview of how web services lie at the heart of BEA's approach to application and data integration. In the next generation of the BEA WebLogic Integration product, WLI 8.0, business processes are naturally exposed as web services. In addition, business processes in WLI 8.0 can themselves consume other web services, making it easy to compose new long-running application activities through web service and business process orchestration. Inside, these processes can manipulate XML data, based on XML Schemas, using a combination of high-level JWF actions, XQuery logic, and Java logic. BEA's new Liquid Data for WebLogic product provides integrated real-time access to disparate enterprise data sources. Again, web service APIs play a key role both outside (where the data architect publishes the available query operations as web services) and inside (where web services can themselves be Liquid Data data sources) the Liquid Data system, and inside, the XML Schema and XQuery standards are key. Coupled with WebLogic Server and WebLogic Workshop as part of the overall BEA WebLogic Platform product suite, Liquid Data for WebLogic and WebLogic Integration provide a comprehensive, web-service-based foundation for addressing the integration problems faced by today's enterprises.

References

[1] BEA WebLogic Workshop: The Future of Web Services - Here Today, BEA Systems White Paper, March 2002.

[2] The BEA WebLogic Workshop JWS Container, BEA Systems White Paper, August 2002.

[3] BEA Liquid Data for WebLogic: Increasing Visibility in the Distributed Enterprise, BEA Systems White Paper, October 2002.

[4] Curbera, F., et al, Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI, IEEE Internet Computing 6(2), March-April 2002.

[5] Landers, T., and Rosenberg, R., An Overview of Multibase, Proc. 2nd Int'l. Symp. on Distributed Data Bases, Berlin, Germany, North-Holland Publishing Co., September 1982.

[6] Oki, B., et al, The Information Bus - An Architecture for Extensible Distributed Systems, Proc. 14th ACM Symp. on Operating Systems Principles, Asheville, NC, December 1993.

[7] Vinoski, S., Distributed Object Computing with CORBA, C++ Report, July-August 1993.

[8] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/, W3C Working Draft, August 2002.

Conductor: An Enabler for Web Services-based Business Collaboration

David Burdett, Qiming Chen, Meichun Hsu
Commerce One Inc.
19191 Vallco Parkway, Cupertino, CA 95014

Abstract

To enable intra- and inter-enterprise application integration, enterprises have invested heavily in EAI (Enterprise Application Integration) and B2B (Business to Business) technologies. Recent momentum in Web Services based on the SOAP, WSDL and UDDI specifications promises a standardized connectivity at a lower cost that would dramatically transform the business connectivity paradigm. In this article, we introduce Commerce One Conductor, an enabler for business collaboration based on robust web services, and provide an overview of 2 specific design points: (a) Conductor's web service extensions that address security, guaranteed delivery, routing, choreography, and document and service version interoperation, and (b) a Conversation Manager-based mechanism to extend the business process engine to provide collaborative web services.

1 Introduction

E-business is moving toward a paradigm in which enterprises interact with each other through exchanging XML documents based on well-defined protocols such as SOAP, WSDL, UDDI, and BPSS. These emerging Web Service standards enable businesses to interoperate in a dynamic and loosely coupled way. Based on Forrester Research [5], firms will expand their use of partners, private, and public Web Services in the next five years, and Global 3,500 firms are expected to spend about USD 7m on a private hub to enhance visibility and improve supplier collaboration.

Commerce One has focused on eliminating friction within enterprises and between trading partners through information exchange via a Marketplace platform called MarketSite. Taking the lessons learned in connecting large corporations together to trade, Commerce One is transforming MarketSite into a business collaboration platform, called Commerce One Conductor, based on emerging web service standards and the service oriented application architecture. It abstracts both human activities and business applications into "services": human activities are known as Portal Services, while business applications expose Web Services. It enables new business processes to be rapidly constructed by linking services offered by new and existing business systems. Conductor fuses three key technologies - Service Registry, Web Service Engine, and Collaboration Engine - to enable public or private hubs (Figure 1):

- *Service Registry*: All Portal Services and Web Services are described and published in the Conductor Service Registry. A *Conductor Community Hub* is a server that administers the registry as well as managing

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

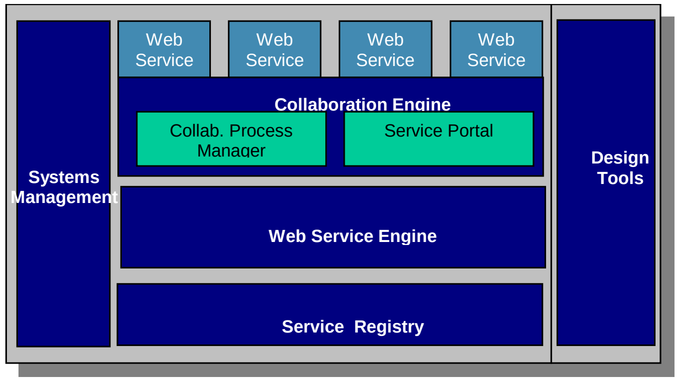

Figure 1: Key technology components of Conductor.

service provisioning, subscription, and interoperability. A Service Registry defines business organizations and business services in a "community". A community can be an enterprise, an extended enterprise, or a public marketplace. The Service Registry significantly extends from UDDI-based registries to include specifications of organizational structures, trading partners, and related security information. Cross-community interoperation (e.g., between a buyer enterprise and a supplier enterprise) is supported through Service Registry interoperation. Service Registry contents can also be uploaded to, or aggregated into, a standard UDDI registry to be searched by UDDI-based tools for partner and service discovery.

- *Web Service Engine*: Web Service Engine provides the underlying connectivity fabric for web service messages. Instead of just supporting simple web service invocations, Web Service Engine supports Soap extension for robust business interactions (see next section); it obtains information needed for these extensions from the Service Registry.
- *Collaboration Engine*: Collaboration Engine allows new business processes to be quickly constructed out of new or existing business systems while weaving in human activities, creating new business models and partner interactions. It consists of a *Collaborative Process Manager (CPM)* and a *Service Portal*. CPM links and orchestrates human actions (portal services) as well as business applications (web services) across organizations. End users access activities assigned to them in the context of a business process via a Service Portal accessible to users within an enterprise as well as to trading partners.

2 From Simple Web Services to Collaborative, Enterprise-grade Web Services

While simple web services as currently defined by SOAP and WSDL standards are useful in intra-enterprise information aggregation, or for readily available inter-enterprise queries, they lack security, reliability, and other key collaboration features needed for general business transactions. Conductor directly supports the following extensions to standard web services:

- Security, reliability, routing and addressing: A service provider can specify if collaboration parties are required to provide signing, encryption, and/or other security information when invoking a service instance. It can also specify if reliable transport must be used, and if other delivery features are expected, such as delivery receipts or error returns. Furthermore, services can be logically routed within a community or cross communities, eliminating the restriction and impracticality of point-to-point service invocations.
- Business document schemas and document family: Business document schemas are explicitly supported in the Registry and they are grouped into document families. For example, XML schemas for xCBL 3.5 PurchaseOrder and 4.0 Purchase Order are two versions of documents in the PO document family. A service definition may refer to xCBL PO as the message content, while the service instance can further specify which version or versions of PO are supported in the service implementation.

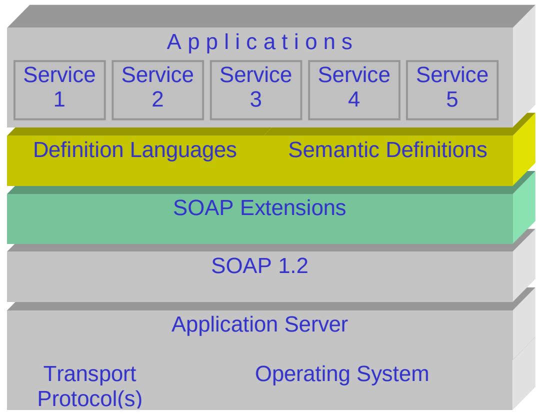

Figure 2: Standards Stack for Service-based Applications in Conductor.

- Choreography and collaborative (or long-running) services: Choreography specifies the sequence of service invocations expected of collaboration parties. For example, a service definition "OrderManagement_SupplierSide" service may contain definitions of 2 operations: "receive_order" and "receive_order_confirmation". A choreography specification may require that the first service invocation occurs before the second one. It is expected that all service instances that conform to this service definition also conform to the choreography specification.

The standards stack that a Conductor application (a web service) is built on is shown in Figure 2. It outlines 2 major extensions on top of standard Application Server and SOAP 1.2: (a) Definition Language and Semantic Definition extensions, and (b) SOAP extensions.

3 Implementing Web Service Extensions

Service Registry supports extended definitions and service descriptions. Like WSDL, service descriptions are partitioned into a definition part ("types") and implementation and binding part ("instances"). The Registry provides structured storage for all elements in the service definitions and descriptions, promoting reusability while offering the ability to "materialize" the descriptions in WSDL documents. Service Registry supports the following definitions in addition to standard WSDL information:

- Service instance attributes such as security, reliability, delivery properties, routing flexibilities
- Document schemas (based on XSD definition language) and semantic document family (based on published xCBL [10] and UBL [9] standards); transformation maps (based on XSLT but also accommodating other high performance variations) among members of document family and between document families.
- Service choreography definitions.

3.1 Enterprise-grade Web Services

Conductor's Web Service Engine processes service invocation messages based on the extended service standards. It embeds an interoperability resolver and supports Soap Extensions. Interoperability resolver is capable of computing document transformation rules and routing rules based on service descriptions in the Service Registry. For example, if a service invoker invokes an OrderManagement_Supplier service with a version 3.5 of the PO document, while the service only supports version 4.0 of PO, a transformation map is applied directly in the Web Service Engine before the invocation is sent to the service.

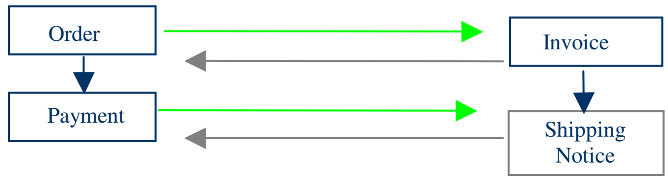

Figure 3: Choreographed document exchange as conversation process.

Soap Extensions define SOAP modules that provide enhanced functionality over simple SOAP to provide security, reliability, and collaboration extensions. In defining such extensions, we adopt small, self-contained, reusable SOAP module specifications; each specification provides useful functionality on its own. We also define profiles that describe how specifications are used in combination. For example, in a B2B scenario, signing, encryption, and authentication are needed in the protocol, which can be ommitted in an intra-enterprise query. This approach is taken in contrast to a single monolithic specification of complex web service, which will be too big for lightweight solutions, and will take too long to develop. The Soap Extensions supported in the Web Service Engine are summarized below, with references to other related evolving standards where appropriate:

- Security: Digital signatures, encrypted messages, and authentication are supported according to WS-Security [8]. In addition, a SAML-based mechanism [6] is used to provide the ability to pass authorization between servers.
- Reliability: Reliable messaging is achieved in Web Service Engine by adding a reliable queue-based transport. This is done in contrast to developing a reliable messaging extension in Soap. Therefore, reliable messaging is possible only when both ends of messaging use the same reliable queue implementation. On top of the reliable transport, delivery receipts (evidence that the message was delivered to the application), message expiry (how to determine if a message has ”expired”), and return messaging (how to specify the types of message to send back) are supported which strengthens audit and diagnosis required by most business transactions. The emerging WS-Transaction [8] standard, which attempts to provide atomic commit semantics, is an alternative.
- Addressing and Routing: Instead of direct, physical addressing to the service, a logical addressing scheme in the form of <Collaboration Party, Service Identifier, Action> has been defined. When sending a message to a community, the logical address uniquely identifies the server that handles the message within that community. In addition, the protocol includes a provisioning for specifying a ”return address”, and for dynamic routing. The latter differs from the WS-Routing [8] spec which is a mechanism based on computing the complete route of the message in advance. Dynamic routing offers more flexibility when crossing multiple communities, especially relevant in the B2B scenarios.
- Message Metadata: Message Metadata specifies message identification and correlation. It also provides references to agreements and choreographies. This is a key extension to support collaborative or long-running web services.

3.2 Collaborative Web Services

In order for enterprises to collaborate at the business-process level, web services must be extended to allow long-running interactions. Point activities in peer services are linked together in the context of a long-running business process instance that we call a conversation. Figure 3 illustrates a conversation.

Conductor Soap Extensions support a correlation identifier field in Metadata, which identifies the message with a long-running conversation. This field is automatically populated and passed along during the course of the conversation when collaborative services interact. Collaborative services are driven by process descriptions in addition to service interface descriptions. We distinguish between public conversation models, specifying inter-service document flows, and private business process models, specifying local workflows or compositions

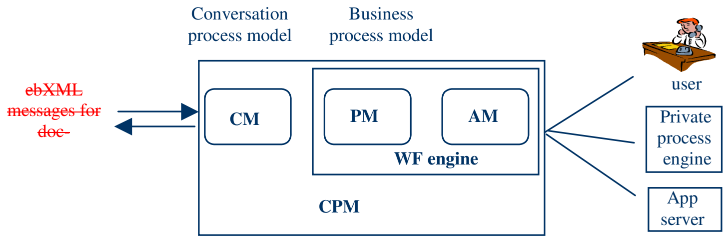

Figure 4: CPM - conversation manager, process manager and action manager.

of local services. Emerging standards for public conversation models fall into 2 categories: those specifying the global interface of a conversation (e.g. BPSS [2]), and those specifying the local interface of a conversation (e.g. WSCI [7], BPEL4WS [1]). In Conductor, a collaborative service definition can refer to a BPSS-based public conversation specification as its Choreography. By specifying a choreography, the service defines the expected behavior of collaborating services. Support for languages other than BPSS can be added. Collaborative services are implemented by the CPM component (see Figure 4) within Collaboration Engine [3, 4]. CPM consists of a Process Manager (PM) and a Conversation Manager (CM). PM executes the private business processes, while CM is responsible for validating and enforcing the choreography in the context of a conversation. The Action Manager (AM) dispatches and invokes local applications, services or processes to perform process tasks.

4 Conclusions

In this article we provided an overview of a web services-based business collaboration platform. We described key web service extensions required to serve the needs of business collaborations; these requirements are compiled based on Commerce One's extensive experience in developing B2B business applications and conducting B2B transactions on Market Place platforms. Conductor represents a wave of next-generation collaboration platforms to power standards-based application networks for extended enterprises.

References

[1] Business Process Execution Language for Web Services, Version 1.0, July 2002. http://www-06.ibm.com/developerworks/library/ws-bpel/

[2] Business Process Specification Schema, V1.01, 2001. ebXML.org.

[3] Qiming Chen, Meichun Hsu, Inter-Enterprise Collaborative Business Process Management, Proc. of ICDE 2001, Germany.

[4] Qiming Chen, Meichun Hsu, CPM Revisited - An Architecture Comparison, Proc. of CoopIS 2002, USA.

[5] The Web Services Payoff, Forrester Report, Dec. 2001. Which Industries will adopt Web Services?, Forrester Brief, Dec. 2001.

[6] Committee working draft, Binding and Profiles for the OASIS Security Assertion Markup Language (SAML). http://www.oasis-open.org/committees/security/docs/draft-sstc-bindings-model-09a 10 Jan. 2002

[7] Web Service Choreography Interface, Tech Report by Italio, SAP, BEA, Sun Microsystems. 2002.

[8] WS-specs http://msdn.microsoft.com/webservices/understanding/specs/default.aspx

[9] http://www.oasis-open.org/committees/ubl/

[10] http://www.xCBL.org.

Phoenix Application Recovery Project

Roger S. Barga
Database Research Group, Microsoft Research
One Microsoft Way, Redmond, WA 98052

Abstract

High availability for both data and applications is rapidly becoming a business requirement. Yet even after decades of software engineering research, computer systems and production applications still fail, primarily due to nondeterministic bugs that can typically be resolved by simply rebooting the system or restarting the application. The Phoenix project takes the position that such hardware faults and software failures are facts to be coped with, not problems to be solved This position is supported both by historical evidence and by recent studies on the primary sources of outages in production systems. Conceding that Heisenbugs will remain a fact of life in computing, the goal of the Phoenix project is to enable applications to survive system crashes, without requiring application developers to take special measures for state persistence and application recovery. This simplifies application programming, reduces operational costs, masks failures from users, and increases application availability. In this paper we describe the principles behind the Phoenix project and outline some research areas and recent projects.

1 Introduction

Availability is as important to e-commerce as breathing in and out is to human beings.
- Kal Raman, CEO Drugstore.com.

High availability is crucial to the success of any company engaged in e-commerce and e-business. There is a growing consensus within parts of the systems and research communities that the traditional focus on performance has become misdirected in enterprise computing, where availability has become at least as important as peak performance, if not more so. One need only open a recent issue of the *New York Times* or *Wall Street Journal* to find evidence of this – the number of articles on recent outages of big e-commerce providers and the economic impact of those outages is striking. In April 2002 when Ebay suffered a 22-hour sustained outage affecting most of its online auctions the loss of revenue was estimated at five million US dollars [15] but their stock value fell by 26% during this downtime, representing a loss of over five billion dollars in market capitalization. This and similar outages are highly visible, noteworthy, and affect customer loyalty and investor confidence [7]. While the immediate result of key application downtime is lost revenue, in the longer term customer frustration, damaged reputation and investor confidence may lead to even greater losses.

Unfortunately, system outages and application failures do occur. The Gartner Group [9] estimates that over 40% of unplanned downtime in business environments is due to application failures; 20% is due to hardware

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

faults, of which 80% are transient [6, 13], hence resolvable through system restart. Most bugs in production quality software are Heisenbugs [6, 10, 1]. Heisenbugs are difficult to reproduce, or depend on the timing of external events, and often there is no better way to work around them other than to simply restart the application. While restart will work around transient hardware failures and Heisenbugs in software, the volatile state of user sessions and running applications is lost, exposing the failure to end users. Hence, automatic restart alone does not fully address the problem.

Transaction processing monitors, exploiting a database or durable queue, and "stateless" applications, have long been the preferred solution for constructing failure-resilient enterprise applications. Stateless applications are not truly without state, however. Rather, they are written so that each application step executes entirely within a transaction. A step begins by retrieving prior state from a queue or database, executes, and then stores the resulting end state back into a queue or database. Such an application does not have "control state" outside of a transaction, though of course, the state saved in the queue or database lives across transaction boundaries.

With the advent of web-based electronic services, however, such prior solutions have not carried over to multi-tier web architectures. Indeed, how they might be readily adapted to deal with middle-tier web application servers is complex and still speculative. Clearly, the enterprise computing world is different today than it used to be. With traditional back-office transaction processing systems being supplanted by multi-tier, Internet-based service delivery, and component-based application servers, the assumptions behind system design have changed, and the traditional techniques for providing high availability have lost much of their power. Since high availability is as important today as it has always been, we need new techniques and approaches for improving application availability, ones based on the realities of modern server environments.

