A Framework for Constraint-based Collaborative Web Service Applications and a Travel Application Case Study

A. Hariharan* S. K. Prasad* A.G. Bourgeois*
E. Dogdu* S. Navathe** R. Sunderraman*
Y. Pan*

**Department of Computer Science,Georgia State University, Atlanta*
*** College of Computing,Georgia Institute of Technology, Atlanta*
E-mail: {sprasad, abourgeois, edogdu, rsunderraman, ypan}@cs.gsu.edu

Abstract

Future Web applications will be more collaborative, and will use the standard and ubiquitous Internet protocols. Independently developed applications will have to be integrated seamlessly despite their heterogeneous origins. Heterogeneity stems from programming languages, development environments, operating systems, and host devices. It can be addressed by providing means of composing (or reusing) existing application functionalities in the form of Web services. This works toward the goal of developing new Web applications and processes with limited programming effort. We have developed System on Mobile Devices (SyD) middleware for rapidly developing and deploying collaborative applications over heterogeneous and possibly mobile devices. SyDLink is a key module of SyD; it enables creation of coordination links (Web bonds) between Web services (independently developed applications and application components). SyDLink-enabled entities allow high-level description and enforcement of application constraints among application components. Using such entities, a collaborative application can be created rapidly with limited coding. To illustrate the concepts and implementation of SydLink-enabled collaborative applications, we present a travel reservation system as a case study to show how such an application can be developed and deployed quickly. The travel reservation system is put together by creating SyD coordination links between the independent flight, car, and hotel reservation Web services.

1. Introduction

A typical application is no longer a stand-alone component as there is usually a need to access other applications and to modify data held by other applications. Enabling non-technical users to be able to easily and rapidly compose and link existing Web services to create ad-hoc applications is a long-term goal [1]. This paper addresses this crucial issue for technical users and presents a novel way to integrate existing Web services. We leverage off of legacy Web services and enable them to coordinate with each other to perform seamless transactions across the various entities. The current technology available to develop such collaborative applications over a set of wired or wireless devices and networks has several limitations [2]. System on Mobile Devices (SyD) is a middleware technology that addresses the key problems of heterogeneity of device, data format and network, and that of mobility [3, 10]. We employ this middleware and expand the capabilities of its coordination module, namely, the SyDLink module, and demonstrate the development of a new collaborative Web service application for travel reservations. A key module of the SyD middleware, SyDLink enables an application to create and enforce interdependencies, constraints and automatic updates among entities.

We also provide a Java Application Programming Interface (API) employing which developers can create domain-specific collaborative applications based on Web services without much effort. The API not only provides methods to integrate independent Web services, but also to preserve business rules across them. We achieve this by forming and maintaining "live" SyD coordination links among the various entities. By "live", we mean that any change made in

any one of the entities involved in the link would automatically trigger events to handle interdependencies among the rest of the entities. In the current prototype of SyDLink module, a coordination link is specified by embedding triggers originating from a particular method invocation of a source Web service to specific methods on a target set of Web services, along with a global logic/constraint to be enforced among these services. The API and the associated development environment that we developed provides methods and functions to: (i) read in a WSDL link and parse it, (ii) list methods (or Web services) that the application developer can have access to, (iii) set business rules (like constraints, such as budget, deadlines, etc), (iv) maintain global constraints, and (v) automatically trigger events based on certain conditions. As we demonstrate, by using this API, the application developer is left with minimal coding. The necessary coding would involve application logic that cannot be expressed by SyD coordination links (also referred to as Web coordination bonds in the context of Web services [1]) and the necessary GUI needed for the applications. We also provide simple GUIs with which the developer can establish the database and tables for SyD links. The developer is also be able to specify global constraints and other relevant details for the initial set up.

The rest of the paper provides a brief introduction to the SyD middleware and the advantages of using this middleware in the next section. Section 3 contains an overview of our Web service coordination architecture by using a travel application as an example. Section 4 discusses the SyDLink module and demonstrates how Web services can be integrated. Section 5 discusses some related work and Section 6 includes the conclusion and future work.

