Ubiquitous Web Services for E-Government Social Services

Athman Bouguettaya, Denis Gracanin, Qi Yu, Xiaoyu Zhang, Xumin Liu, Zaki Malik
Department of Computer Science
Virginia Tech
{athman,gracanin,qyu,zhangxy,xuminl,zaki}@vt.edu

Introduction

The *Semantic Web* is defined as an extension of the existing Web, in which information is given a well-defined *meaning* (Berners-Lee, Hendler, & Lassila 2001). The ultimate goal of the envisioned Semantic Web is to transform the Web into a medium through which data and applications can be *automatically understood* and *processed*. The development of technologies for supporting the Semantic Web has been the priority of various research communities (e.g., database, artificial intelligence). A major component in enabling the Semantic Web is the concept of *Web services* (Alonso *et al.* 2003; McIlraith, Son, & Zeng 2001; Medjahed *et al.* 2003). A *Web service* is a set of related functionalities that can be programmatically accessed through the Web. Examples of Web services span several application domains including *e-government* (e.g., welfare social service) and *B2B E-commerce* (e.g., stock trading). Web services are gradually taking root because of the convergence of business and government efforts to making the Web the place of choice for all types of activities. The maturity of XML-based Web service technologies such as SOAP, UDDI, and WSDL is a prominent factor contributing to the large adoption of Web services in the near future (Curbera *et al.* 2002).

Web services have spurred an intense activity in industry and academia to address challenging research issues such as the *automatic composition*, *optimized querying*, and *customized presentation* of Web services. Service composition includes the automatic selection and interoperation of individual Web services to provide value-added and personalized composite services. Optimized service query offers complex and optimized query facilities for combining accesses to diverse service providers. The customized service presentation presents interfaces based on the service input/output data and service semantics. The diversity of these issues calls for the design and development of a comprehensive *Web Service Management System* (WSMS), where Web services would be treated as first-class objects that can be manipulated as if they were pieces of data. A WSMS includes the architectural components that introduce more convenience, flexibility, and effectiveness in managing Web services. In this paper, we present a comprehensive WSMS for e-government called *WebSenior*. Web Services are used *ubiquitously* across all the layers of the system. WebSenior helps senior citizens and their helpers use the Web to request and receive e-government services.

WebSenior: A WSMS for e-Government

WebSenior is a WSMS for digital government. It provides a framework for efficiently accessing e-government services. Its main contributions revolve around three features, including *composing* e-government services, *optimized querying* of e-government services, and *customized* user interface based on Web services. In this section, we provide an overview of these features.

Composing E-government Services

We organize Web services into *communities*. Communities provide a means for an ontological organization of the available service space based on *categories*. All services that have similar categories belong to the same community. Providers identify the community of interest and register their services with it. Services can leave and reenter a community at any time during their life-span. During the registration process, providers must define the mappings between generic operations defined in their community and those defined in their service. A service may offer all or some of the operations defined within a community. For each generic operation, it may use all operation's parameters, a subset of those parameters, and/or add new parameters.

We propose a novel approach for the automatic composition of Web services. This approach consists of four conceptually separate phases: *specification*, *matchmaking*, *selection*, and *generation*. The *specification* phase enables high level descriptions of the desired compositions. The *matchmaking* phase automatically generates *composition plans* that conform to the composers' specifications. The matchmaking algorithm uses as input, the composer's specification and a repository (e.g., UDDI) of pre-existing service interfaces described in WSDL language (extended with semantic constructs). Composers select a generated plan (*selection phase*) based on *Quality of Composition* (*QoC*) parameters (e.g., ranking, cost). Using the selected plan, a detailed description of the composite service is automatically generated

(*generation phase*). This description includes the list of outsourced services, mappings between the composite and oustourced services operations and messages, and the control flow of outsourced operations. The control flow refers to the execution order of the operations outsourced by the composite service.