2 The Phoenix Project

The goal of the Phoenix project, a project within the Database Group at Microsoft Research, is to enable applications to survive system failures, in particular Heisenbugs in software and transient hardware failures. Permanent members of the Phoenix project are David Lomet and Roger Barga. Our approach is to enable the underlying system to take responsibility for application state persistence across failures. It is the system that acts to ensure that volatile session and application state is recoverable by transparently logging application interactions, and should a failure occur it is the system that will automatically recover the application on restart. This enables applications to be written naturally as stateful programs and simplifies application development by eliminating the need to write explicit code to persist application state and to deal with failures. This approach also enhances overall application availability by avoiding the extended down-time that failures can produce when manual intervention is needed to make sense of failed application state and restart the application. A secondary goal of our work is to be general enough to enable the implementation of other useful state management services, such as resource management and administrative utilities.

To realize this goal for multi-tier Internet based applications with communicating components, a comprehensive form of data, component state, and message recovery is required, going beyond traditional database recovery and transaction processing. In designing protocols to accomplish this we have addressed a number of issues, such as:

- To what extent can failures be masked from end users, and which logging actions are necessary for this.
- Which system component logs which messages or state to be recoverable, mask failures, and provide exactly-once semantics to a user?
- How are logs managed, when is a log force written to disk, and how are logs coordinated for log truncation, crucial for fast restart and thus for high availability?
- How are critical components, e.g., database servers, kept from being held "hostage" to other components (applications servers, clients), which may interfere with their independent recovery or normal operation?

To realize our goal, we have explored logging and recovery principles, resulting in a framework for recovery guarantees in multi-tier applications, and have built a number of prototype systems to demonstrate the practicality of our approach. In the remainder of this article we outline our framework for recovery guarantees and then describe three of our more recent projects based upon this framework.

3 Selected Research Areas and Projects

Our recent project work is based on a framework for recovery guarantees in general multi-tier applications that makes system component state persistent, as described in [5]. To a user, the framework masks all failures such that the user's initial request has exactly-once execution semantics. To provide this guarantee for applications we require piecewise deterministic applications and identify the needs for logging specific non-deterministic events. This ensures that after a failure, a system component can be replayed from an earlier installed state and arrive at the same state as its pre-failure incarnation. There are three types of components considered in our original version of this framework: eXternal components (**XCom**) modeling human users and parts of the system outside the framework, Persistent components (**PCom**) representing mid-tier applications, and Transactional components (**TCom**) modeling resource managers such as database servers and transactional queues.

The framework introduces *interaction* contracts between two persistent components. For example, a committed interaction contract between persistent components requires guarantees related to persistence of sender and receiver state and messages. Contracts exist for persistent component interactions with external components (including users) and with transactional components that provide all-or-nothing state transitions (but not exactly-once executions).

Finally, we compose contracts into system-wide agreements such that persistent components are provably recoverable with exactly-once execution semantics. We strictly separate the obligations of a contract from its implementation in terms of logging. We can thus give strong guarantees to the external users while frequently avoiding forced logging and expensive measures.

3.1 Persistent Middleware via Phoenix/APP

Phoenix/APP is a runtime service that provides transparent state persistence and automatic recovery for component-based applications [4], and was the first prototype system based on the recovery guarantees framework. Building highly available enterprise applications using web-oriented middleware is difficult. Consider an enterprise middle tier application. It is typically made up of one or more server components that implement business logic and expose a set of interfaces. A handful of the components in the application may access persistent data, typically stored in a relational database. Many components perform a specific task (calculation, data formatting, etc) on behalf of server components, possibly modifying data and returning a result but they retain no state. Finally, there are a small number of critical components which maintain state for the application during the session. A classic example is a middle tier e-commerce system for shopping and price comparison. A client session begins when a customer logs in and customer information is read from a database into a component. As the customer shops, purchases are recorded in the stateful component representing the market basket. When the customer checks out, items in the market basket are written to databases (e.g., orders and billing) and the customer database is updated to reflect recent activity. If a failure occurs during the session, all volatile state is lost and the session must be restarted.

To achieve fault-tolerance using Phoenix/APP simply requires the application programmer to identify the stateful components and declare them as *persistent*, *transactional* or (by default) *stateless*. The runtime service transparently logs volatile state and, if a failure occurs, automatically recovers all components marked *persistent* up to the last logged interaction[1]. Components marked *transactional* are recovered up to the last successfully

[1]The time required to recover failed components is a consideration. For middle tier applications, the major response time overhead

completed transaction – transactions in-flight during the failure will be aborted by the database and it's assumed the application is written to deal with (retry) failed transactions – a reasonable assumption for transactional applications. The remaining components in the application, which by default are treated as stateless, are simply restarted (no state is recovered). The application can continue with the session without loss of state (market basket, orders, etc) and having to take any special actions for it own recovery or state persistence.

Phoenix/APP is implemented on the Microsoft .NET framework and supports any component based application. Our implementation is based on a mechanism called *contexts*, a component wrapper mechanism that transparently intercepts object events, such as creation, activation, and method calls. New component services can be added to the .NET runtime by implementing "handlers", referred to as *policies*, for object events and method calls, and including them "in the context". Phoenix/APP runtime service for automatic component recovery is the composition of context policies that work together to mask failures from the application and initiate recovery to reconstruct impacted components. These policies involve:

1. **Logging** – intercepts interactions between components and logs information sufficient to recreate components and recover their state to the last logged interaction.
2. **Error Detection** – detects errors arising from component failure, masks the error from the client, and initiates component recovery.
3. **Component Recovery** – recreates an instance of a failed component and re-installs its state by replaying intercepted calls of the failed component from the log. After recovery, the .NET runtime tables are updated to direct future calls to the new component.

Post failure execution requires a different process and thread than used by the original execution. Therefore, Phoenix/APP virtualizes components, providing each with a logical id that is independent of its mapping to a physical process or thread. This logical id is used to identify the program and the persistent state of a component. During execution, this id is then mapped to the specific threads and/or processes that realize the object. Further details on the design and implementation of Phoenix/APP can be found in [4].

The Phoenix/APP effort has benefited from the efforts of several graduate students interning with our group. Sirish Chandrasekaran (UC Berkeley, Summer 2000) contributed to the initial design of Phoenix/APP. Stelios Paparizos (Northeastern University, Winter 2000 and Summer 2001) extended the functionality of the prototype during his first internship and during his second internship, together with Haifeng Yu (Duke University, Summer 2001), they ported the prototype to the .NET runtime and performed a detailed performance evaluation.

3.2 New Component Types for Improved Logging and Recovery Performance

In Phoenix/APP, logging, in particular forced logging, is the largest performance cost associated with the normal execution. Thus, reducing forced logging is the most important aspect to reducing the overhead required to provide application state persistence. In our paper describing Phoenix/APP [4] we introduced three new component types, *subordinate components*, *functional components*, and *read-only components*. Each of these new component types has reduced logging overhead during interactions between it and other component types. As with our previous component types, *persistent*, *transactional* and *external*, an application builder simply declaratively states the type of the component in their application and the Phoenix/APP runtime service will tailor its logging actions based on the type of the component involved.

During his internship with our group, Shimin Chen (CMU, Summer 2002) examined the logging and recovery requirements in Phoenix/APP to eliminate unnecessary logging and log forces, and took advantage of these new component types and methods to improve runtime logging overhead. In addition, he devised mechanisms to checkpoint component state so that the log could be truncated, reducing processing costs during recovery,

is communication with remote components or resource managers. During recovery, the service replaces these interactions with logged effects of the original interactions. Hence replay is much faster than original execution.

and introduced methods to coordinate the state saving of multiple components in a process. These enhancements were implemented in a new version of the Phoenix/APP research prototype, which is now the subject of a performance evaluation that will be presented in a forthcoming paper [3].

3.3 EOS - Persistent State for Internet Browsers

Gerhard Weikum and German Shegalov have implemented a prototype system at the University of the Saarlands that extends our recoverable components work to an Internet browser as client, an http server with a servlet engine as middle-tier application server, and a database system as backend data server [14]. Specifically, they have built the prototype using Microsoft's IE5 as the browser, Apache as the http server, and PHP as the servlet engine. The prototype transparently provides an external interaction contract XIC between the browser and user, and a CIC between the browser and mid-tier application server. This permits both browser and mid-tier application to be persistent components. This should prove to be very relevant for Internet-based e-services. Building the prototype required extensions to the IE5 environment in the form of JavaScript code in dynamic HTML pages, modifications of the source code of the PHP session management in the Zend engine, and modifications of the ODBC-related PHP functions as well as stored procedures in the underlying database, as described in [14].

Additional Information on the Phoenix Project
The Phoenix project home page is http://www.research.microsoft.com/db/phoenix. This webpage is updated regularly and includes links to relevant references.

References

[1] E. Adams. Optimizing preventative service of software products. IBM Journal of Research and Development, 28(1):2-14, 1984.

[2] Nick Allen. Gartner Group Research Report, 2000.

[3] S. Chen, R.S. Barga and D. Lomet. Improving Logging and Recovery Performance in Phoenix/APP. In preparation, 2002.

[4] R.S. Barga, D. Lomet, S. Paparizos, H. Yu and S. Chandrasekaran, Persistent Applications via Automatic Recovery. Submitted 2002.

[5] R.S. Barga, D. Lomet and G. Weikum, Recovery Guarantees for General Multi-Tier Applications. ICDE 2002.

[6] T.C. Chou. Beyond fault tolerance. IEEE Computer, 30(4):31-36, 1997.

[7] Chet Dembeck. Yahoo cashes in on Ebay's outage. E-commerce Times, June 18, 1999. http://www.ecommercetimes.com/perl/story/545.html

[8] http://www.forrester.com/research/cs/1995-ao/jan95csp.html.

[9] http://www.gartner.com/hcigdist.htm.

[10] J. Gray. Why do computers stop and what can be done about it? In Proc. Symposium on Reliability in Distributed Software and Database Systems, pages 3-12, 1986.

[11] J. Gray. Locally served network computers. Microsoft Research white paper. February 1995. Available from http://research.microsoft.com/ gray.

[12] D. Scott. Making smart investments to reduce unplanned downtime. Tactical Guidelines Research Note TG-07-4033, Gartner Group, Stamford CT, 1999.

[13] D. Milojicic, A. Messer, J. Shau, G. Fu and A. Munoz. Increasing relevance of memory hardware errors - a case for recoverable programming models. In ACM SIGOPS European Workshop "Beyond the PC: New Challenges for the Operating System", Kolding, Denmark, Sept. 2000.

[14] G. Shegalov, G. Weikum, R. Barga, D. Lomet: EOS: Exactly-Once E-Service Middleware (Demo Description), Proceedings of International Conference on Very Large Data Bases, Hong Kong, China, 2002 (pp. 1043-1046).

[15] D. Scott. Operation Zero Downtime, a Gartner Group report, Donna Scott, 2000.

Business Operation Intelligence Research at HP Labs

Fabio Casati, Malu Castellanos, Umeshwar Dayal, Ming Hao, Mehmet Sayal, Ming-Chien Shan
Hewlett-Packard Labs
1501 Page Mill road, Palo Alto, CA, USA, 94304
{casati,malu,dayal,mhao,sayal,shan}@hpl.hp.com

Abstract

In the last decade we have witnessed an increased trend towards automating all the interactions that a company has with its customers, suppliers, and employees. Once a robust e-business infrastructure has been developed and deployed, companies quickly turn their attention to the problem of assessing and managing the quality of their e-business operations. This paper presents the work ongoing at HP Labs, aimed at enabling a simple, qualitative, and "intelligent" analysis of e-business operations.

1 Introduction and motivations

To increase revenue, improve efficiency, and reduce costs, companies deploy and integrate different kinds of software systems and applications that can automate and manage their business processes, within and across organizations. The resulting software architectures are typically complex, and include a variety of technologies and tools. The left side of Fig. 1 shows a quite common infrastructure that supports both the execution of internal business processes as well as B2B and B2C interactions. As the complexity of the system grows, the need for solutions that provide ubiquitous and automatic end-to-end management of business operations arises. In particular, business analysts/managers are faced with the need of quickly identifying problems affecting their operations, of assessing the impact of such problems in terms of quality of their business operations, and of understanding the root cause of the problems. In an ideal scenario, the entire e-business infrastructure should be self-managing, by automatically reacting to problems by executing corrective actions.

To address these needs, we are developing a set of applications that enable the analysis, monitoring, and optimization of e-business systems. The work involves the definition and implementation of measurement models and tools combining and aggregating raw operational data from each component, as well as the development of a Business Intelligence (BI) solution for historical, real time, and predictive analysis of the behavior of the entire e-business suite. We refer to this research and applicative area as Business Operation Intelligence (BOI), since it applies BI techniques to the management of business operations. The work presented here derives from earlier efforts focused in the workflow management domain, documented in [1, 2].

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

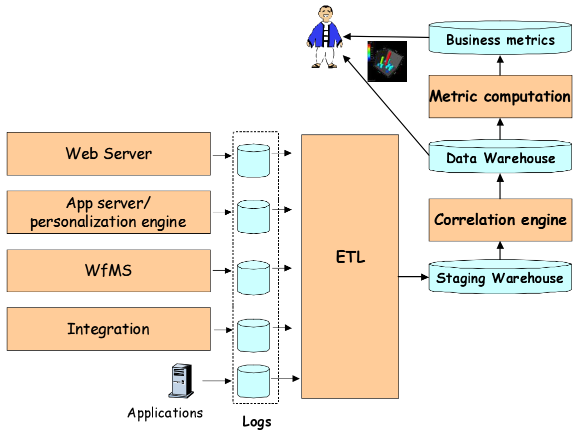

Figure 1: BOI architecture.

2 BOI Architecture

This section presents the overall approach and architecture of the BOI tool suite. In our architecture, we assume that either the individual components of the BOI platform have facilities for logging execution (meta)data to a file or a relational database, or that instrumentation has been put in place for this purpose. Once logged, data are periodically collected by an ETL application and inserted into a staging data warehouse. Data in the staging warehouse is then processed by the correlation engine, that inserts correlation information, to denote records logged by different systems that logically belong to the same business transaction. Once correlation information has been computed, the data is loaded into the warehouse. The warehouse is a very useful component in itself, providing a wide range of reporting functionalities that span across the whole e-business suite. Moreover, BOI aims not only at enabling a low-level, IT view of the collected information, but also at providing a higher-level, business view. In fact, while the warehouse can provide information about number of executions or average duration of each execution, business users are likely to need information at higher abstraction levels. For example, they may be interested in analyzing the cost or the quality of a business transaction, where cost or quality are user-defined criteria.

To enable and simplify the business-level analysis of low-level execution data collected by system logs, BOI provides users with a metric computation framework. This framework allows users to define business metrics, i.e., measurable properties that can be associated to data elements stored in the warehouse. The cost for a business transaction is an example of a business metric. Metrics are characterized by a definition and an implementation part. The metric definition includes the metric name and a data type, that can be Numeric, Boolean, or Enumeration (taxonomy). "Quality" is an example of a taxonomical metric, while cost or profit are examples of numeric metrics. Finally, the metric definition includes the specification of the target entity, i.e., the type of elements to which the metric is applied to (e.g., events, process instances, business transactions, web service operations). The metric implementation consists of a set of references to *mapping functions*, i.e., parametric code (typically SQL or Java) that can be executed on top of the warehouse and returns a numeric or Boolean value. For example, mapping functions could compute the number of nodes executed in a process instance or whether the execution of a web service operation has been faster than the average.

Within the metric implementation, it is possible to specify that a certain mapping function should be used to compute the metric for a given context. For example, users can specify that the cost for a process execution is dependent on the process duration, but only for instances of process "Fulfill order", while the cost for instances of process "approve proposal" is instead dependent on the number of nodes executed within the process instance.

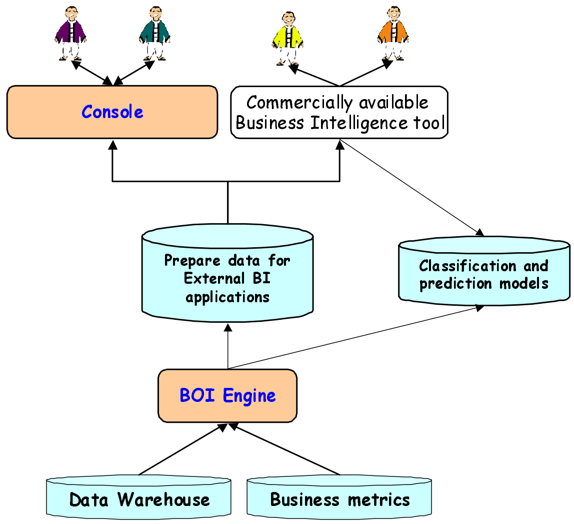

Figure 2: The BOI Engine prepares data for BI applications and generate classification and prediction models.

Hence, while the metric definition remains the same, its implementation actually depends on the specific element being analyzed. This enables users to get, at metric analysis time, a uniform view of otherwise heterogeneous properties. Once metrics and mappings have been defined, the metric computation engine executes the mapping functions to obtain the metric data (measurements). Such data can then be analyzed through either commercial reporting tools or though a built-in console, provided with the BOI solution, which among other features makes use of powerful visualization techniques.

3 Adding Intelligence and Management

The BOI metric framework, along with the data warehouse, can provide analysts with information about the business metric they consider relevant. While this is a very important aspect of the solution, it also naturally leads users to desire more advanced functionalities. In particular, users typically demand more "intelligent" information processing that can provide for example root cause analysis and prediction of user-defined business metrics. To provide these and other form of intelligent data analysis, BOI is equipped with a business intelligence engine that operates on top of both the warehouse and the metric data, and identifies explanations for the value of business metrics, prediction of the future value of such metrics, as well as service invocation patterns. The engine includes simple algorithms that perform basic analytics. In addition, it can also prepare data so that it can be consumed by a commercial data mining tool, such as Oracle Darwin or SAS. The approach towards generating analysis and prediction models out of metric data consists in applying data mining techniques to measured elements (process instances, operations, events, etc). In particular, analyzing metric data can be mapped to a classification problem, where measured elements constitute the training (labeled) set, and the different metric values (e.g., good or bad) are the classes. For each class, a classifier model is learned from the training set along with a set of classification rules, i.e., mappings from a condition on the objects' attributes to the class, with the meaning that objects whose attributes satisfy the condition belong to that class. These conditions become the basis for root cause analysis providing a deeper insight into the causes of the metric values and allowing the user to address such causes. A similar approach can be taken to perform predictions. We successfully applied this technique in several domains, and specifically in the context of workflow management [1].

In addition to providing information to users, BOI can interact with the system in order to take corrective actions and, in general, to manage system operations in reaction to the computed values of business metrics.

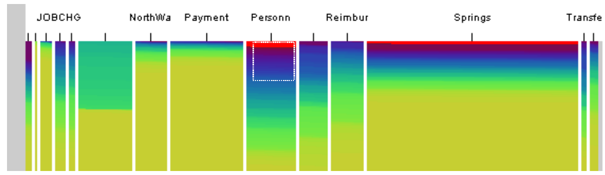

Figure 3: Analysis of process duration by process type.