2. Overview of SyD Middleware Technology

System on Mobile Devices (SyD) middleware [3,9,10,12,13] for collaborative applications is a new technology that addresses the key problems of heterogeneity of device, data format and network, and that of mobility. SyD combines ease of application development, mobility of code, application, data and users, independence from network and geographical location, and the scalability required of large enterprise applications, concurrently with the small footprint required by handheld devices. SyD uses the simple yet powerful idea of separating device management from management of groups of users and/or data stores. Each device is managed by a SyD deviceware that encapsulates it to present a uniform and persistent object view of the device data and methods. Groups of SyD devices are managed by the SyD groupware that brokers all inter-device activities, and presents a uniform world-view to the SyD application to be developed and executed on. All objects hosted by each device are published with the SyD groupware directory service that enables SyD applications to dynamically form groups of objects hosted by devices, and operate on them in a manner independent of devices, data, and underlying networks. The SyD groupware hosts the application and other middleware objects, and provides a powerful set of services for directory and group management, and for performing group communication and other functionalities across multiple devices.

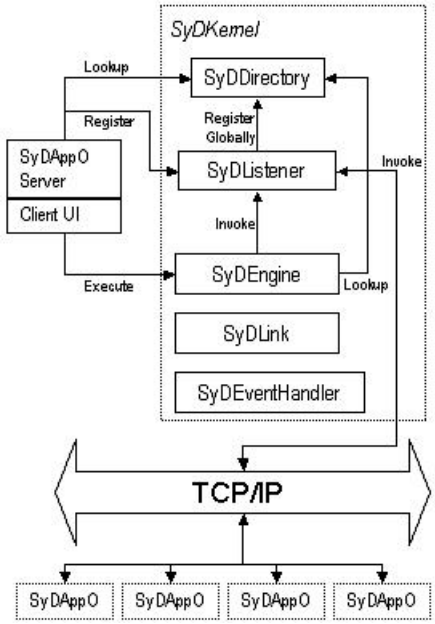

Figure 1: Interactions among modules of SyD kernel

The SyD Kernel includes the following five modules:

1. **SyDDirectory**: Provides user/group/service publishing, management, and lookup services to SyD users and device objects. Also supports intelligent proxy maintenance for users/devices.
2. **SyDListener**: It provides a uniform object view of device services, and receives and responds to clients' synchronous or asynchronous XML-based remote invocations of those services [14]. It also allows SyD

device objects to publish their services locally to the listener and also globally through the directory service.

3. **SyDEngine**: Allows users/clients to invoke individual or group services remotely via XML-based messaging and aggregates responses. This yields a basic composer of mobile Web services.
4. **SyDLink**: Enables an application to create and enforce interdependencies, constraints and automatic updates among groups of SyD entities and Web Services [1,2,9].
5. **SyDEventHandler**: This module handles local and global event registration, monitoring, and triggering.

3. Overview of Web Service Coordination Architecture and a Travel Application

A coordination framework over Web services should allow integration of Web services' methods and embedding of interdependencies among various entities, and provide the user with the appearance of one seamless transaction that may actually consist of several sub-transactions. Figure 2 depicts the overview of our Web service coordination architecture. The user will see a single application that will invoke methods across multiple Web services and provide a single result. Bridging the gap between the user's collaborative applications and the legacy Web services and similar atomic applications is the SyDLink module and its infrastructure including its coordination link database. The architecture shown in Figure 2 is specific to a travel application that we have developed, and we will use it here on as a running example.