System management in BOI is defined through simple policies, comprising a condition and an action. The condition is a Boolean expression over the value of metric data. The action can be one of the following: send a message to a JMS compliant bus, send an event to a business process, invoke a Java application, trigger BOI optimization algorithms, or send an email or SMS to a user. The sample policy shown below defines that once the completion of a process instance I is predicted to be late more than 3 hours, an event should be notified to the process instance. *IF predicted_completion_delay(I)> 03:00:00 THEN send_event(I,large_delay_predicted)*. How the event is handled depends on the business logic implemented by the process. The management engine only takes care of notifying the process.

4 Visualization

A key component of a BOI solution is *visualization*. The challenge here is how to visualize high volumes of data in a way that is easy to consume. Basic bar charts have been useful, but only show highly aggregated data. Therefore, we have developed a new visualization technique - pixel bar charts [3]. Pixel Bar Chart integrates the idea of bar charts with an X-Y diagram. The approach is to show each metric data item (e.g. the cost for each process instance) by a single pixel in the bar chart. The detail information of one attribute of each data item is encoded into the pixel *color*. The pixel bar chart technique has been prototyped in several business service applications at Hewlett Packard Laboratories. Fig. 3 illustrates 63,544 process instances executed within HP. Each pixel is a process instance. The x-axis is classified by process name; the y-axis is ordered based on the duration (which is the metric we are representing in the figure). The colors (from light to dark) in the different bars represent different duration time values in days.

Pixel bar chart, best viewed with colors, deliver both aggregate and detail information. Aggregation is performed visually by the human eye. At the same time, users can view the detail, since they have instance-level information, represented by the pixels. For example, this is useful to observe situations where the average duration happens to be very high, but where the average value is influenced by a small number of transactions being particularly problematic, while most transactions are just fine. With pixel techniques, this phenomenon can be spotted immediately. On a pixel bar chart, users can also select an area of the chart (possibly spanning across multiple bars), and then drill down, the semantics being that the next analysis will only focus on the selected pixels (instances).

References

[1] D. Grigori, F. Casati, U. Dayal, and M.C. Shan. Improving Business Process Quality through Exception Understanding, Prediction, and Prevention. Procs. of VLDB'01. Rome, Italy, Sept. 01.

[2] M.Sayal, F. Casati, U. Dayal, and M.C. Shan. Business Process Cockpit. Procs. of VLDB'02. Hong Kong, China, Aug. 02.

[3] D. Keim, M.C. Hao, U. Dayal. Pixel Bar Charts: A Visualization Technique for Very Large Multi-Attribute Data Sets. Information Visualization Journal. 2001

Cooperative Information Systems in Virtual Districts: the VISPO Approach

Enzo Colombo, Chiara Francalanci, Barbara Pernici, Pierluigi Plebani
Politecnico di Milano
Massimo Mecella
Università di Roma "La Sapienza"
Valeria De Antonellis, Michele Melchiori
Università di Brescia

Abstract

The VISPO project studies cooperative information systems in virtual districts to derive possible co-operation patterns and to provide an architectural framework for cooperation. Different organizations cooperate through e*-Services, which are available through a Service Oriented Architecture.*

1 Introduction

The last few years have witnessed several efforts for the development of service oriented environments, with the purpose of supporting the retrieval, composition, and execution of *e*-Services over the Internet [1, 2, 3].

Within the Italian VISPO (Virtual-district Internet-based Service PlatfOrm) project we aim to create a flexible environment to support the design of service-based cooperative processes and the dynamic composition of *e*-Services, by considering not only the design of complex *e*-Services starting from simple ones, but also the dynamic substitution of failed or modified *e*-Services.

A district is a consortium of independent member enterprises which operate in an integrated and organic way, often on the basis of personal and informal relationships, to exploit business opportunities. The aim of the project is to consider the application of information technology to such realities, in order to (semi-)automate the creation of business relationships (not necessarily based on geographical aggregation) and thus obtaining virtual districts.

In our model, cooperating organizations follow predefined process specifications when interacting. The structure of the process and the modalities of interaction are designed according to the nature of interactions in the virtual district, ranging from completely hierarchical structures to open markets [4].

Cooperative processes are defined as a composition of abstract services. Only at execution time, given an abstract service specification, a compatible concrete *e*-Service is retrieved and invoked. The VISPO architecture allows the execution of dynamically evolving cooperative processes based on these specifications. In particular, mechanisms for run-time substitution of *e*-Services with compatible ones and for adapting *e*-Services are being developed.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

This paper is organized as follows: Section 2 introduces the VISPO cooperation model. Section 3 presents the VISPO architecture for the composition and execution of cooperative processes based on *e*-Services, whereas Section 4 outlines the development environment.

2 VISPO Cooperation Model

Cooperative processes in VISPO are defined and enacted on the basis of cooperating *e*-Services. The following models are provided:

- an e-*Service Model*: the characteristics of *e*-Services are specified in terms of abstract services;
- an *Orchestration Model*: the interactions between *e*-Services and their coordination and enactment are specified;
- a *Business Transaction Model*: the process is designed on the basis of pre-defined business transaction patterns, which specify the characteristics of interactions between organizations and provided *e*-Services .

On the basis of such models, a cooperative process is designed on the basis of the business transaction model, which specifies how to assemble together different organizations in terms of hierarchical structures; the effective enactment of the cooperative process, to be carried out as coordination of different *e*-Services provided by the identified organizations, is specified through the orchestration model; finally the *e*-Services are designed and provided according to the *e*-Service model.

2.1 *e*-Service Model

Each *e*-Service is defined as a set of abstract services, each defined in terms of: *(i)* its interface, specified as the set of incoming and outgoing messages, and *(ii)* the admissible sequences of messages, in terms of state diagrams, specifying the possible sequence of pairs of input and output messages.

Each *e*-Service is specified in WSDL using port-types to define abstract services, and with an XML based description of its state diagram (WSTL) [5].

Compatibility and substitutability relationships among *e*-Services can be derived by analyzing their interfaces and behaviors [6]. A descriptor provides the condensed information needed for intelligent retrieval of compatible *e*-Services, and can be derived from the interface provided by the *e*-Service, in terms of incoming and outgoing messages and their parameters [7].

e-Services are registered in a registry, based on UDDI [8]; in addition, for each *e*-Service, the descriptor is provided.

2.2 Orchestration Model

The orchestration model specifies *(i)* the global evolution of a cooperative process involving two or more *e*-Services in terms of received and sent messages, and *(ii)* the association of global control task on the cooperative process. The orchestration model is based on Coloured Petri Nets, and distinguishes between control places, which are used for controlling the execution order of activities in an *e*-Service involved in the process, input/output places used to synchronize and compose messages, and orchestration places used to indicate transfer of responsibility on the overall process as the process evolves [9].

Cooperative processes need to be controlled and monitored by some organizations, but such a global control task cannot be assigned to a single organization for all the process duration, conversely it should be moved all along the process enactment. Therefore the model allows specifying the shifting of the global control task among organizations participating in the process in a controlled way, assuming that only one organization can

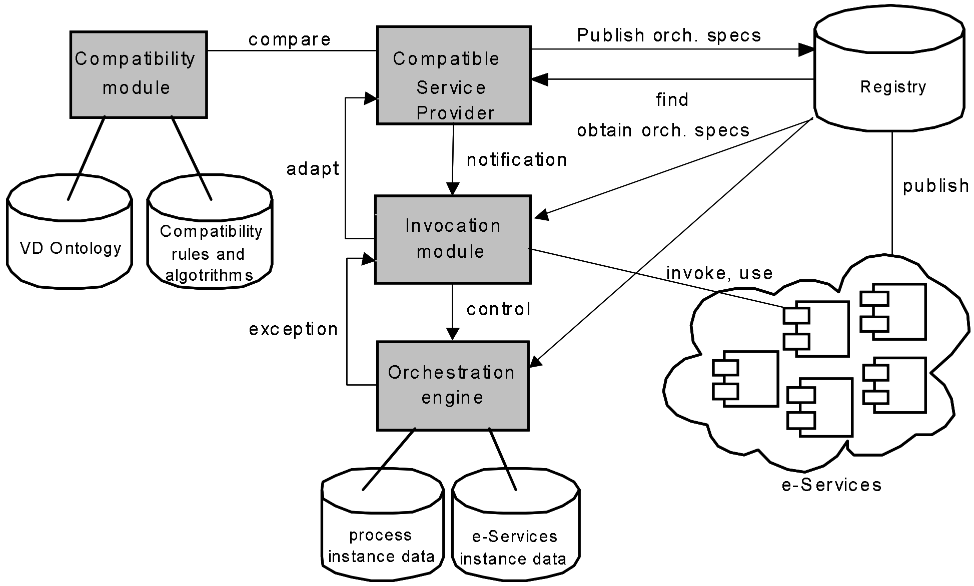

Figure 1: VISPO architecture

be responsible for controlling an instance of the process at a given time, while the task can move between organizations as the process evolves.

2.3 Business Transaction Model

The business transaction model is used to help in the definition of coordination and control of *e*-Services in a process. A set of business transaction patterns have been defined, according to a basic classification of possible interactions as market-oriented or hierarchy-oriented. In the first case, organizations (and provided services) are involved assuming a market context: services may be selected among the ones available, negotiation may be needed, post-settlement activities performed after service execution. Conversely in a hierarchical organization of the district, we assume that one organization coordinates all services, and negotiation is not performed when a service is used, but rather the contract conditions are established in advance; on the other hand, the coordinating organization can control the service execution while it is performed.

In order to support the different business transaction patterns, we assume that each *e*-Service, in addition to the abstract services it can provide, may be able to interact with other *e*-Services to negotiate its parameters and conditions for activation, and that it may provide quality control data on the services it is executing.

According to business transaction paradigms, patterns for messages used for coordination and control have been designed, and these messages must be provided by *e*-Services invocated in a given context. The basic transaction structures are used to design the orchestration for a cooperative process, including not only service execution, but also control and coordination message exchanges [10, 11].

3 VISPO Architecture

The VISPO architecture is illustrated in Figure 1. The environment for supporting cooperative process execution is composed of the following modules:

- *Registry*: the registry contains *e*-Services descriptions according to UDDI specifications, augmented with their descriptors extracted from the service interfaces for supporting *e*-Service retrieval. In addition, the registry stores the orchestration schemas for the cooperative processes in the virtual district, providing the characteristics of the service to be invocated to execute a process instance.

- *Orchestration Engine*: the orchestration engine controls the process execution following the orchestration specifications for the process obtained from the registry. The orchestration engine is composed by a set of local orchestration engines, one for each participating organization. Each local orchestration engine is responsible for local workflow enactment, and, when the organization is responsible for the whole process, the orchestration engine is also responsible for the global execution of the process and for composing output messages into input message for other *e*-Services according to the orchestration specifications.

- *Compatible Service Provider*: *e*-Services to be instantiated are chosen according to the specifications of services in the orchestration schema. The compatible service provider provides functionality to select services that, in terms of interface and state diagrams, are compatible with the description in the specification. The compatible service provider ranks compatible services on the basis of a domain ontology, which allows the tool to recognize similarities between services. A compatibility module evaluates similarities using descriptors.

- *Invocator*: it selects the *e*-Service to be invocated in a process, selecting from the compatible service list notified by the compatible service provider, and provides wrapping functionalities. Each *e*-Service is invocated in terms of concrete services it provides, and the mapping to actual parameters is performed by the invocator. A first implementation provides only wrapping facilities based on mapping of names of message or parameters, but more advanced functionalities are being studied.

4 Implementation

VISPO is being developed in a service-oriented implementation framework. Each component is a service which provides functionalities to other modules in the architecture. Currently, the IBM DB2 implementation of UDDI has been extended with API to support descriptor-based service retrieval, thus obtaining the Registry. The Compatible Service Provider interacts with the ARTEMIS tool [12] for ontology management with request-response interactions based on HTTP. The Invocator and Orchestrator have been implemented as a single module using IBM MQ Series Workflow [13] as a basis for the workflow engine. The Invocator interacts with the Compatible Service Provider over HTTP, receiving a list of compatible services for each *e*-Service to be invocated in the process.

5 Concluding Remarks

Future work in VISPO will concentrate in dealing with exception during process execution, to handle both failures local to organizations, and infrastructure failures on the basis of a peer-to-peer approach for interaction between architectural components. In addition, the invocation module will be extended with features to adapt available *e*-Services to given process specifications.

The work described in this paper will evolve in the framework of a new project, entitled MAIS (Multi-channel Adaptive Information Systems). The main focus in MAIS will be on providing *e*-Services on a multi-channel platform, and in studying mechanisms for transaction flexibility according to global parameters, such as network connections availability and bandwidth, middleware services, local computing and storage facilities, quality of service parameters.

Acknowledgments

This work has been partially supported by Italian MIUR through the "Fondo Strategico 2000" Project *VISPO (Virtual-district Internet-based Service PlatfOrm)* (`http://cube-si.elet.polimi.it/vispo/index.htm`) and the "FIRB 2001" Project *MAIS (Multi-channel Adaptive Information Systems).*

References

[1] J. Yang, W. Heuvel, and M. Papazoglou, "Tackling the Challenges of Service Composition in *e*-Marketplaces," in *Proceedings of the 12th International Workshop on Research Issues on Data Engineering: Engineering E-Commerce/E-Business Systems (RIDE-2EC 2002)*, San Jose, CA, USA, 2002.

[2] C. Bussler, R. Hull, S. McIlraith, B. Pernici, M. Orlowska, and J. Yang, Eds., *Proceedings of the CAISE 2002 Workshop on Web Services,* e-*Business, and the Semantic Web: Foundations, Models, Architecture, Engineering and Applications (WES 2002)*. Springer-Verlag LNCS, in press.

[3] F. Casati and M. Shan, "Dynamic and Adaptive Composition of *e*-Services," *Information Systems*, vol. 6, no. 3, 2001.

[4] O. Williamson, *Mechanisms of Governance*. Oxford University Press, Oxford, 1996.

[5] M. Mecella and B. Pernici, "Bulding Flexible and Cooperative Applications based on *e*-Services," submitted for publication, November 2002.

[6] M. Mecella, B. Pernici, and P. Craca, "Compatibility of *e*-Services in a Cooperative Multi-Platform Environment," in *Proceedings of the 2nd VLDB International Workshop on Technologies for* e-*Services (VLDB-TES 2001)*, Rome, Italy, 2001.

[7] V. D. Antonellis, M. Melchiori, and P. Plebani, "An approach to Web Service compatibility in cooperative process," in *Proceedings of Workshop on Service Oriented Computing (SOC)*, Orlando, 2003.

[8] UDDI.org, "UDDI Technical White Paper," http://www.uddi.org/pubs/lru_UDDI_Technical_Paper.pdf, 2001.

[9] M. Mecella, F. Parisi Presicce, and B. Pernici, "Modeling *e*-Service Orchestration Through Petri Nets," in *Proceedings of the 3rd VLDB International Workshop on Technologies for* e-*Services (VLDB-TES 2002)*, Hong Kong, Hong Kong SAR, China, 2002.

[10] E. Colombo, C. Francalanci, and B. Pernici, "Modeling coordination and control in cross-organizational workflows," in *Proceedings of the 10th International Conference on Cooperative Information Systems (CoopIS 2002)*, Irvine, CA, USA, 2002.

[11] E. Colombo and C. Francalanci, "Formalizing governance in virtual districts," in *Proceedings of Workshop on Service Oriented Computing (SOC)*, Orlando, FL, USA, Jan. 2003.

[12] S. Castano and V. D. Antonellis, "A schema analysis and reconciliation tool environment for heterogeneous databases," in *Proceedings of IDEAS'99 Int. Database Engineering and Application Symposium*, Montreal, Canada, August 1999.

[13] IBM, "MQSeries Workflow," http://www-3.ibm.com/software/ts/mqseries/workflow/, link checked November 2002.

Planning for Requests against Web Services

Mike Papazoglou,[1,2] Marco Aiello,[1] Marco Pistore[1] and Jian Yang[2]

1. DIT, University of Trento
Via Sommarive, 14
38050 Povo, Trento, Italy

2. INFOLAB, University of Tilburg
PO Box 90153
NL-5000 LE Tilburg, The Netherlands

{mikep,jian}@uvt.nl {aiellom,pistore}@dit.unitn.it

Abstract

One of the most serious challenges that web service enabled e-marketplaces face is the lack of formal support for expressing service requests against UDDI-resident web services in order to solve a complex business problem. We present a framework in which such a formalization is possible by integrating AI planning and constraint satisfaction techniques with web service technology. This framework is designed to handle and execute requests performing planning under uncertainty on the basis of refinement and revision as new service-related information is accumulated (via interaction with the user and UDDI) and as execution circumstances necessitate change.

1 Introduction

The current phase of the e-business revolution is driven by enterprises that look to B2B solutions to improve communications and provide a fast and efficient method of transacting with one another. E-marketplaces are the vehicles that provide the desired B2B functionality. An e-marketplace is an electronic trading community that brings multiple customers, suppliers, distributors and commerce service providers in any geographical location together to conduct business with each other through the exchange of XML based messages (over the Internet) in order to produce value for end-customers and for each other.

Industry-based (or *vertical*) e-marketplaces, e.g., semiconductors, chemicals, travel industry, and aerospace, provide to their members a unified view of sets of products and services and enable them to transact business using diverse mechanisms, such as web services. The goal of web services when used within the context of e-marketplaces is to enable business solutions by assembling and programming pre-built software components offering business functionality on the Web. Each of these components behaves like a self-contained, modular mini-application with its own interface described in the web service description language WSDL (`www.w3.org/TR/wsdl`) that can be published and invoked over the Internet. This allows companies to conduct electronic business, by invoking web services, with all partners in a marketplace rather than with just the ones with whom they have collaborative business agreements. Service offers are described in such a way, e.g., WSDL over UDDI (`www.uddi.org`), that they allow automated discovery to take place and offer request matching on functional and non-functional service capabilities.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

One of the biggest challenges that web service enabled e-marketplaces face is the lack of support for appropriate service request languages that retrieve and aggregate services which contribute to the solution of a business problem. Users typically require services from an e-marketplace based on their characteristics and functionality. Currently, there are no formal specification mechanisms for formulating user requests and no automated support for expressing requests over UDDI-resident services other than the primitive enquiry portion of the UDDI API. Requests based on the UDDI enquiry API are programmed within application programs and need serious re-coding efforts every time that there is need for a new request or an extension to a previous request. The baseline desiderata for a request language, at a much higher level of abstraction than what is currently offered by primitive UDDI APIs, are the following:

- *Genericity* The language must be generic in that it can be used for the same kind of services offered by different e-marketplaces.
- *Separation of goals* The language should distinguish between qualitative and quantitative goals. The former are more abstract general goals, e.g., a business process goal, the latter are the quantitative values tied to such goals, e.g., achieving a particular business process goal within a specified time-frame.
- *Reactiveness to non-determinism and change* The language must be able to express different sub-goals depending on events which are not known a priori and are not under the control of the requester, e.g., the requester does not know beforehand and cannot control whether a payment is going to be accepted, or the conditions of a particular instance of a web service.
- *Interoperability* The constructs of the request language should be directly mappable to existing web service standards and protocols. In other words, it should be possible to translate a request into a program interacting with web services as specified by current standards.

There are also a number of features, which would further enhance the usability and expressive power of the language. These include the following items:

1. The language could be enriched with the capability of distinguishing between different types of goals and sequencing preferences. A user needs to be able to state that one goal is preferable to another one, which in turn is preferable to a third one. For example, it should be possible to state that a goal is vital or that it is optional, and also express an explicit ordering of goals based on personal preferences.
2. The language could be enriched with similarity operators. There are three levels at which such semantic operators can work: **(a)** at the constraint objects level: a user could specify that s/he wants something similar to a compact car (topology), or some location close to a given city (proximity), and so on; **(b)** at the service level: a user could specify that s/he wants something functionally similar to a specific service, e.g., we may choose to replace a train service by a plane service; **(c)** at the plan level: a user could specify a goal similar to an already successfully executed plan, e.g., make a trip similar to the one the user has previously requested or done.

Our research concentrates on creating such a language and ensure an appropriate execution environment for it. For this purpose we do not rely solely on web service technologies, but also use AI planning techniques, which serve the purpose of expressing high-level goals and the handling of non-determinism and change, and constraint satisfaction techniques to ensure the satisfaction of the quantitative part of user requests. In [1] we introduced the basic constructs of this kind of language by integrating temporal constructs for expressing goal sequencing and constraints over quantitative values over the goals, in [2] we have moved one step further by completely specifying a request language, called XSRL, and described an execution environment for it. In this paper, we present the main constructs of XSRL, briefly overview its execution environment and give an example of an XSRL-based request.

2 Expressing requests against web services

In the following we use an application scenario in the business domain of e-travelling based on the specifications of the open travel agency (OTA, `www.opentravel.org`), where we consider an application booking flight segments and making hotel reservations for travellers. OTA has specified a set of standard business processes for searching for availability and booking a reservation in the airline, hotel and car rental industry, as well as the purchase of travel insurance in conjunction with these services. OTA specifications use XML for structured data messages to be exchanged over the Internet.

For the purposes of this paper we have chosen the OTA schema (document model) for an air segment reservation. This business process specifies the format of an air segment schema in XML (WSDL and BPEL) can be used as alternative representations) as well as the request and response formats for the air segment schema. The input (request) document necessary for the air flight segment includes departure and arrival airports, arrival and departure dates, desirable price ranges and seat numbers. The output document may include similar information but with actual destinations, dates and prices that are supplied after interacting with service providers. A comparable schema exists for hotel reservation purposes.

In Figure 1, we use an activity diagram to represent graphically the standard OTA air segment reservation business process. Formalising the OTA specification with an activity diagram is just one of many possible options, which is currently only used for illustration purposes. Solid arrows in the diagram represent flows of control while dashed arrows represent flow of messages. Each node in the diagram represents an activity. This formalism allows for cycles shown as outgoing arrows from end states. There are two business processes in this activity diagram, a travel agent business process and a tour operator business process. Business processes specify how agents, e.g., a booking application (user application in Figure 1), a travel agent, and a tour operator, in an e-travelling marketplace interact in order to satisfy a traveller's requests. The interaction between roles takes place as a choreographed set of business steps or actions that can be represented in BPEL.

In Figure 1, the business process is initiated by a traveller who provides the OTA request document necessary for an air flight segment (arrival airports, dates, seat quantity, etc). This triggers a request activity at some travel agent, which in turn triggers a package availability request for the from some tour operator. Air segment reservation information messages are exchanged until the traveller is able to choose between rejecting, modifying or accepting the air segment offered by the service providers. Potential users (travellers) can come up with requests to book their holidays on the basis of the business process described by this activity diagram. A similar process can be used for hotel reservation purposes but is not illustrated due to space limitations.

The request shown in Figure 2 expresses the wishes of a user who wants to travel from New York to a destination in Italy such as Rome or Venice. This traveler wants to use Alitalia or United Airlines as flight carrier, spend between 500 and 800 dollars per passenger, reserve 3 seats on the flight, leave on the 1st of June and return on the 10th. These are the input parameters required by the standard XML schemas and business process specification of the OTA marketplace. The traveler also requires accommodation for the same destination, in an hotel of the Hilton chain. Furthermore, the traveler finds it vital to have a flight ticket, but would still travel even if s/he did not acquire a hotel reservation. Obviously, the passenger does not wish to reserve accommodation without a confirmed flight ticket. These wishes are expressed in XSRL using the code snippet depicted in Figure 2 and are executed against the canonical air segment reservation process schema, which is represented graphically in Figure 1.

The interpretation of the request in Figure 2 follows. The "pure" request is specified in the part enclosed between the `<REQUEST>` tags. In the part enclosed between the `<GOAL>` tags the user specifies that s/he wishes to receive a confirmation with respect to a variable `$a`, and this goal is `<vital>` with respect to the request. If this succeeds `<then>` he is also willing to receive a confirmation for an hotel `$h`, but this is an `<optional>` goal (that is, if it fails, the whole request should still proceed). The portion of the request enclosed in the `<REQUEST>` tags defines the quantitative values tied to the various variables named in the goal portion and also specified the way the information is returned to the requester in case of success.

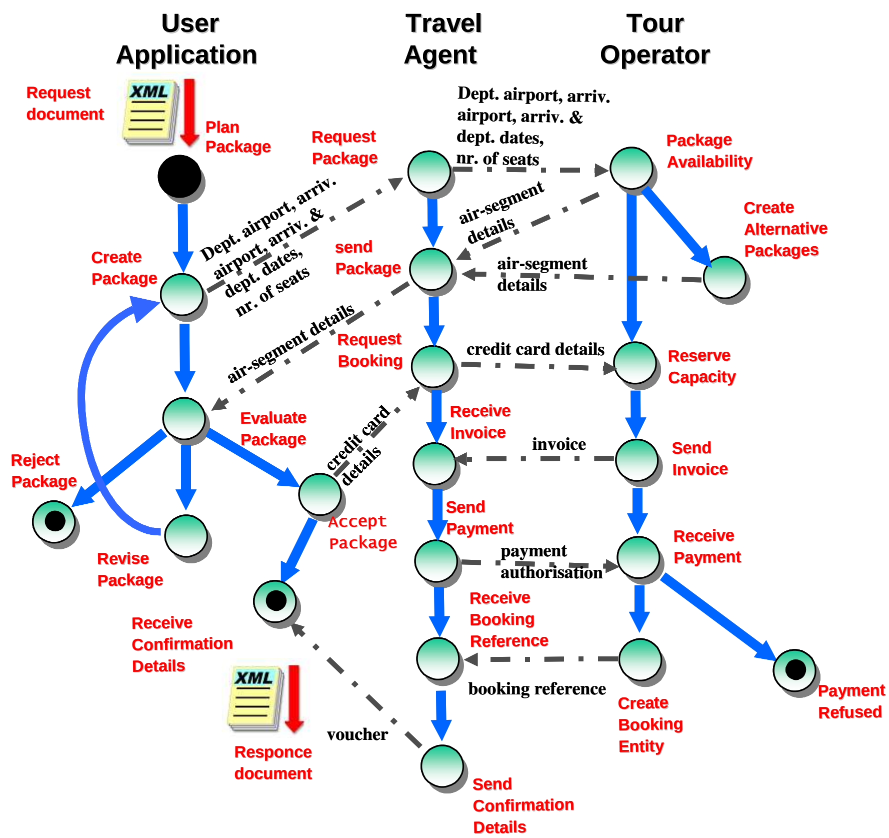

Figure 1: The AirSegment activity diagram.

3 The planning framework

The XSRL language has the properties of being generic, separating goals, dealing with change and non-determinism and of interoperability. We illustrate how these characteristics are achieved by briefly sketching the execution environment underlying the XSRL. The language is built on two inter-dependent components:

1. A "pure" *request specification component* that enables users to express a request in terms of the language constructs and enables complex formulations over these constructs,

2. and a *scheduling or goal component* that expresses user objectives (goals) as well as scheduling preferences and dependencies among the requested services.

The request specification component is specified in terms of XQuery constructs (`www.w3.org/TR/xquery`) and is combined with the scheduling part that is developed on the basis of the planning language $EaGLe$ [3], which it also extends. The idea is that one can describe activities and goals and compose them, and decide their sequencing. In addition, one can also use constructs that react to failure and non-determinism, such as "try to achieve a goal" or "if a subgoal fails then try to achieve an alternative subgoal." The language $EaGLe$ is fully implemented in a working prototype which comprises a planner MBP (model based planner). To work, the MBP needs an $EaGLe$ request expressed against a domain model. The domain model in the case of XSRL is a set of standard business processes. The descriptions of such processes are usually available in the case of web services in form of definitions based on modelling and specification languages such as BPEL (`www-106.ibm.com/developerworks/webservices/library/ws-bpel`) or BPML (`bpmi.org`). These need to

```
<XSRL>
   <REQUEST> {
      FOR $a in document(PkgTravelSegment.xml)//AirSegment
         [CarrierName =  "Alitlaia"| "United Airlines" AND
          DepartureAirport = "NewYork" AND
          ArrivalAirport = "Rome" | "Venice" AND
          (Price <= 800 AND Price >=500) AND
          SeatQty = 3 AND
          ArrivalDate = "1 June, 2002" AND
          DepartureDate = "10 June, 2002"]
      RETURN
         <ArrivalAirport>{ $a/ArrivalAirport}</ArrivalAirport>
         <price>{ $a/price}</price>
         <ArrivalDate>{ $a/ArrivalDate}</ArrivalDate>
         <DepartureDate>{ $a/DepartureDate}</DepartureDate>
         <HotelList> {
            FOR $h in document (hotelReference.xml)//HotelReference
               [ChainHotel = "Hilton"]
            WHERE ($h/Area =$a/ArrivalAirport AND
                   $h/HotelArrivalDate = $a/ArrivalDate + 1 AND
                   $h/HotelDepartureDate = $a/DepartureDate 1)
            RETURN
               <HotelName>{ $h/HotelName }</HotelName>
               <HotelAddress>{ $h/HotelAddress }</HotelAddress>
         }</HotelList>
   }</REQUEST>
   <GOAL>
      <Then><Vital>receive_confirmation($a)</Vital>
      <Optional> receive_confirmation ($h)</Optional></Then>
   </GOAL>
</XSRL>
```

Figure 2: An XSRL request.

be translated into a format understandable by the MBP planner ($NuPDDL$, a non deterministic extension of the Planning Domain Description Language, [4]). The output of MBP after a request is a program which represents the plan corresponding to the request of the user. This plan, named *generic plan*, is also encoded as an XML document.

The user specifies quantitative properties related to the business activities he desires to engage in. These are invoked via the UDDI registry in order to qualify the service requests. The responses of the web services contacted via the UDDI and the values provided in the request need to be matched. This is done by a constraint satisfaction solver, which takes all the typed numeric values and propagates the constraints. The result is a set of satisfied constraints, which are tied to a request and to a particular web service that partially satisfies it. These are then bind to the generic plan, yielding a set of plans that can potentially achieve the goals expressed in the request. We call a plan belonging to this set an *instantiated plan*.

Finally, the requester can inspect and change the instantiated plans via iXSRL: a set of interactive XSRL constructs that express acceptance, revision or rejection of instantiated plans. We refer the reader to [2] for further details.

Figure 3 provides a high-level view of the architecture for interpreting and executing XSRL requests. In this figure the user issues an XSRL request which is divided into its basic components. The goal portion is send to the MBP planner, which also needs a business process description. MBP generates a generic plan that interacts with the UDDI API to retrieve services from the UDDI registry. The values returned by web services offered by the UDDI registered providers are then matched with the constraints provided by the user via the XSRL request. In this manner we obtain a set of instantiated plans which are managed by the user via an iXSRL expression.

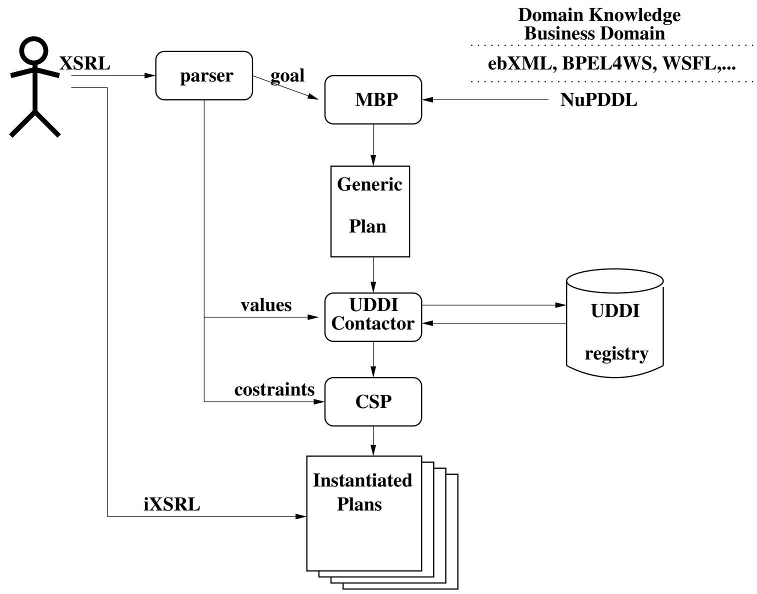

Figure 3: The XSRL request execution framework.

4 Concluding remarks

In this paper we highlighted an approach for web service interaction based on planning and constraint satisfaction and a service request language developed on the basis of this framework. The planning framework was developed on the basis of a coherent view of the issues arising when planning requests against web services under uncertainty (as plans inevitably do not execute as expected) in dynamic environments where there is the constant need to be able to identify critical decision trade-offs, revise goals and evaluate alternative options. This approach recognises that in an uncertain and dynamic world such as that of web services a correspondence must be drawn between the formal representation of a business domain model and the planer's model of it. It then instantiates plans on the basis of the plan model in terms of a user specific request and via interaction with the UDDI infrastructure; and when necessary, dynamically reconfigures plans on the basis of user interaction. These design considerations are reflected at the level of the request language that generates plans over web services residing in an e-marketplace and its run-time environment.

References

[1] M. Aiello, M. Papazoglou, J. Yang, M. Carman, M. Pistore, L. Serafini, and P. Traverso. A request language for web-services based on planning and constraint satisfaction. In *VLDB Workshop on Technologies for E-Services (TES02)*, 2002.

[2] M. Papazoglou, M. Aiello, M. Pistore, and J. Yang. XSRL: An XML web-service request language. Technical Report DIT-02-0079, Univ. of Trento, 2002.

[3] U. Dal Lago, M. Pistore, and P. Traverso. Planning with a language for extended goals. In *18th National Conference on Artificial Intelligence (AAAI-02)*, 2002.

[4] M. Ghallab, A. Howe, C. Knoblock, D. McDermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language. In R. Simmons, M. Veloso, and S. Smith, editors, *4th Int. Conf. on AI Planning and Scheduling*, 1998.

Definition and Execution of Composite Web Services: The SELF-SERV Project

Boualem Benatallah
School of Computer Science & Engineering
The University of New South Wales
Sydney NSW 2052, Australia
boualem@cse.unsw.edu.au

Marlon Dumas
Centre for IT Innovation
Queensland University of Technology
GPO Box 2434, Brisbane Q 4001, Australia
m.dumas@qut.edu.au

Zakaria Maamar
College of Information Systems
Zayed University
PO Box 19282, Dubai, U.A.E
zakaria.maamar@zu.ac.ae

Abstract

Web services composition is emerging as a promising technology for the effective automation of business-to-business collaborations. It allows organizations to form alliances by connecting their applications, databases, and systems, in order to offer "one-stops shops" for their customers. The SELF-SERV project aims at providing tool support and middleware infrastructure for the definition and execution of composite Web services. A major outcome of the project has been a prototype system in which Web services are declaratively composed, and the resulting composite services can be orchestrated either in a peer-to-peer or in a centralized way within a dynamic environment. Work is underway to extend this system in order to enable user-driven composition of Web services, and their execution in a mobile environment.

1 Introduction

The growth of Internet technologies has unleashed a wave of innovations that are having tremendous impact on the way organizations interact with their partners and customers. In particular, Web services composition is emerging as a new model for automated interactions among distributed and heterogeneous applications. This approach is an alternative to hand-coding an integrated application. A Web Service is defined as a modular and self-described application that uses standard Web technologies to interact with other services [8, 1]. An example of a Web service is bidding in an auction through a SOAP interface.

SELF-SERV is an ongoing research project that aims at providing tool support and middleware infrastructure for facilitating the composition of Web services in large, autonomous, heterogeneous, and dynamic environments. More precisely, the SELF-SERV system considers the following key aspects related to service

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

composition. Firstly, *declarative service composition* approaches are required to enable rapid development and flexible adaption of composite services. Secondly, the number of services to be composed may be large and continuously evolving. Consequently, approaches where the development of a composite service requires understanding and establishing interactions among component services at service-definition time are inappropriate. Instead, a divide-and-conquer approach should be adopted, whereby services providing similar capabilities (also called *alternative services*) are grouped together, and these groups take over some of the responsibilities of service composition. Thirdly, given the highly distributed nature of services, and the large number of network nodes that are capable of service execution, we believe that novel mechanisms involving scalable and completely decentralized execution of services will become increasingly important. Specifically, it should be possible to execute composite services under different communication topologies: peer-to-peer, centralized, and hybrid.

At present, the technological infrastructure for Web Services is structured around three major standards: SOAP, WSDL, and UDDI [4]. These standards provide building blocks for the description, discovery, and invocation of Web services. Other proposed standards such as BPEL4WS, WSCI, WS-Coordination, and WS-Transaction (`dev2dev.bea.com/techtrack/standards.jsp`), layer up functionality related to composition and transactions on top of the three basic standards.

The SELF-SERV system leverages these standards (especially SOAP, WSDL, and UDDI) and builds upon lessons learned in previous related work (e.g. eFlow [3], CMI [6]), in order to provide a scalable service middleware through which Web services can be declaratively composed, and the resulting composite services can be enacted according to different communication topologies within a dynamic environment. The rest of the paper provides an overview of the concepts (Section 2), architecture (Section 3), and planned extensions (Sections 4 and 5) of the SELF-SERV system.

2 Web Service Composition in SELF-SERV

A composite Web service is an umbrella structure that aggregates multiple other elementary and composite Web services, which interact according to a given process model. For example, a composite Web service "Travel Planner" may aggregate multiple Web services for flight booking, travel insurance, accommodation booking, car rental, itinerary planning, etc., which are executed sequentially or concurrently.

In SELF-SERV, the process model underlying a composite service is specified as a statechart [5] whose states are labeled with invocations to Web services, and whose transitions are labeled with events, conditions, and variable assignment operations. Statecharts possess a formal semantics and offer most of the constructs found in contemporary process modeling languages (sequence, branching, structured loops, concurrent threads, inter-thread synchronization, etc.). This ensures that the service composition mechanisms and orchestration techniques developed within the SELF-SERV project can be adapted to other process modeling languages for Web Services such as BPEL4WS and WSCI.