The travel application allows for automatic rescheduling and cancellation of itineraries. Once an itinerary is decided and the trip is planned for the user, links that are created are maintained in the user's SyDLink database. This way, the itinerary is still "alive," meaning a global relationship is maintained over these Web services. The links are maintained among the services until the itinerary expires. Any changes made in any one of the Web services will affect the other Web services associated with that current service. For example, if the itinerary involves a flight reservation, car rental, and hotel reservation, the user's database will have links among the three entities. If the flight is cancelled, then automatic cancellation of car and hotel reservations will be triggered, thus easing the burden on the user to manually cancel all associated reservations.

The travel application that we have prototyped is very lightweight and could be developed with great ease and speed. Harnessing the advantages of SyDLink infrastructure, the developer is left with developing just the menu and GUI. The rest is taken care of by SyDLink.

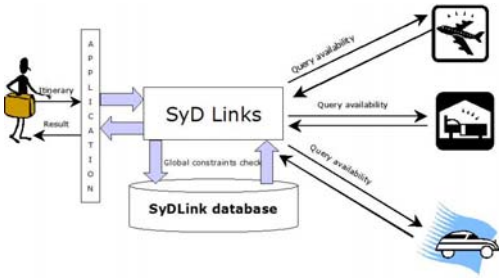

Figure 2. Overview of web service coordination architecture (Travel reservation scenario)

Initial Set Up: SyDLink parses the WSDL URL for all the Web services that are to be integrated. This is internally done by executing the method *parseWsdl* (See Appendix for more detail). The list of services and their methods are available for the application developer after the parsing. The application developer can then choose the services to be integrated, specify the methods that will have constraints, set automatic triggers, etc.

Reservation Scenario Using Global Constraints: Consider a typical travel reservation scenario. A traveler wishes to book flight tickets and make hotel and car reservations. These services are existing Web services. He specifies the necessary details for his trip, and also specifies budget constraints, if any. The application gathers these details and invokes the method *packAndSendWithConstraints* passing the list of methods, their associated parameters, and budget constraints (if any) to SyDLink module. SyDLink, upon receiving this request, invokes the methods on each Web service, aggregates the results, and returns those results that conform to the constraints back to the application. Thus the issues of invoking the services, collecting the results and filtering them are completely taken care of by SyDLink. This scenario is depicted in Figures 2 and 3.

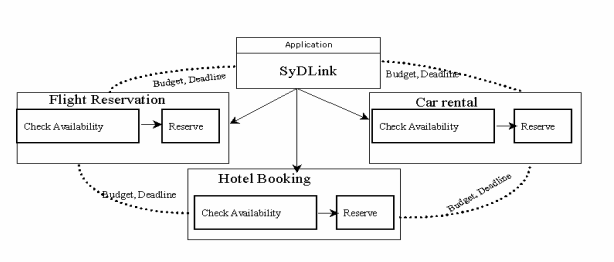

Figure 3. Global constraints within the travel application

Automatic Cancellation Scenario: Links are formed once a traveler has decided on his itinerary. If he chooses to delete any one part of his itinerary, say his flight, then the rest of the trip reservation is also automatically cancelled. This is achieved by SyDLink module. It checks for any associated services pertaining to that link and automatically executes the cancellation requests to other Web services. Figure 4 shows the cancellation ripple effect. Upon canceling the flight schedule, SyDLink automatically triggers cancellations on the corresponding car and hotel reservations.

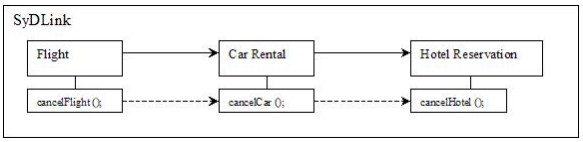

Figure 4. Automatic triggering of events

Developer's Nook: We provide a simple GUI-based interface for the application developer to initially set up and develop SyDLink-enabled collaborative applications. SyD developer's nook provides a separate working area for the application developer. The goal of SyD is to make things as automated as possible. The developer needs to initialize certain entities based on the business logic. He is given access to different Java servlets to perform such functions as setting up of link tables, populating of those tables, specifying constraints, methods that qualify for auto-trigger, etc.