SELF-SERV exploits the concept of *service community* in order to address the issue of composing a potentially large and changing collection of Web services. Service communities are essentially containers of services that provide a common capability. They provide descriptions of desired services (e.g., flight booking) without referring to any actual provider (e.g., UA flight-booking Web service). When a community receives a request for executing an operation, it delegates it to one of its current members. The choice of the delegatee is based on a selection policy involving parameters of the request, the characteristics of the members, the history of past executions, and the status of ongoing executions. The set of members of a community can be fixed when the community is created, or it can be determined through a registration mechanism, thereby allowing service providers to join, quit, and reinstate the community at any time.

Communities are services themselves. In particular, their operations can be invoked by a composite Web service. In this way, dynamic provider selection and even some forms of dynamic planning can be achieved. Specifically, by labeling a state of a composite service with an invocation to an operation provided by a community, the selection of the actual service which is to execute this invocation, is delayed until the last possible moment, thereby allowing the decision to be taken according to the current execution status. Similarly, when there are several ways (or plans) for executing a "portion of a composite service", it is possible to select at run-time which of these plans is the most appropriate given the current context (e.g., choose the fastest way if time is important, or the cheapest otherwise). To achieve this, each of the alternative plans is modeled as a composite service, and all these composite services are grouped in a single community which can be invoked from within an encompassing composite service. The decision as to which of the available plans to adopt (i.e., which of the composite services in the communities should be invoked), is taken at the last possible moment.

The execution of a composite service in SELF-SERV is orchestrated by a collection of software components called *state coordinators*. One coordinator is attached to each state of a composite service. The coordinator of a state is in charge of initiating, controlling, and monitoring this state, as well as collaborating with its peers to orchestrate the composite service execution. In particular, the coordinator of a state S is responsible for determining: (i) when should S be entered for a given execution of the composite service; (ii) which other states may potentially need to be entered after S is exited and which data items should be sent to the coordinators of these states; and (iii) how to handle events such as cancellation or timeouts, while the service invocation labeling the state S is underway. The knowledge required at runtime by each of the coordinators of a composite service (e.g. location, peers, and control flow routing policies) is statically extracted from the composite service's statechart and represented in a simple tabular form (as *routing tables*). In this way, the coordinators do not need to implement any complex scheduling algorithm.

By distributing the responsibility of orchestrating a composite service across several state coordinators, SELF-SERV enables both peer-to-peer and centralized orchestration. Peer-to-peer orchestration is achieved by placing each state coordinator in the site of the Web service that is invoked when the corresponding state is entered. The state coordinators are therefore placed in different physical nodes (the sites of the components of the composite service), and they interact in a peer-to-peer way. Centralized orchestration is achieved by placing all the state coordinators in the host of the composite service. The collection of state coordinators placed in a single physical node can then be seen as a central scheduler, which remotely invokes the components of the composite service. Hybrid approaches can be achieved by placing some state coordinators in the host of the composite service provider, and others in the hosts of the Web services participating in the composition.

3 SELF-SERV Architecture and Implementation

The SELF-SERV architecture (Figure 1) consists of three modules, namely the *service builder*, the *service deployer*, and the *service discovery engine* [7]. These modules have been implemented using Java and the IBM Web Services Toolkit (`alphaworks.ibm.com/tech/webservicestoolkit`).

The service builder facilitates the creation and configuration of composite services and service communities. It provides an editor for describing registration modes and selection policies of service communities, as well as an editor for drawing and annotating the statechart diagrams of composite service operations. Annotated statecharts are translated into XML documents for subsequent processing by the service deployer.

The service deployer generates the routing tables of every state of a composite service statechart using the algorithms presented in [2]. The input of this generation process are statecharts represented in XML, while the outputs are routing tables formatted in XML as well. Once generated, routing tables are uploaded into the sites of the corresponding component services (for peer-to-peer orchestration), or into the site of the provider of the composite service (for centralized orchestration). Peer-to-peer orchestration requires in addition that the participating service providers download and configure a class called `Coordinator`, which implements the concept of state coordinator. This class contains the code required to load and interpret a routing tables, and to communicate with other coordinators in order to orchestration composite service executions. Centralised orchestration on the other hand does not require the service providers participating in the composition to install

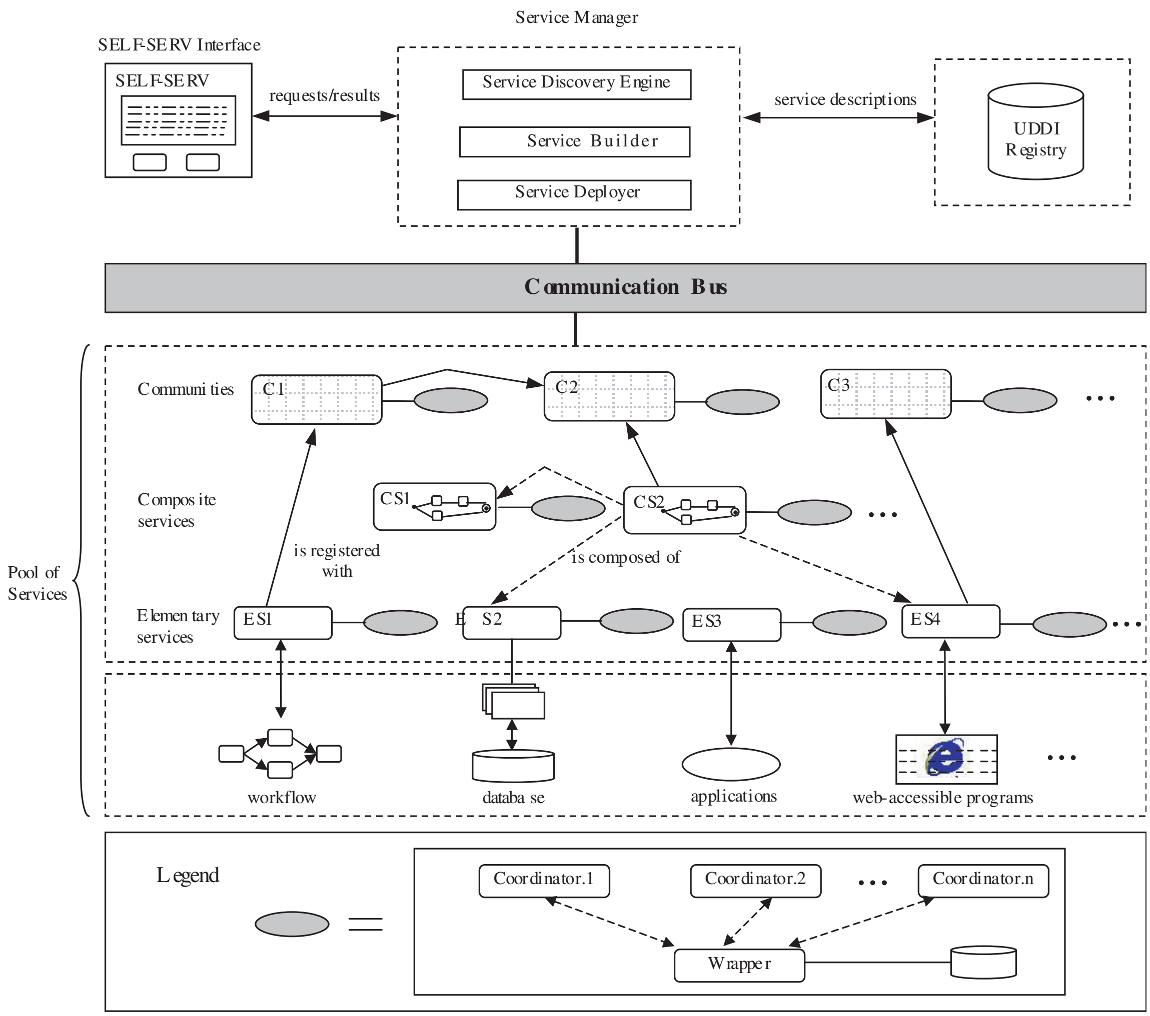

Figure 1: Architecture of SELF-SERV.

any SELF-SERV-specific runtime environment: they only need to provide their Web services through SOAP.

The SELF-SERV prototype has been used to experimentally compare the performance of the peer-to-peer orchestration model with respect to the centralized one. The experimental results have shown that peer-to-peer orchestration leads to less message exchanges in average, better load distribution among the participants, and as a result, lower execution times.

4 User-Driven Web Services Composition

Many Web sites that presently deliver their services through HTML Web pages, are likely to provide programmatic access to these services in the near future by exploiting emerging Web Services standards. Google, eBay, and Amazon are examples of popular Web sites that have already made some of their functionalities available through programmatic interfaces based on SOAP and WSDL[1]. As this trend continues to unfold, Web services with dual (interactive and programmatic) interfaces will become a currency. This will in turn open the possibility for dual navigation, whereby a set of interactions between a user and one or several Web sites can be assimilated to a composite Web service, and vice-versa.

[1]See `http://www.xmethods.com` for other examples of Web services with HTML-based interfaces that have been made available through SOAP/WSDL.

This idea is better explained through a simple scenario. A user opens the Web page of a stock quote site using his/her browser. (S)he enters a stock symbol in an input field, clicks a submit button, and obtains the latest quote for this stock in US Dollars. (S)he then opens a currency converter Web page, enters the quote obtained from the previous operation into an input field, selects the currency code "USD" in a menu, the currency code "AUD" in another menu, clicks the submit button, and obtains the stock quote in Australian Dollars. If the currency converter or the stock quote service are unavailable, (s)he will try using alternative ones. This sequence of navigation operations are expected to be repeated often, so the user would like to build his own Web page, that will allow him/her to enter a stock symbol in an input field, and obtain the quote in Australian Dollars in an output field, by reproducing the above process. An approach to achieve this outcome would be to map the sequence of navigation operations performed by the user to an equivalent composite Web service with two sequential states: one that invokes an operation over a community of stock quote services, and another one that invokes an operation over a community of currency converter services. The input of this composite Web service will be the stock symbol, and the output would be the quote in Australian Dollars.

More generally, in the context of Web services with dual interfaces, it is conceivable to specify composite Web services by dragging and dropping elements from several Web pages into a *composition panel*, and relating these elements through control and data flow relationships (e.g. the output of one element is given as input to another). With some simplifying restrictions, these manipulations can be performed by non-expert users, opening the door for the widespread use of composite Web service technology as a means to enable user-driven integration of Web services. An application of this type of integration can be found in account aggregation, where users perform repetitive navigation tasks in order to obtain a complete view of their accounts and investments.

Within SELF-SERV, we are developing a front-end tool to facilitate this kind of user-driven composition of Web services. Specifically, the tool will enable users to drag-and-drop elements from multiple Web pages (especially input and output fields within Web forms), in order to specify invocations to elementary services, which can then be grouped and linked together to form communities and composite Web services. In order to connect elements of Web pages to operations of Web services, SELF-SERV requires that the provider of services with dual interfaces defines links between the programmatic and the interactive interfaces. For example, a submit button in a Web form can be mapped to a Web service operation described in WSDL, while a form input element can be mapped to an input parameter of an operation. Such links can be expressed using RDF triples, where the "subjects" of the triples are XPath expressions designating an XHTML element such as a button or an input field, while the "objects" are Web service operations or input/output parameters.

Having specified a set of invocations to elementary services, the user can then group these services into communities. In the previous example, the currency converters could be grouped in a community, and a selection policy could be specified by providing an order over this set: the community would then invoke the first service in this order, unless it is unavailable, in which case the next service in the order would be selected, and so on.

The front-end composition tool will also allow the user to specify control and data-flow relationships between the elementary services and communities created in the previous steps by means of a visual language based on a subset of statecharts. Specifically, the control-flow operators provided by the composition tool will include sequence, choice, repeat, and parallel branching, while the data-flow will be specified through simple variable assignments and arithmetic expressions. The composition tool will then translate the resulting composite service specification into an annotated statechart, that can be enacted using a subset of the SELF-SERV environment. Since the composite service is intended to be used mainly by its creator, a client-side centralized orchestration model is appropriate, whereby the composite service is orchestrated from the user's machine.

5 Composite Web Services in Mobile Environments

The explosive growth of interconnected computing devices (e.g., PDAs, wireless technologies) enable new environments where ubiquitous information access will be a reality. More importanly, Web services will be accessible from mobile devices [9]. However, existing service provisioning techniques are inappropriate to cope with the requirements of these new environments. Several obstacles still hinder the seamless provisioning of Web services in mobile environments. Examples of such obstacles are: throughput and connectivity of wireless networks, limited computing resources of mobile devices, and risks of communication channel disconnections. We are extending the SELF-SERV architecture to support service provisioning in mobile environments. More precisely, we are investigating the following issues:

- *Context-sensitive service selection*: In addition to criteria such as monetary cost and execution time, service selection should consider the location of requesters and services, and the capabilities of computing resources on which services are executed (e.g., CPU, bandwith). This calls for context-aware service selection policies that enable the system to adapt itself to different computing and user requirements.
- *Handling disconnections during composite service execution*: In a mobile environment, disconnections are frequent (e.g., disconnection due to discharged battery, change of location, or to minimize communication cost). To cope with issues related to client or provider disconnection during a composite service execution, we are investigating an agent-based service composition middleware architecture. In this architecture, users and services are represented by delegate agents, which contain disconnection control policies related to requesters and providers. For example, a user delegate may be used to collect execution results during disconnection of the user's device, and returns these results to the user upon re-connection. Delegate agents reason and act upon evolution regarding user and service devise disconnections.

References

[1] B. Benatallah and F. Casati, editors. Special Issue on Web Services. *Distributed and Parallel Databases, An International Journal*, Kluwer Academic Publishers, 12(2-3), September 2002.

[2] B. Benatallah, M. Dumas, Q.Z. Sheng, and A. H.H. Ngu. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proceedings of the International Conference on Data Engineering (ICDE)*, pages 297–308, San Jose CA, USA, February 2002. IEEE Press.

[3] F. Casati and M.-C. Shan. Dynamic and adaptive composition of e-services. *Information Systems*, 26(3):143–162, May 2001.

[4] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, and S. Weerawarana. Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. *IEEE Internet Computing*, 6(2):86–93, March 2002.

[5] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[6] H. Schuster, D. Georgakopoulos, A. Cichocki, and D. Baker. Modeling and composing service-based and reference process-based multi-enterprise processes. In *Proc. of the Int. Conference on Advanced Information Systems Engineering (CAiSE)*, Stockholm, Sweden, June 2000. Springer Verlag.

[7] Q. Z. Sheng, B. Benatallah, M. Dumas, and E. Mak. SELF-SERV: A Platform for Rapid Composition of Web Services in a Peer-to-Peer Environment. In *Proc. of the 28th VLDB Conference (Demonstration Session)*, Hong Kong, China, August 2002. Morgan Kaufmann.

[8] G. Weikum, editor. Infrastructure for Advanced E-Services. *IEEE Data Engineering Bulletin*, 24(1), March 2001.

[9] S. J. Vaughas-Nicholas. Web services: Beyond the Hype. *IEEE Computer*, 35(2), February 2002

Model-driven Specification of Web Services Composition and Integration with Data-intensive Web Applications*

Marco Brambilla, Stefano Ceri, Sara Comai, Piero Fraternali, Ioana Manolescu
Dip. Elettronica e Informazione - Politecnico di Milano - I 20133 Milano, Italy
{mbrambil, ceri, comai, fraterna}@elet.polimi.it , Ioana.Manolescu@inria.fr

Abstract

The integration of Web services and Web applications offers challenging opportunities to academic and industrial research; the relevant issues include new methods, paradigms, and standards for supporting Web services and their composition. This paper focuses on the specification of Web applications that compose Web services in order to support arbitrarily complex processes. The proposed approach extends a high-level modeling language, called WebML, used for the specification of the front-end of Web applications. We develop two orthogonal extensions for the inclusion of Web services inside Web applications, accompanied by suitable protocols for message exchange, and the empowerment of Web modeling primitives with workflow capabilities. The combined use of these two features gives to WebML enough expressive power for specifying complex Web service interactions.

1 Introduction

In modern Web applications, most pages are dynamically composed from content extracted from corporate databases, and the business logic is based upon several interacting components. The design of ”data-intensive Web applications”, i.e., of Web applications making extensive use of databases, takes advantage of declarative Web site specification and modeling languages, such as WebML (http://www.webml.org), capable of producing high-level, unified descriptions of the data and front-end requirements of the application. While such specifications adequately cover the needs of applications developed within a single organization, they do not cover well the integration of Web applications with externally provided business logic. Nowadays, the concept of Web service is gaining popularity as a uniform interface for distributed software components, and several related standards and tools are being proposed for wide-scale interoperability.

This paper addresses the design and specification of data- and service-intensive Web applications, leveraging an existing visual language for modeling data-intensive Web sites, and applies the benefits of high-level, declarative specification to the combined use of data and services. In order to represent the Web services behavior, we provide a set of graphical primitives that represent the services and allow one to compose services and hypertexts in a diagrammatic representation, amenable to automatic code generation.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

*This research is part of the WebSI (Web Service Integration) project, funded by the EC in the Fifth Framework.

One of the main contributions of our work is to recognize that primitives for invoking Web services and primitives for composing them into complex processes are orthogonal. In our proposal, the former primitives reflect the Web Service Description Language standard(http://www.w3.org/TR/wsdl), while the latter primitives correspond to the Workflow Management Coalition model(http://www.wfmc.org). These extensions give WebML the ability to describe arbitrarily complex processes, possibly interacting with remote services, granting a significant increase of expressive power; the incorporation of new primitives is seamless, thanks to the extensibility of WebML.

2 Modeling data-intensive Web applications using WebML

Traditional data-intensive Web applications can be modeled using high-level specification languages. In this paper we present Web Modeling Language (WebML) [1] [2] [3], but the examples and results are independent of its particular notations and could be applied to other specification languages. WebML allows specifying a Web site on top of a database. Such a conceptual Web specification consists of a data schema, describing application data, and of one or more hypertexts, expressing the Web interface used to publish these data.
The data model. The WebML data model is the standard Entity-Relationship (E-R) model, widely used in general-purpose design tools. As an example, Figure 1 shows the E-R schema describing the database of the online computer store. Every entity has a unique identifier attribute, named ID, which is not shown.

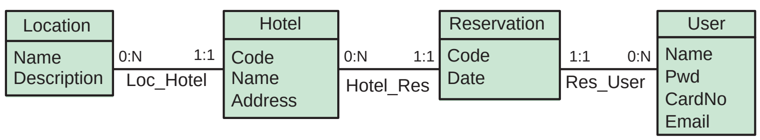

Figure 1: Sample data model for an online travel agent.