Once the initial set up of the link database is complete, he is in position to develop an ad hoc application. The developer can give the WSDL URL. SyD's development environment parses this URL and identifies the methods available with each Web service. The developer can then choose methods across different Web services that are to be linked in order to maintain global constraints such as overall budget, etc. Once the choice is made, a global logic is maintained across these Web services. Any change in one of the associated methods causes an automatic triggering of the associated methods. The important methods of the SyDLink API are discussed in the Appendix.

4. Implementation of SyDLink Module

The SyDLink module is a significant component of the SyD framework as forming and managing dynamic groups of objects is one key aspect of SyD technology [3]. SyD coordination links enable applications to create a relationship between entities and enforce interdependencies and constraints. Entities here include objects and databases. When specifically applied for Web service entities, coordination links are also called Web Coordination bonds [1]. Within this section we provide details of the existing SyDLink module and also the extensions we performed in order to develop the current methodology to integrate existing Web services.

There are two types of coordination links: *subscription links* and *negotiation links*. A subscription link is used for automatic flow of information from a source entity to other entities that subscribe to it. This can be employed for synchronization as well as more complex changes. Negotiation links enforce dependencies and constraints among entities and trigger changes based on constraint satisfaction.

A coordination link is specified by its type (subscription / negotiation), subtype (permanent / tentative), references to one or more entities, triggers associated with each reference (event-condition-action, ECA, rules), a priority, a logical group constraint (AND, OR, XOR) or other constraints, and link creation link expiry times. The permanent links act as acquired locks or reservations that are released at the expiry time; the tentative links are essentially queued requests for a semaphore-like prioritized reservation or lock [2, 3].

Automatic triggering of events: List of methods associated with a particular method can be automatically triggered by the SyDLink module. Figure 5 shows an example of how invocation of one Web service ("cancel" method) automatically triggers the "cancel" methods on other services. The steps are as follows:

Step 1: A method, say, *cancel*, is invoked on service #1 through SyDLink.
Step 2: SyDLink checks to see if there are any associated methods to be invoked upon this invocation of *cancel* method on service #1.

Step 3: Then *cancel* method on service #1 along with its associated methods (*cancel* on service #2, *cancel* on service #3) are invoked.
Step 4: An optional confirmation of the method is returned back.
Step 5: Final result is then sent back to the user.

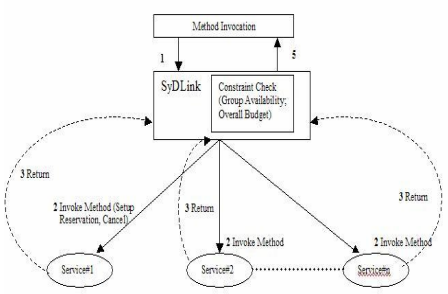

Figure 5. Automatic triggering of methods

SyDLink, thus, automatically triggers requests to Web services that are interlinked. It is therefore in-charge of wrapping the method invocation into XML/SOAP requests receiving the responses and returning only the appropriate result back to the user.

Link Constraint Check: This is one of the important highlights of SyDLink as it facilitates maintaining global constraints across autonomous Web services. Constraints such as budgets or deadlines are maintained across heterogeneous legacy entities that are not self-coordinating and not related to each other in other ways. In such cases, a group of methods are initially selected which are tied together by some constraint. Upon invocation of these methods, the results are not immediately returned, but they are aggregated and checked. Those results that satisfy the constraint are communicated back to the user. Thus, SyDLink enforces global relationships among unrelated entities.

5. Related Work

There has been some research in transaction support for Web services such as the W3C tentative hold protocol [4], and the OASIS Business Protocol (BTP) [5]. These protocols define models for coordinating the transaction execution of Web services based on a predefined set of transaction messages. While these works provide support for distributed transactions on the Web environment through enforcing a predefined set of transaction messages, our work provides a means of coordinating Web services without requiring of all Web services to support a unique and standard transaction protocol. This way, SyDLinks provides more flexibility while preserving the autonomy of Web services.

The present work in the area of e-service composition (WSFL [6], XLANG [7], WSCL [8], WSCI [15], WS-Coordination [16], WS-Conversation [17], BPML [18], XLANG [19], and BPEL4WS [20]) define primitives for composing services and automating service coordination. Most of these consider XML-based standards for Web service technology. However, the primitives for composition proposed by these works do not directly address the problems associated with the necessary homogenization of Web services. And many do not consider the actual coordination and interaction of Web services with dissimilar transaction support. A key differentiation of our effort from others is the focus on high-level specification of coordination primitives as opposed to low level language details, and an enabling development and run-time middleware environment, all geared to ease program development and deployment.

6. Conclusion

Future Internet applications will be collaborative applications among heterogeneous, self-governing, entities. A lot of research is being conducted to leverage off existing transactional concepts and adjusting them to work within a Web services framework. In this paper, we have introduced the architecture and implementation of SyDLinks module, a component of SyD middleware platform, to enable collaborative applications over Web services. This module enables collaborative applications to create and enforce interdependencies and constraints across a group of Web entities using high-level logic. The proposed SyDLink Application Programming Interface enables rapid development and deployment of collaborative applications. Currently, we employ simple time outs for sessions and method invocations and post appropriate error messages to tackle the basic issues of fault tolerance. Possible future work involves allowing distributed coordination among legacy Web services (as opposed to the current centralized coordinator applications) possibly by enhancing the Web service infrastructure [1].

References

[1] S. K. Prasad and J. Balasooriya, Web Coordination Bonds: A Simple Enhancement to Web Services Infrastructure for Effective Collaboration, To appear in the Proceedings of the 37th Hawaii International Conference on System Sciences, January 5-8, 2004, Big Island, Hawaii

[2] S. K. Prasad, A. G. Bourgeois, E. Dogdu, R. Sunderraman, Y. Pan, S. Navathe, and V. Madisetti, "Enforcing Interdependencies and Executing Transactions Atomically Over Autonomous Mobile Data Stores Using SyD Link Technology," Proc. International Workshop on Mobile and Wireless Networks, ICDCS, (2003), pp. 803–809.

[3] S. K. Prasad, V. Madisetti, et al. 2003. A Middleware for Collaborative Applications over a System of Mobile Devices (SyD): An Implementation Case Study. Technical Report CS-TR-03-01, Dept. of Computer Science, Georgia State University.

[4] W3C (World Wide Web Consortium) Note, "Tentative Hold Protocol Part 1: White Paper". [http://www.w3.org/TR/tenthold-1/], November 2001.

[5] Potts, M., Cox, B., Pope, B., "Business Transaction Protocol Primer". OASIS Committee [https://www.oasis-open.org/committees/businesstransactions/documents/primer/Primerhtml/BTP%20Primer%20D1%2020020602.html]

[6] Leymann, F., "Web Services Flow Language (WSFL 1.0)". [http://www-4.ibm.com/software/solutions/Webservices/pdf/WSFL.pdf], May 2001.

[7] Thatte, S., "XLANG: Web Services for Business Process Design". [http://www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm], Microsoft Corporation, 2001

[8] W3C (World Wide Web Consortium) Note, "Web Services Conversation Language (WSCL) 1.0". [http://www.w3.org/ TR/2002/NOTE-wscl10-20020314/], March 2001.

[9] S. K. Prasad, A. G. Bourgeois, E. Dogdu, R. Sunderraman, Y. Pan, S. Navathe, and V. Madisetti, "Implementation of a Calendar Application Based on SyD Coordination Links," Proc. 3rd International Workshop on Internet Computing and E-Commerce, IPDPS, (2003).

[10] S. K. Prasad, et al. 2003. System on Mobile Devices (SyD): Kernel Design and Implementation, MobiSys '03: First International Conference on Mobile Systems, Applications, and Services, Poster and Demo Presentation, May 5-8, 2003, San Francisco.