The hypertext model. Upon the same data model different hypertexts, called site views, can be defined, e.g., for different types of users or access devices. A site view is a graph of pages, which are presented on the Web. Pages enclose units, representing atomic pieces of information to be published; a unit selector specifies a predicate identifying the data to be extracted from the underlying database and displayed by the unit. Pages and units can be linked to form a hypertext.
Besides content publishing, WebML allows specifying operations, like the filling of a shopping cart or the update of the users' personal information. Basic data update operations are: the creation, modification and deletion of instances of an entity, or the creation and deletion of instances of a relationship. Operations do not display data and are placed outside of pages; special purpose operations can be specified, such as sending e-mail, login and logout, and e-payment.
Example. Figure 2 shows an instance of a simple online travel agent Web site, and the corresponding WebML graphical specification. The leftmost page shows a list of generic touristic locations, displayed as text anchors. The corresponding WebML model presents a Locations page enclosing an index unit named "Locations Index". When the user selects a location, he is taken to a page displaying data about the selected location and the list of related hotels. This target page is modeled in WebML enclosing a data unit ("Location Details"), displaying the attributes of the selected place, and an index unit displaying all the hotels. The Hotels page encloses an entry unit enabling the user to insert a new hotel; clicking on the "submit" button activates an operation chain composed by a create operation, generating a new hotel instance, and a connect operation, connecting the new hotel to the currently selected location.
The language includes several other units and operations, and is extensible with custom operations and units. WebML has a well-defined semantics that enables automatic generation of full-fledged, functional Web sites [2].

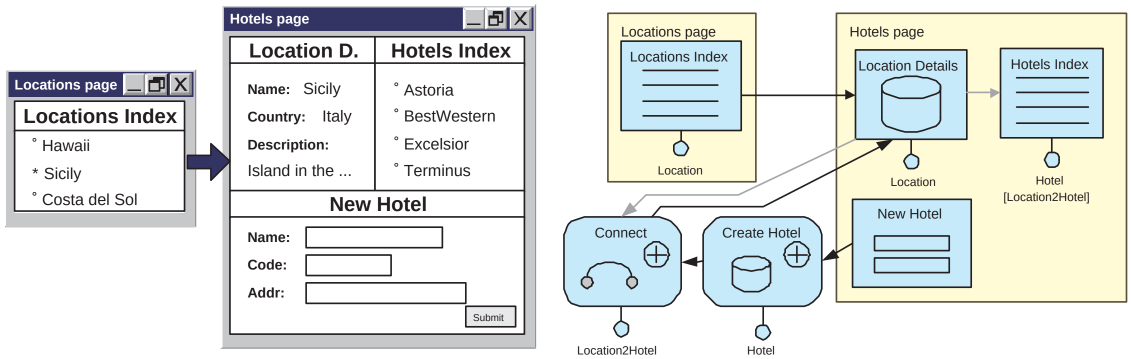

Figure 2: Hypertext fragment defined on the computer store data model.

3 Extensions for modeling Web services

According to WSDL , a Web service consists of operations that can be invoked following simple patterns; each operation may receive an input message from the Web application and may send an output message to it. Thus, the extension of WebML with support for web services requires:

- New operations for interacting with the Web service.
- Data extraction capabilities for transferring information from messages to the database hosted at the Web site and vice-versa.
- Support for synchronous and asynchronous bi-directional messaging.

The three requirements above are essential for the design and specification of a data- and service- intensive Web applications; fulfilling them is the novelty and the specific contribution of this paper. The main source of complexity arises from the fact that Web services are designed as building blocks for complex interactions among peers, which can be started by anyone of the peers. Web applications use instead the HTTP protocol, which is always initiated by user's requests. This mismatch has some consequences:

- By the nature of Web applications, we expect to see only a few cases of interactions started by the service (i.e., notification or solicit-response in WSDL).
- Most interactions with services included in Web applications are supposed to be synchronous, as imposed by the HTTP protocol, in which, after issuing a request, the client has to wait for the response. The solution adopted for asynchronous messages is to store them in a persistent manner and notify them to the user whenever he/she returns to the application.
- In a Web application, it is not possible to force the user to respond synchronously when solicited.

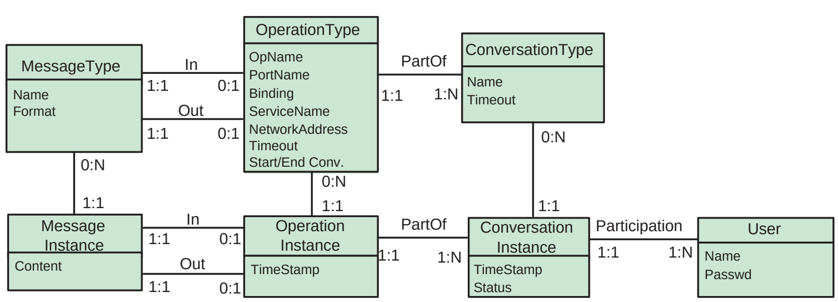

Figure 3: E-R diagram for user interactions with Web services.

Our solution provides (i) the data model used for recording the actual occurrences of conversations, operations, and exchanged messages, (ii) the WebML primitives describing WSDL operations, and (iii) the mapping languages for assembling data of the WebML application into XML messages and for disassembling the XML format of an output message into (E-R) data of the WebML application (not included here for space reasons).

Web Services Data Model. To record the interactions between the Web application and the Web services, we use a data schema, depicted in Figure 3. This schema is centered on the OperationInstance entity, which is described by the related OperationType; an operation may have one outgoing message and/or one incoming message, whose content is described as a text. Operations are part of conversations. Each conversation is a sequence of operations fulfilling a common goal, e.g., browsing a product catalog to buy some product. A conversation refers to a given user, and a user can participate to multiple conversations.

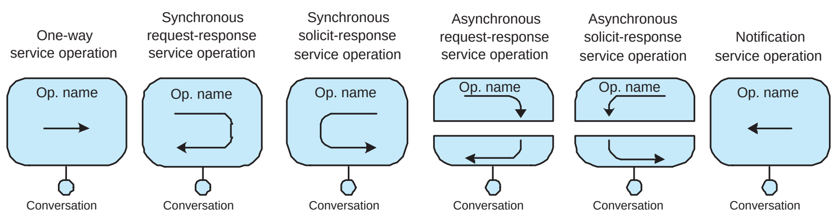

Figure 4: WebML primitives for communicating with Web services.

Web Services Hypertext Primitives. We propose six new graphical WebML operations reflecting the WSDL specification, shown in Figure 4. Operations belong to four categories: *one-way* (initiated by the user, by issuing an outbound message), *notification* (initiated by the service, by sending an inbound message), *request-response* (initiated by the user, with one outbound message followed by one inbound message), and *solicit-response* (initiated by the service, with one inbound message followed by one outbound message). Two-message operations are represented as round-trip arrows; arrows from left to right correspond to outbound messages (i.e., messages sent out by the Web application); asynchronous operations are represented by two superposed halves, one for each constituent message. The two parts of an asynchronous operation have independent lives, since they are triggered by different events and lead to different possible actions; therefore, each part can be considered as an independent WebML operation.

As any other operation, these new ones have input and output links, which are used to transfer context and to navigate within the hypertext. Four operations are triggered by the navigation of their input links: synchronous request-response, one-way operations, the upper half of an asynchronous response-request, and the lower half of asynchronous solicit-response. In all cases, input parameters and state information are assembled into messages, sent to the remote Web service; encoding of WebML parameters and state information into XML is specified by means of a simple mapping language. In the case of a synchronous request-response, the user waits until the response message is received, then continues navigation as indicated by the operation's output link. In the other three cases, the user resumes navigation immediately after the message is sent. Four operations are instead triggered by the reception of an external message: notification, synchronous solicit-response, the upper half of asynchronous solicit-response, and the lower half of asynchronous request-response. None of these operations can have output links leading to pages, since we cannot force the user to browse to a given page following an external event; instead, these operations can be linked to other WebML operations, carried out by the Web application without requiring the user's participation. The encoding of XML messages into state information is specified by means of a simple mapping language.

To express the boundaries of a conversation, the first operation of a conversation is tagged with a StartConversation property and the last operation is tagged with an EndConversation property. Eventually, a disposal mechanism can be provided, to terminate conversations when a timeout occurs. The execution of service-related

operations causes the implicit update of the data: a new OperationInstance is created with the proper parameters; it is connected to the OperationType and to the current Conversation by a PartOf relationship; for each message sent or received, Message instances are created and connected to the relative Operation instance; all operations sending a message include rules to construct the XML message from relational data; operations receiving a message include the symmetric rules for extracting data from the XML content of the message, to create new instances of the data model.
We now illustrate a Web application that includes interactions with Web services; Figure 5 describes some pages of the Web site of a travel agent offering to its costumers the possibility of getting weather forecasts and making hotel reservations. Both functionalities are provided by remote services, which the site integrates.

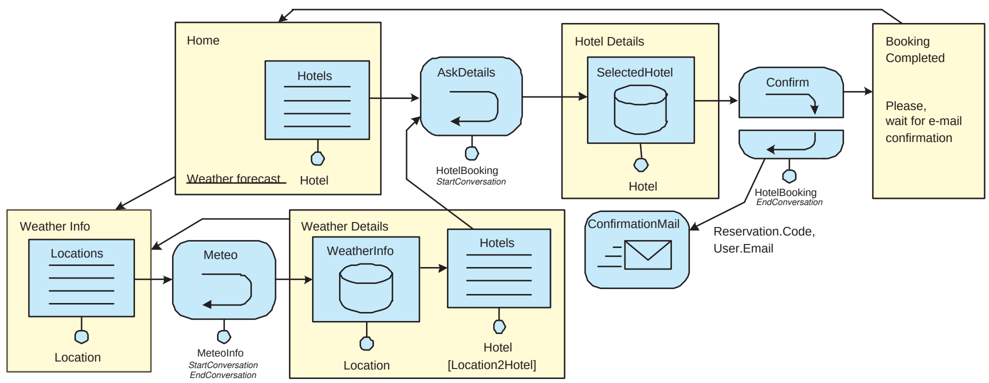

Figure 5: Hypertext model for the purchase application.

When a customer enters the Web site, he can choose a hotel from an index. More details about the chosen hotel are supplied by the AskDetail service, a synchronous request-response operation whose input message contains the hotel identity and whose output message contains details about the hotel. If the customer decides to make a reservation, he invokes the asynchronous request-response Confirm service; after the request, the buyer is immediately taken to the BookingCompleted page. When the response message is received, the reservation code is extracted from the output message of the Confirm operation, and an e-mail notification is automatically sent to the costumer by using the SendMail WebML operation. The user can also browse an index of locations and then request weather information for that location; such information is displayed in the WeatherDetails page, together with an index of hotels in that location. Users can next ask details about those hotels, and confirm reservations.

4 Extensions for modeling workflows

Pure workflow managements systems aim at imposing strict behavior constraints to their users, guiding them in the accomplishment of a particular objective; instead, conventional Web applications leave users completely free of of choosing their next activities in a navigation. We combine hypertext and workflow into what we call *workflow-driven hypertexts*: these hypertexts represent well-defined situations between the two ends of the spectrum, e.g., hypertexts representing well-defined processes, possibly with complex activity structure and execution constraints, but supported by the HTTP protocol, where each user can freely detach from the workflow and start independent browsing sessions. The main ingredients for the high-level specification of workflow-driven hypertexts are:

- A workflow data model for representing hypertext and workflow concepts, for managing access control, tracking the workflow activities and supporting logging and auditing.
- Constraint modeling primitives, for representing the execution sequence of activities. In particular, WfMC proposes: Sequences of activities, AND-split and join (a thread of control splits into two or more threads,

and joins when all the activities have been performed), OR-split and join (a thread of control makes a decision upon which branch to take, and joins when the activities of one of the branches terminate), Iteration (repetitive execution of one or more workflow activities), Pre-conditions and Post-conditions. The WebML primitives that we add enable the high-level specification of processes but are then mapped into lower-level hypertext specifications; conditional execution is modeled by *switch* operations in the hypertext.

Workflow Data Model. The workflow data model (Figure 6) includes the entities and relationships needed for managing one or more workflows. At the type level, it comprises Groups, ActivityTypes, and Processes; at the instance level, it contains Users, ActivityInstances, and Cases. ActivityInstances and Cases describe the actually occurred activities and include information about their start and end time. Application data can be added and connected at will to the Workflow data model entities by means of semantic relationships. A default one-to-many relationship may connect each ActivityInstance to application entities, with the meaning that entity instances are "assigned-to" a given activity. The default assignment of a given object (e.g., a Document) to a given activity instance indicates that the object is used (or will be used) by the activity instance (e.g., it will be created, read or modified by that activity). The default relationship linking an applicative entity to the ActivityInstance entity is denoted by adding a "W" symbol to the applicative entity in the E-R diagram (see Figure 6). If the default relationship is used, unit selectors over entities may be simplified with the shorthand shown in Figure 7, where the "W" symbol denotes a default predicate linking the entity to the relevant activity instance.

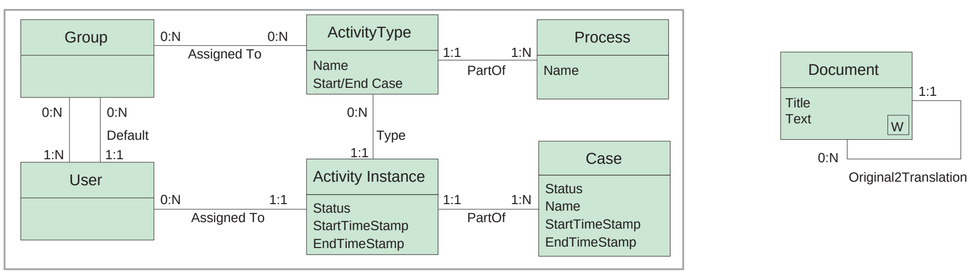

Figure 6: Workflow Data Model

Workflow Management Hypertext Primitives. The minimal extension required for supporting workflows includes operations for starting and ending activities. Operations *StartActivity* and *EndActivity* indicate the events at which operations start and terminate; their execution causes several updates to the workflow data.
The first activity of a process performs the creation of a new case; termination of either one of the "last" activities of a process may cause the termination of a case. Workflow specification also requires generic switch units for expressing conditional navigation. A switch unit evaluates, for each output link, a condition based on the values received by its input links, and enables the navigation of the first output link whose condition is true. This permits the modeling of workflow constraints, like AND and OR splits, the corresponding joins, iterations, pre- and post-conditions; we omit a further presentation of switch units for brevity. Additional units and default notations simplify the specification of workflows. Assign units connect application objects to the ActivityInstance they refer to using the default relationship, as discussed in the previous paragraph. A special "W" tag on a unit extracting application data expresses a conjunctive clause in the unit's selector, for selecting the data connected to the current ActivityInstance. Information about current activities and cases is available as WebML global parameters.
Figure 7 shows a simple application of the workflow concepts, to model a process for the creation and translation of documents. At high-level, the process is composed by two activities, creation and translation of documents, to be executed one after the other. The two activities are implemented as two hypertexts for the two different users involved in the process. The hypertext exploits the StartActivity, EndActivity and Assign operations. In

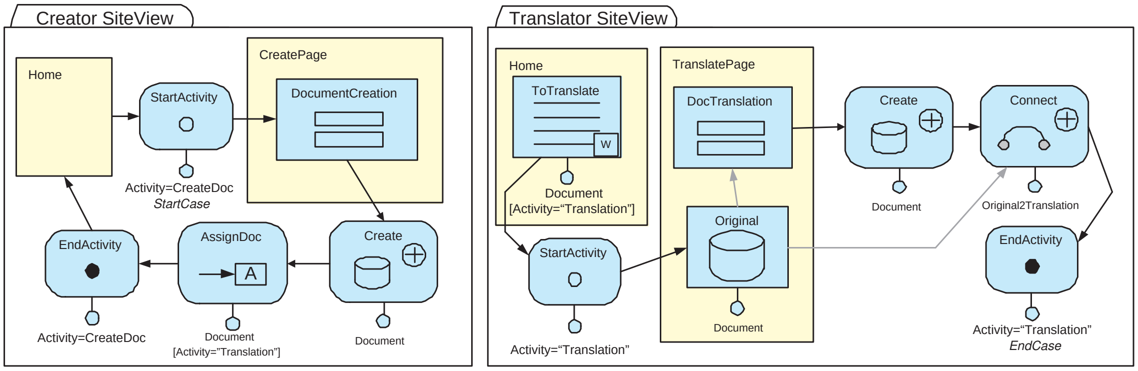

Figure 7: Hypertext implementing a simple workflow example.

particular, when a Creator enters the Home page of his/her siteview, he can start creating a new document by clicking on the proper link. The Create Document operation is the first of the process, thus the StartActivity is marked also as StartCase. The activity consists of the submission of a new document through a form in the CreatePage. The submission triggers the creation of the new document and the assignment to the Translation operation.
In his/her Home page, the translator can choose a document to translate, among the documents assigned to the Translation activity. Then, he can submit the new translation, which will be created as a new document, and connected to the original version. Once done, the activity and the current case are ended.

5 Workflows and Web services at work

The extensions proposed for Web services and workflows increase the expressive power of WebML and support the modeling of enhanced Web applications, incorporating Web services conversations and orchestrations. In fact, simple Web services conversations can be achieved through the use of the proposed Web services data model and hypertext primitives (as shown by the example in Figure 5). More complex conversations, involving branching on condition testing, parallel executions of Web services calls, and so on, can be obtained by integrating the two proposed extensions. In particular, it is always possible to define activities whose implementation consists of Web services calls; such activities, as parts of a complex process, can then be structured according to an arbitrary workflow, and can be executed depending on pre- or post-conditions.
Conclusions. This paper shows how visual specifications of Web applications can be integrated with Web services and workflows. These two extensions are orthogonal and require the introduction of new operations, which can be accomplished in a very natural way due to the extensibility of WebML. Experimentation is ongoing in the context of the WebRatio Web modeling and code generation tool (http://www.webratio.com).

References

[1] Stefano Ceri, Piero Fraternali, Aldo Bongio: Web Modeling Language (WebML): a modeling language for designing Web sites. *WWW9/Computer Networks*, 33(1-6), 2000: pp. 137-157.

[2] Stefano Ceri, Piero Fraternali, Aldo Bongio, Marco Brambilla, Sara Comai, Maristella Matera: Designing Data-Intensive Web Applications. Morgan-Kaufmann, December 2002, to appear.

[3] Stefano Ceri, Piero Fraternali, Maristella Matera: Conceptual Modeling of Data-Intensive Web Applications. *IEEE Internet Computing*, 6(4), July-August 2002, pp. 20-30.

The Role of Web Services in Information Search

Jens Graupmann, Gerhard Weikum
University of Saarland, Saarbruecken, Germany
{graupman,weikum}@cs.uni-sb.de

Abstract

State-of-the-art Web search engines are inherently limited in their abilities to search information in Deep Web beyond portals. This paper discusses how Web services and Semantic-Web-style ontologies can be leveraged for a new kind of Deep Web search tool. We argue that keyword based search should be replaced by a more expressive style of concept based querying, and we outline how portals can be automatically wrapped into Web services and how a focused crawl can automatically generate queries for exploring information beyond portal boundaries. As a first, simple demonstrator for the usefulness of the developed framework and tool suite, we briefly present an automatically generated meta portal about movies.

1 Introduction

The Web and its many information sources in Web accessible databases have the potential for being the world's largest knowledge base, encompassing all kinds of business oriented and scientific areas. The predominant way of searching this huge information wealth still is via Google-like Web search engines using keyword based queries. The inherent problems and limitations of this IR-style search paradigm are well known [DE00a], and the general hope is that the next generation of the Web will have technical means for boosting our abilities to search and find information. In particular, Web service technology is expected to improve the organization and accessibility of rich information sources. This paper discusses how Web services can indeed contribute to better search capabilities. More specifically, our approach aims to leverage three major trends that reflect the rapid evolution of the Web as a universal information infrastructure:

- The *Deep Web* (aka. Hidden Web) comprises all information that reside in autonomous databases behind portals, and this data cannot be reached by traditional crawlers [RG01]. While this phenomenon is known for many years, the growing number of Web portals with dynamically generated content and frequently changing hyperlinks aggravate the problem and pose a major challenge.
- The *Automated Web* (or Service Web) addresses the need for streamlining the programmed access to information and e-service portals using WSDL interfaces, as opposed to wrapping form based HTML pages in an ad hoc manner. While this technology is rooted in the area of B2B data exchange, it offers benefits for incorporating Deep Web sources into Web search tools and could eliminate the existing plethora of ad hoc portal wrappers and gateways. E-commerce sites like Amazon and even search engines like Google start offering Web service interfaces; however, standardized and more explicitly typed interfaces of this kind are only a first step towards "mastering" the Deep Web.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

- One problem that Web services do not address is that of semantic diversity. Information on the same subject may exhibit a wide diversity in terminology and structure. The *Semantic Web* [DE02a] aims to overcome these problems by putting emphasis on the adequate organization of Web data sources, using various kinds of ontologies as metadata frameworks for rich modeling and annotation of information. The most important languages for this purpose are RDF and DAML+OIL; another approach would be using carefully chosen XML tags for specific domains. However, despite the growing awareness of the need for semantic annotations, very few real Web sites are following one of these approaches so far.

Our main thesis explained in this paper is that we need to combine technologies from all of the above three trends in order to enable synergies and make significant progress towards better capabilities for Web search. The paper outlines our ongoing work on building a next generation search engine that leverages Web services for Deep Web access and ontologies for coping with semantic diversity.

2 From Keyword-based to Concept-based Search

Keyword search in the form supported by virtually all Web search engines is inherently inadequate for building a search tool that encompasses multiple portals, say for a meta portal on some specific domain. The reason is that the words that appear in a simple keyword list of a user query cannot be easily mapped to the various fields of a portal's query form or the various input parameters of a Web service. An automatic mapping is feasible only in the trivial case when the portal has a single input field that accepts a list of words. Likewise, advanced tools like focused crawlers [Cha02], which aim to automatically classify crawled Web pages based on training data for one or more topics, are inevitably stopped at portal boundaries (unless a portal offers an explicit link collection in addition to the usual query form). For proceeding beyond the portal the focused crawler would have to automatically generate meaningful words for the input parameters of the portal, and this has so far been perceived as too difficult for an automated tool.

As an example for the above problem and a motivation for our approach consider a user who is looking for the movie "The Caine Mutiny" but forgot the exact title and can only phrase the query "Caine" to some search tool. When the search tool attempts to map this query to the input parameters of some portal, it can at best generate the same kind of generic keyword search on the underlying database. A rich movie portal such as IMDB would probably return many movies starring the actor Michael Caine, leading to poor precision of the search result, and the user would have a hard time finding the requested information in this very large result. The irony of this poor outcome lies in the fact that most portals actually offer richer search capabilities with more specific interfaces for "typed" input parameters. In the example, most movie portals can probably distinguish between searching movie titles and actors or other film making crew. If the initial user query would have been something like "title = Caine", the search tool would have had a much better chance to map the query to appropriate input parameters of the portal; likewise, if the query had indeed requested information about Michael Caine then it should have been phrased in the form "actor = Caine".

In the following we refer to the above kind of refined querying as *concept-based* search. A query generally consists of one or more *concept-value pairs*; for example, "actor" denotes a concept and "Caine" is its value that we are interested in. A more advanced example, either on some computer science portal or the entire Web, would be the query "teaching = database systems & research = XML & country = Germany", looking for German researchers that teach database systems and are working on XML. Such query expressions resemble the capabilities of XML query languages to search for both element (or attribute) names and values [TW02]; the key difference is that we are employing this feature to search the Web, not just schematic sources.

Relative to keyword-based search, concept-based queries are more precise in expressing the actual information demand; however, the underlying data is still semantically diverse or even vague (after all, much of the information is in natural language). To cope with this problem we interpret the equality comparison of the form "concept = value" as a *semantic similarity* test: all concept-value pairs whose concept is closely related to the one in the query and whose value is closely related to the value in the query are considered as approximate matches, and a similarity score is computed for a ranked result list. For similarity testing and scoring we use an ontological network, which is essentially a weighted graph that captures semantic relationships like synonyms, hypernyms (i.e., broader terms), hyponyms (i.e., narrower terms), etc. and derives weights from the correlation of words in a large corpus like the Web [TW02].

3 Concept-based Meta Portal Construction

We are currently implementing concept-based Web search facilities in a comprehensive software framework that leverages Web services and ontologies and is particularly geared for Deep Web search. Our software enables the dynamic generation of meta portals on specific subjects (e.g., movies, gene expression data, or handcraft business). To this end we combine the focused crawler BINGO! that we have developed [SMS02, SBG03] with several other components for converting HTML pages into XML, creating Web service entries for HTML forms and effectively wrapping such HTML pages, searching ontologies for semantically related concepts, and automatially generating queries to Web services from the context of the focused crawl and the results of ontology lookups. Our approach proceeds in the following six steps:

- *Content gathering and query interface:* For creating a topic specific meta portal or when the user asks a concept-based query, we initiate a focused crawl. The crawl is driven by sample data that the user has to provide for training the classifier and as initial seeds for the crawl. When the user merely submits a query and does not provide anything else, our system first calls an external Web search engine and prompts the user to provide feedback on the results and thus select appropriate training data. This is a rather exceptional use case; the standard use would be to first create a meta portal by running the crawler and the other components for an extended time period and then submit queries. Queries are evaluated against the indexes that our system maintains about the crawled information, but when the user is not satisfied with the results she can additionally invoke the focused crawl mode with selected, particularly promising Web pages as seeds.
- *Portal discovery:* Whenever the focused crawler discovers a Web page that contains a form it tests, by looking up a UDDI registry (or using another standardized way for discovering Web service such as WSIL), if a page is already registered as a Web service.
- *Portal wrapping:* When a portal candidate is not yet registered as a Web service, a component is invoked that applies heuristic rules for generating a WSDL entry on the fly. Typically, highlighted text next to form fields will become parameter names, and the type of a form field determines the corresponding parameter type (e.g., an enumeration type for fields with a pulldown menu). The WSDL entry (and also a UDDI entry) can either be registered automatically, or can be placed in a candidate queue for later inspection and possible modification by a human.
- *Ontology searching:* For a dynamically generated portal, the parameter names of its WSDL interface are viewed as important concepts and inserted into a locally maintained ontology index. Furthermore, for enumeration types the possible values are likely to be in close semantic relationship to the corresponding concept and are also added to the ontology index. In addition, we search external ontologies like WordNet [Fel98] for newly found concepts and values, extract related words and their relationships, and add them to the ontology index, too. The external ontologies are manually registered with our software, and are themselves wrapped as Web services.
- *Portal query generation:* The focused crawl is driven by either a topic description, given in the form of training data, or an explicit user query. In either case it attempts to explore the data behind portals by automatically generating queries to the portals and indexing the returned result pages. The query generator attempts to match keywords from the training data or concept names from the user query with parameter names of the portal. If this fails, the local ontology index is searched for related terms and the matching

is reattempted. Once appropriate parameters have been identified for portal search, a limited number of parameter values are generated from the values in the query or topic-specific keywords, with preference given to words that have an is-instance-of semantic relationship to the parameter name.

- *Meta portal construction:* All the Web pages that are retrieved either directly on the Web or via automatically generated portal queries are added to the meta portal repository, with the usual kind of text and link indexing. For both static and dynamically generated HTML pages, a component is invoked that converts these pages into XML format, stripping away all the non-interpretable HTML parts (e.g., Javascript code) and attempting to generate meaningful tags (e.g., from table headings or words in highlighted fonts). All this content is automatically classified and ranked relative to the training data and/or user query, and the user can explore the data in an interactive manner.

Figure 1 depicts the main tasks of portal generation described above. It is important to realize that all the above steps heavily rely on heuristics. Our emphasis so far has been on setting up a comprehensive framework for federated Web search and meta portal construction. We are now starting to experiment with the various options that we support, and the next section briefly presents one of these experiments.

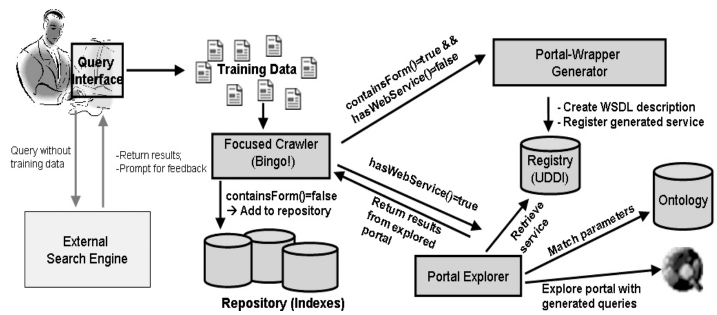

Figure 1: Portal Generation

4 Example Application

We implemented a prototype for an automatically generated meta portal about movies, called MIPS (Movie Information Portal and Search Assistant) that integrates static HTML pages as well as Deep Web sources. The focused crawler BINGO! [SMS02, SBG03] is used for gathering the content. BINGO! analyzes HTML documents (and also other data types like Word, PDF, etc.), computes feature vectors, and automatically classifies documents according to the user-specified topic hierarchy. Part of the analysis is the detection of form fields in the retrieved pages. When potentially relevant forms are found, the Web service Generator is invoked. The Web service Generator automatically creates a wrapper for the form, thus providing a query interface in WSDL. The new service is also added to a private UDDI registry. The system then attempts to classify the Web service into

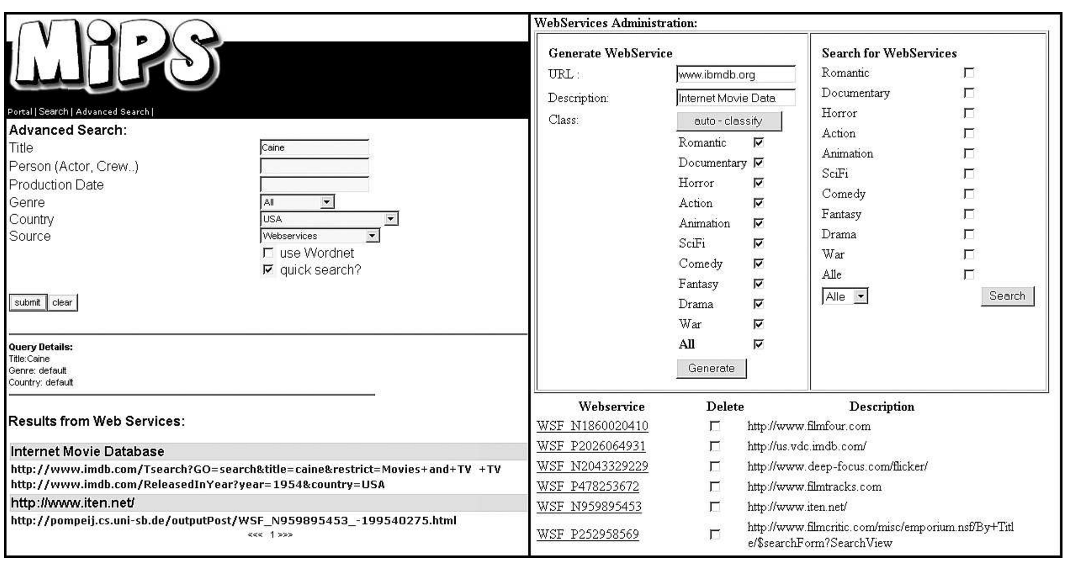

Figure 2: GUI of the MIPS meta portal

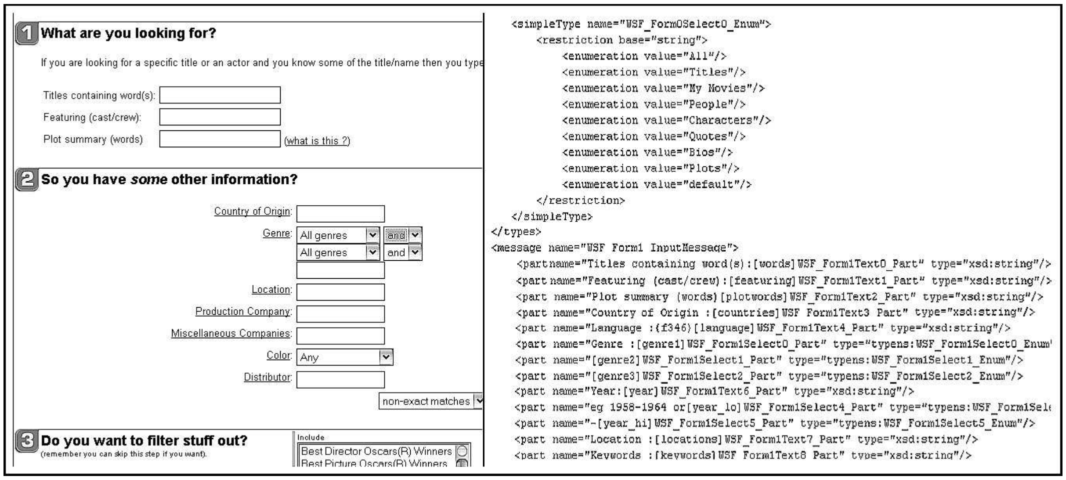

Figure 3: IMDB search form and generated WSDL file

movie genres, based on a list of available genres and small sets of representative movies as training data. On the basis of this information the system generates queries for each Web service and each genre. When the Web service returns results that fit with the training data of the inquired genre, the portal is added to the corresponding topic. If the Web service does not qualify for any genre it is removed from the database.

Figure 2 shows the search form and the Web service administration interface of our prototype. The search form, on the left, shows the result of the previously mentioned example query for "title=Caine", using only Web services. The administration interface, shown on the right, allows the user to manually add new or delete existing Web services and to invoke the automatic classification. Figure 3 shows an HTML form and the generated WSDL file for the interface of the generated Web service. The names of the method parameters of the generated Web service reflect the descriptors of the form fields and the data types correspond to the valid value assignments of the form fields (e.g. the values of drop down boxes are translated into XML enumeration types). This is important for the invocation of the service, because the arguments of the MIPS search form (Figure 2) are mapped to these parameters. For this purpose the system has its own ontology modeling synonym and hypernym relationships for the movie domain. For performance purposes this mapping is computed in advance and locally stored. For a given query the query processor first retrieves the best matches among the indexed static HTML pages. Then it queries the UDDI registry for matching Web services and invokes them with the precomputed parameter mapping and the appropriate values. Although the returned pages are built dynamically their URL usually includes all information to access the page from its source server. So dynamic pages can now be classified and added to the internal index along with their URLs. For subsequent queries the pages can be searched directly in the index, without calling the Web service.

5 Related Work

Our work has close relationships to several strategic research areas. With the Semantic Web direction we share our interest in ontologies and their role for data integration and schema matching across heterogeneous information sources (see, e.g., [DE02a, DE02b, MBD02]). In the area of Deep Web exploration, the work on wrapper generation (see, e.g., [BFG01, SA99, CGM02]) and on classification of "hidden" databases (see, e.g., [PGS01, RG01]) is highly related to our Web service generation and focused crawling approach. Web services and XML technologies for searching Web and intranet information are also investigated, for example, in [ABMT02, TW02, PV02] . Finally, in terms of the architecture our approach resembles work on federated service architectures like Object Globe [BKK01].

6 Conclusion and Future Work

Although our work is at a relatively early stage, we are confident that the pursued architecture provides an appropriate framework for next-generation search tools. In particular, the combination of Web services with ontologies and the focused crawler paradigm has a great potential that we have to investigate further in larger-scale applications and experiments. Near-future issues in our ongoing work include better forms of automatically tagging HTML and other weakly structured documents, to generate more meaningful and better searchable XML data. So far, our approach to this problem is based on relatively simple heuristic rules; we are considering the use of more advanced machine learning techniques for this purpose (see, e.g., [BDS01, DDH01]). With more data expressed in richly annotated XML form, we plan to carry over some of our prior work on XML querying with ranking [TW02] to a (Deep) Web setting. In particular, we consider adding similarity joins between heterogeneous, but semantically closely related sources to our repertoire of concept based search capabilities.