[11] Transaction over services [http://www.Webservices.org/index.php/article/articleview/381/]

[12] V. Madisetti, "SyD: A middleware infrastructure for mobile iAppliance devices," *EE Times Network*, [http://www. iapplianceWeb. com/story/OEG20021105S0031], November 5, 2002.

[13] S. K. Prasad, M. Weeks, et al. 2003. Toward an Easy Programming Environment for Implementing Mobile Applications: A Fleet Application Case Study using SyD Middleware, *IEEE Intl Wksp on Web Based Syst. and Applns (WEBSA)*, at 27th Ann Intl Comp. Softw and Applns Conf (COMPSAC 2003), Dallas, Nov 3-6.

[14] S. K. Prasad, Erdogan Dogdu, Raj Sunderraman, *Bing Liu* and Vijay Madisetti. Design and Implementation of a listener module for handheld mobile devices. Proceedings of *41st Annual ACM Southeast Conf.*, Savannah, Georgia, 2003 March 7-9

[15] W3C "Web Service Choreography Interface (WSCI) 1.0," *http:// www.w3.org/TR/wsci/*, November 2002.

[16] "Web Services Coordination" (WS-Coordination 1.0), 2002, [*http://www.106.ibm.com/developerworks library/ws-coor/*]

[17] "Web Service Conversation Language" (WSCL) , HP Submission to W3C, *http://www.w3c.org*, 2002.

[18] A. Ankolekar , et al., "DAML-S: Web Service Description for the Semantic Web," *Proc. Int'l Semantic Web Conf.*, Springer-Verlag, Berlin/Heidelberg, 2002, pp. 348-363.

[19] S. Thatte, "XLANG: *Web* Services for Business Process Design," Microsoft, Redmond, Wash., 2001, [*http://www.gotdotnet. com/team/xml_wsspecs/xlang-c/default.htm*].

[20] P. Wohed, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede, " Pattern based analysis of BPEL4WS", Technical Report FIT-TR-2002-04, QUT, *Queensland University of Technology, 2002.*

Appendix: Important Methods of SyDLink API

- boolean c**reateSyDLinkDatabase** (String sourceusername): This method is invoked to create all the necessary tables of SyDLink. This call is done initially when the application developer needs to make an application SyDLink- enabled.

- boolean **parseWsdl**(String wsdladdr): This method parses the WSDL file of the Web service, lists out the methods that can be invoked, parameters to be used etc. The application developer initially, gives the URL location of the Web services that he desires to integrate. This method is then invoked to parse the WSDL. A DOM parser is used in This method is also used to invoke methods of Web services. This creates the SOAP envelope by packing the necessary parameters and sends the request to the Web services. However, methods that are associated with constraints like budget are executed through this method of SyDLink. The resulting response is aggregated and the only results that satisfy the constraints are returned.

- Vector **viewLinks**(String dbusername, String dbpassword, String dburl, String starttime): This method is used to view all the links associated with a particular user. This is a simple yet useful method of SyDLink. In case of a travel reservation application, upon invocation, it would result in the itineraries being displayed.

- boolean **deleteLinks**(String linkid): This method is invoked upon any link deletion. A check is done to see if there are any associated links to be deleted and the links are physically removed from the database.

- String **checkConstraintMethod (**String servicename, String methodname): This is a private method used to see for associated methods. This is used by the *packandsend* method of SyDLink.

this method to parse the XML document. Methods names (and their parameter types) of the given Web service are then extracted and placed in a table for further reference.

- String **packAndSend**(String servicename, String method, Enumeration enum): This method is used to invoke methods of Web services. This creates the SOAP envelope by packing the necessary parameters and sends the request to the Web services. When a SOAP response it obtained, it then returns the desired output back to the user.
- String p**ackAndSendWithConstraints** (String[] servicenames, String[] methods, Enumeration[] enums, String constraintval):