References

[ABMT02] S. Abiteboul, O. Benjelloun, I. Manolescu, T. Milo, R. Weber: Active Xml: Peer-to-Peer Data and Web Services, 28th Conference on Very Large Data Bases (VLDB 2002), 2002

[BDS01] V. Borkar, K. Deshmukh, S. Sarawagi: Automatic Segmentation of Text into Structured Records, ACM SIGMOD Conference (SIGMOD 2001), 2001

[BKK01] R. Braumandl et al.: ObjectGlobe: Ubiquitous Query Processing on the Internet, In: The VLDB Journal: Special Issue on E-Services, Vol. 10, No. 3, 2001

[BFG01] R. Baumgartner, S. Flesca, G. Gottlob: Visual Web Information Extraction with Lixto, 27th Conference on Very Large Data Bases (VLDB 2001), 2001

[Cha02] S. Chakrabarti: Mining the Web: Discovering Knowledge from Hypertext Data, Morgan Kaufmann, 2002

[CGM02] V.Crescenzi, G. Mecca, P. Merialdo: RoadRunner: Automatic Data Extraction from Data-Intensive Web Sites, ACM Sigmod Conference (SIGMOD 2002), 2002

[DDH01] A.Doan, P.Domingos, A. Halevy: Reconciling Schemas of Disparate Data Sources: A Machine-Learning Approach, ACM Sigmod Conference (SIGMOD 2001), 2001

[DE00a] Special Issue on Next-Generation Web Search, Data Enginieering Bulletin, Vol.23 No.3, 2000

[DE02a] Special Issue on Organizing and Discovering the Semantic Web, Data Enginieering Bulletin, Vol.25 No.1, 2002

[DE02b] Special Issue on Integration Management, Data Enginieering Bulletin, Vol.25 No.3, 2002

[Fel98] C. Fellbaum (Editor): WordNet: An Electronic Lexical Database, MIT Press, 1998

[MBD02] J. Madhavan, P.A. Bernstein, P. Domingos, A. Halevy. Representing and reasoning about mappings between domain models. In Eighteenth National Conference on Artificial Intelligence (AAAI'2002), 2002

[PGS01] G. Panagiotis, L. Gravano, M. Sahami: Probe, Count and Classify: Categorizing Hidden-Web Databases, ACM Sigmod Conference (SIGMOD 2001), 2001

[PV02] Y. Papakonstantinou, V. Vassalos: Architecture and Implementation of an XQuery-based Information Integration Platform, In: [DE02a], 2002

[RG01] S. Raghavan, H. Garcia-Molina: Crawling the Hidden Web , 27th International Conference on Very Large Data Bases (VLDB 2001), 2001

[SA99] A. Sahuguet, F. Azavant: Building light-weight wrappers for legacy Web data-sources using W4F, 25th Conference on Very Large Data Bases (VLDB 1999), 1999

[SBG03] S. Sizov, M. Biwer, J. Graupmann, S. Siersdorfer, M. Theobald, G. Weikum, P. Zimmer: The BINGO! System for Information Portal Generation and Expert Web Search, First Semiannual Conference on Innovative Data Systems Research (CIDR 2003), 2003

[SMS02] S. Sizov, M. Theobald, S. Siersdorfer, G. Weikum: BINGO!: Bookmark-Induced Gathering of Information, 3rd International Conference on Web Information Systems Engineering (WISE 2002), 2002

[TW02] A. Theobald, G. Weikum: The Index-based XXL Search Engine for Querying XML Data with Relevance Ranking, 8th International Conference on Extending Database Technology (EDBT 2002), 2002

Profiling and Service Delivery in Internet-enabled Cars

M. Cilia and A. P. Buchmann
Databases and Distributed Systems Group, Department of Computer Science
Darmstadt University of Technology - Darmstadt, Germany
<lastname>@informatik.tu-darmstadt.de

Abstract

This paper presents our experience applying web services and active database technology in an automotive scenario. Particularly, we concentrate on developing an infrastructure for customizing the user's experience in a vehicle. The motivation of this work is to allow the users to carry with them a personalized environment and apply their preferences not only to instrument adjustments and to maintenance and diagnostic information handling services but also to location-based services, infotainment and a briefing service that links to the user's office environment. We describe the infrastructure, the realization of the internal as well as the application services and discuss work in progress related to privacy and billing issues.

1 Introduction

Similar to other pervasive computing environments, cars will see a convergence of Internet, multimedia, wireless connectivity, consumer devices, and automotive electronics [5]. Assuming this context, wireless links between the car systems and the outside world open up a wide range of telematics applications. Automotive systems are no longer limited to information located on-board, but can benefit from a remote network and service infrastructure. Consider an automotive scenario, where vehicles, persons and devices have a web presence. Within this scenario new possibilities emerge, for example preferences can be stored and maintained in portals and then used for customization purposes in a uniform way. This avoids the known problem of defining many times your preferences in different systems. For illustration purposes just consider the adjustment of instruments in a vehicle. This can be done according to the personal settings stored in your portal. This approach not only provides the possibility of adjusting instruments of one vehicle, but the possibility of applying these settings (or at least part of them) to different cars. A frequent traveler can then use any rented car and it automatically adjusts its instruments (display of units of measurement, radio stations, car internal temperature, seat settings, etc.) according to the driver's preferences. But not only instruments can be adjusted, services can be personalized too. Services such as, "read my e-mail on the way to the office", "download/play my favorite music", "find and set the route to the next gas station", or "book an appointment to change oil" can take into account driver, car and company preferences.

This paper presents a case study of using an active functionality service for customizing user experience and invoking Web services. The active functionality, implemented as internal services, such as event detection service, notification service and action invocation service, is used to coordinate and invoke external, application-specific Web-services. Information integration is accomplished through the use of ontologies. This paper is organized as follows: In Section 2 the automotive scenario and a set of related services are described. In Section 3 the requirements imposed by this scenario are presented and the infrastructure we have developed is described. Section 4 presents the ongoing and future research.

Bulletin of the IEEE Computer Society Technical Committee on Data Engineering

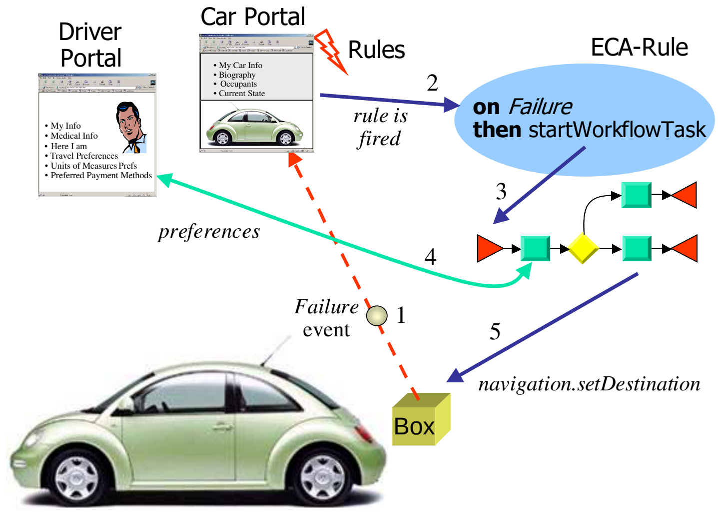

Figure 1: Car scenario

2 Scenario

This section describes a set of services that can be invoked as reaction to predefined situations according to the user's preferences.

Maintenance Services:

Part failure: The car detects a problem or reports a warning of a likely failure. A workflow is started to make the proper appointment (and depending on the severity of the problem) to find and set the route to the closest repair shop. This workflow takes into account the car's current geographical position, driver's (or car's) preferences. The workflow may also order a courtesy car or a taxi. Figure 1 illustrates this case.

Low fuel: When the car is running out of fuel a sensor signals this happening. As a reaction, a location service is invoked to find the next gas station considering current geographical position, destination and driver's preferences, e.g. the preferred vendor.

Adjustment of instruments: Once the driver gets into the car all her preferences (like, the units used for temperature, distance and velocity, the format of date/time, radio stations or music preferences, etc.) are automatically loaded. This is achieved because the driver uses her identification (e.g. badge, smartkey, smartcard) when getting into the car and as a reaction, driver's instrument preferences are read and the car's instruments are adjusted accordingly.

Briefing services:

Driving to work: Consider the case of a commuter and assume the following situation where the driver gets into the car. It is a workday and the current time is between 8:00am and 9:00am. As a reaction, the driver is requested to confirm the detected situation and the following set of actions can be performed: the best route to work is computed (avoiding traffic jams) and is passed on to the navigation service; today's scheduled meetings are checked; company news and other personalized news are obtained; and e-mails are read. Because drivers should concentrate on driving, all this information can be read out by using a text-to-speech service.

Location-dependent services:

Bilingual dictionary: When the car crosses the border of a country and taking into account the driver's profile (in particular spoken languages) a bilingual dictionary (and/or a currency convertor) is loaded in the driver's PDA and/or car's computer.

Restaurant locator: This service combines personal preferences, similar to well-known profiling services provided by portals like Yahoo!, but adjusted as a function of location. The internal ECA service can direct the choice to the on-board navigation service.

3 Infrastructure

The telematic scenario presented in the previous section illustrates many of the issues that should be considered when designing and building a customization infrastructure:

- Data heterogeneity: It is well-known that data from different sources can only be properly interpreted when sufficient context information about its intended meaning is known. For this reason, to exchange and process data from independent sources in a semantically meaningful way explicit information about its semantics in the form of additional metadata is required. [Preferences are applied to diverse vehicles.]
- Active mechanism: In many applications related to context-awareness the ”automatic” detection and response to context changes is desired. To accomplish this, active database technology can be applied in order to avoid hard-wired code scattered in various applications and sensors. The required functionality is made explicit in form of rules that are under the responsibility of a rule engine. [getting into the car .. the adjustment of instruments and services occur automatically; low fuel .. find a gas station]
- Event handling: Events identify happenings of interest (e.g. part failure or driver gets into the car). These events may be consumed from a set of applications subscribes to them. For that purpose a notification service with publish/subscribe capabilities and asynchronous communication support is required.
- Service invocation: The interaction with third-party services is mandatory in scenarios where it is intended to interact with different services the user is habituated to use.
- Profile management: The user profile must be managed in order to satisfy two needs: semantic data representation and privacy aspects. The first issue is also related with the first bullet where relevant customization data must be interpreted by diverse systems. The second relevant issue relates basically to access control. Because the profile contains sensible information the access to this data must be strictly controlled.

With these requirements in mind, an ontology-based infrastructure was proposed and developed. On its foundation it relies on a self-describing data model, MIX [1] which uses shared concepts (ontologies) expressed through common vocabularies as a basis for interpretation of data and metadata. This provides the possibility to support the explicit definition of context information which describes the semantics of exchanged data in the form of additional metadata. Additionally, cleanly integrated conversion functions can be used for converting data coming from diverse systems and services.

Ontologies in the infrastructure are organized in three levels: a) the basic level, where elementary ontology functionality and physical representation is defined; b) the infrastructure level, where basically concepts of the active functionality domain are specified; and c) the domain-specific level, where concepts of the subject domain (e.g. telematics) are defined. An *ontology service* is used to store, manage and provide a common access point to the concepts of the underlying ontologies.

A *notification service* based on a publish/subscribe paradigm is responsible for delivering events to interested consumers. This service provides asynchronous communications, it naturally decouples producers and consumers, it makes them anonymous to each other, it allows a dynamic number of publishers and subscribers, and it provides location transparency without requiring a name service. The notification service uses *concept-based addressing* in order to provide a higher and common level of abstraction to describe the interests of publishers and subscribers. This addressing approach is based on the subject-based addressing principles where the subject name space is hierarchically organized. Here concepts are mapped to the subject name space where the concept name and some of its attributes form part of the subject.

Events produced by different applications are integrated by *event adapters* that convert source-specific events into semantic events (represented by ontology-based concepts enriched with semantic contexts). Events are disseminated to interested consumers by using the notification service.

On this foundation an *active functionality service* was developed. But instead of applying traditional Event-Condition-Action (ECA) engines [6], the ECA-rule processing mechanism was decomposed into its elementary and autonomous services. These services are responsible for complex event detection, condition evaluation and action execution. The rule processing is then realized as a composition of these elementary services according to the rule definition. Interactions among elementary services involved in the processing of a rule are based on the notification service. Elementary services that interact with external systems or services use plug-ins for this purpose. Besides that, plug-ins know how to communicate with their respective services. They are also responsible for maintaining the target context of the system they interact with making possible the correct exchange of data. The active functionality service exposes a web service interface in order to provide to external clients the possibility to define, activate, deactivate, and remove ECA-rules. Figure 2 shows an abstract overview of the overall active functionality service. Details about this approach can be found in [2].

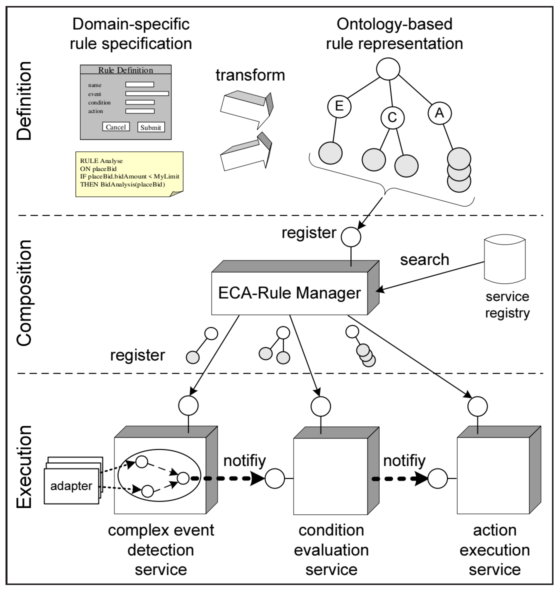

Figure 2: Overview of the active functionality service

3.1 Scenario-related Technology

Under the CoolTown model [4], people, places and things have a "web presence", that extends the "home page" concept to include all physical entities and to include automatic system-supported correlation of the home page or *point of web presence* with the physical entity. This web presence (or portal) provides current information and services relevant to its representative. This model supports nomadic users, based on the convergence of web technology, wireless networks and portable devices.

The profile manager functionality is incorporated to the portal manager. They can run at any server in the Internet or simply at a small device like a PDA or the car box.

It is assumed that vehicles are equipped with a GPS receiver and a box. This box plays the role of a mediator between the vehicle itself and the external world. It can access a vehicle's electronic and diagnostic interfaces, and it is responsible for announcing status changes (events) to its portal, keeping it (always) up to date. To take advantage of these changes/happenings the portal manager is extended with the ability to react to them. This is realized by integrating the active functionality service mentioned before. In this way, portals can provide the possibility to define ECA-rules that react according to happenings of interest and user preferences.

A prototype implementation of this telematic scenario was built on top of the proposed infrastructure [3].

4 Outlook

The active functionality service we developed allows the seamless integration of previously isolated automotive applications with Web services. It consumes the events generated by the car's internal sensors without the risk of interfering with them. It relays them to the manufacturer for maintenance advice or to the on-board rule mechanism for user guidance. These services can be realized immediately. The customization of the dashboard or creature comfort settings must be done strictly through the car manufacturers, since no manufacturer will risk an external system that is not under its control to modify vital components of the car. We are interacting on these issues with car manufacturers.

While car manufacturers are beginning to offer limited services, such as navigation support in response to traffic, these are closed systems. The infrastructure we developed complements the existing on-board systems with the possibility to add and combine new services, particularly the integration of the driver's or passenger's home or office environment and a wide infotainment offering.

At present one of the main issues is the source of payment of these services. Different payment models are emerging that range from one time payments included in the car's price to monthly subscriptions or pay-per-use. We are expanding the infrastructure to handle pay per use for Web services. Another issue we are currently addressing is privacy and security of the profile information. Since trust and human perception that sensitive data is under user control are key issues, we are experimenting with different forms of distribution of the information across external servers, the car's on-board box and the user's PDA. Finally, we are moving from a proprietary to an open platform based on J2EE and OSGi for the internal service architecture. The external services will continue to be accessed as Web services through the standard interfaces.

Acknowledgements

The authors would like to thank Peer Hasselmeyer, Rama Penumarthi and Christof Bornhövd for their collaboration with this project. This work has been partially supported by Hewlett-Packard Laboratories and Hewlett-Packard German Innovation Center.

References

[1] C. Bornhövd and A. Buchmann. A Prototype for Metadata-Based Integration of Internet Sources. In *Proceedings Intl. Conference on Advanced Information Systems Engineering (CAiSE)*, volume 1626 of *LNCS*, pages 439–445, Heidelberg, Germany, June 1999.

[2] M. Cilia, C. Bornhövd, and A. Buchmann. Moving Active Functionality from Centralized to Open Distributed Heterogeneous Environments. In *Proc. of CoopIS'01*, *LNCS* 2172, pages 195–210, Trento, Italy, September 2001. Springer.

[3] M. Cilia, P. Hasselmeyer, and A. Buchmann. Profiling and Internet Connectivity in Automotive Environments. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'02)*, pages 1071–1074, Hong-Kong, China, August 2002.

[4] T. Kindberg et al. People, Places, Things: Web Presence for the Real World. In *Proceedings of the Intl. Workshop on Mobile Computing Systems and Application (WMCSA 2000)*, Monterey, CA, 2000.

[5] U. Hansmann, L. Merk, M. Nicklous, and T. Stober. *Pervasive Computing Handbook*. Springer, 2001.

[6] N. Paton, editor. *Active Rules in Database Systems*. Springer, 1999.

ICDE 2003

CALL FOR PARTICIPATION

March 5 - 8th 2003

Bangalore, Karnataka, India

http://www.aztecsoft.com/icde2003

The **ICDE 2003 International Conference on Data Engineering** will be held in Bangalore, Karnataka, India, the center of Indian software activity with a variety of historical and natural attractions within striking distance.

ICDE 2003 aims at providing a premier forum for:

- presenting new research results
- exposing practicing engineers to evolving research, tools, and practices and providing them with an early opportunity to evaluate these
- exposing the research community to the problems of practical applications of data engineering
- promoting the exchange of data engineering technologies and experience among researchers and practicing engineers

Keynote Speakers

- Bruce Croft, UMass, USA
  Language Modeling for Information Retrieval
- Ramesh Jain, Georgia Tech, USA
  Event Based Management of Multimedia Data

Program

- 50 Technical Papers
- 30 Technical Posters
- 7 Tutorials
- 2 Panels
- Industry/Application Papers

Workshop

13th RIDE – Multi Lingual Information Management, March 10-11th, 2003, Hyderabad, INDIA.
http://www.iiit.net/conferences/ride2003.html

Tutorials

Application Servers and Associated Technologies
C. Mohan, IBM Almaden
Business Process Management Systems,
Anil K. Nori, Microsoft Corp., Former CTO (and founder) Asera Inc.
Database Technologies for eCommerce
Rakesh Agrawal, IBM Almaden
Data Engineering for Mobile and Wireless Access
Panos K. Chrysanthis, Univ. of Pittsburgh, Vijay Kumar, Univ. of Missouri- KC
Evaggelia Pitoura, University of Ioannina, Greece
Sequence Data Mining Techniques and Applications
Sunita Sarawagi, IIT Bombay
Storage and Retrieval of XML Data using Relational Databases
Surajit Chaudhuri, Microsoft Research, Kyuseok Shim, Seoul National University
Approximate Matching in XML
Sihem Amer-Yahia, Nick Koudas and Divesh Srivastava, AT&T Labs--Research

Panels

Cyberinfrastructure for Bioinformatics
Databases for ambient Intelligence

Visit Karnataka
A Land of
Heritage,
Monuments,
Temples,
Wildlife,
Sea coast,
and
Hill stations.

http://kstdc.nic.in/

General Co-chairs:
Deepak Phatak, IIT Bombay, India
Marek Rusinkiewicz, Telcordia, USA

Program Co-chairs:
Umesh Dayal, HP Laboratories, USA
Krithi Ramamritham, IIT Bombay, India

Industry Program Chairs:
C. Mohan, IBM Almaden Research Center
N. Ramani, HP Laboratories, India

Advanced Technology Seminars Chars:
Alex Buchmann, Tech. Univ. Darmstadt, Germany
S. Sudarshan, IIT Bombay, India
Vijay Kumar, Univ. Missouri, Kansas City, USA

Local Arrangements:
V. R. Govindarajan, Aztec Software, India
Jayant Haritsa, IISc. India

Publicity Chairs:
Fabio Casati, HP Laboratories, USA
Kamal Karlapalem, Intl.IIT, India
Kam-Yiu Lam, City Univ. HKSAR

Panels Chairs:
Sham Navathe, Georgia Tech., USA
Hongjun Lu, HKUST, HKSAR
Gerhard Weikum, Univ. Saarbruecken, USA

Demonstration Chairs:
Panos Chrysanthis, Univ. of Pittsburgh, USA
Srinath Srinivasa, IIIT, Bangalore, India

IEEE Computer Society
1730 Massachusetts Ave, NW
Washington, D.C. 20036-1903

Non-profit Org.
U.S. Postage
PAID
Silver Spring, MD
Permit 1398