We describe a *composability model* to check whether Web services can be combined together in the match making algorithm (Medjahed & Bouguettaya 2005). Composability is checked through a set of *rules* organized into four *levels*: *syntactic*, *static semantic*, *dynamic semantic*, and *qualitative*. These four levels check composability of service messages and operations; each rule in a given level compares a specific pair of *attributes* of interacting Web services. Each composability rule specifies the constraints and requirements for checking horizontal, vertical, and hybrid composability. A *Horizontal* composition models a "supply chain"-like combination of operations. A *Vertical Composition* models the "subcontracting" of an operation by another operation. A composite service may also include operations that are horizontally composed and others that are vertically composed. We refer to this type of composition as *hybrid composition*.

Web Service Optimization

We adopt the declarative paradigm of querying services based on the popular *relation* concept in Relational Database Management System (Ouzzani & Bouguettaya 2004). Users submit conjunctive queries over relations in the following form:

$$Q(X) \ :- \ \bigwedge_i R_i(X_i), \bigwedge_k C_k$$

where R_i are relations from the query level. X and X_i are tuples of variables. C_k's represent conditions on variables appearing in the query. Their form is: $C_k = x \ op \ c$, where x is an input or output variable appearing in any X_i, c is a constant, and $op \in \{=, \neq, <, >, \geq, \leq\}$.

A query is resolved via a three level query model to one or multiple service execution plans. The multiple service execution plans are equivalent in terms of their final output, but they may differ according to different parameters, such as response time, resource use, reliability and so on. These differences can be several orders of magnitude large and thus require a service-centric query optimization strategy.

Generation of Service Execution Plans A Service execution plan basically defines a sequence of sets of operations. It can be generated by a three-level query resolution model. The three-level query model consists of:

- *Query Level* – Consists of a set of relations that allow users to formulate and submit declarative queries directed to Web services. Different sets of relations may be defined over the virtual operations using different mapping rules.
- *Virtual Level* – Consists of Web service-like operations typically offered in a particular community. For any relation $R_i \in \mathcal{R}$,

$$R_i(x_1, x_2, ..., x_n) \ :- \ \bigwedge_j Vop_j(y_{j_1}, ..., y_{j_m})$$

  where x_i are the attributes of R_i, $Vop_j \in \mathcal{VOP}$ where $\mathcal{VOP}$ is the set of virtual operations, and y_{j_i} are input and output variables of the corresponding operation. This definition means that to get R_i's tuples, we need to invoke the different virtual operations Vop_j.
- *Concrete Level* – Represents the space of Web services offered on the Web. These are the potential candidates to answer queries. Concrete Web services are *a priori* unknown. They need to be discovered and matched to the virtual operations appearing in a query.

A user query is first mapped into a virtual set of operations. If this set of operations can be serviced by a single Web service, the query processor will consult the service locator to find the "best" concrete Web service. The "best" concrete Web service in this case is computed based on a predefined objective function. An equivalent scalar value is generated using weighted vector values. The weights define the relative importance of the key criteria. If the set of virtual operations can be serviced by a combination of two or more services, the request is forwarded to the service composer. The service composer generates the set of all potentially composable services based on the composability model. These composite services and their service execution plans will be returned to the query processor which will in turn forward those plans to the Quality of Web Service global plan optimizer to select the best service execution plan.

Web Service Presentation

Semantic Web facilitates the automatic data processing between applications. However, the data processed in the Semantic Web is usually not presented in a user friendly way. When users need to interact with services on the Semantic Web, how to automatically provide friendly user interfaces is a challenge. We propose to use a set of user interface Web Services to automatically create user interfaces based on user preferences. The automatically generated user interfaces contain the related service functionality.

In order to facilitate the dynamic generation of the user interfaces, we compose a user interface from different components. These components can host the service description or provide the Web navigation functionality. All of them are hosted and organized by a user interface container application. Generation of the user interfaces is the result of the interactions between the container and the following groups of user interface Web services:

Service User Interface Web Service: This type of service provides a user interface component for a specified Web service so that users can access and operate on that service. After accepting the user interactions dispatched from the container, it invokes the corresponding services. When the invocation is completed, it formats the result data into a well organized user interface component and then transfer it to the container.

User Interface Component Web Service: This type of service acts as a user interface repository. It serves user interface components for container applications. One example

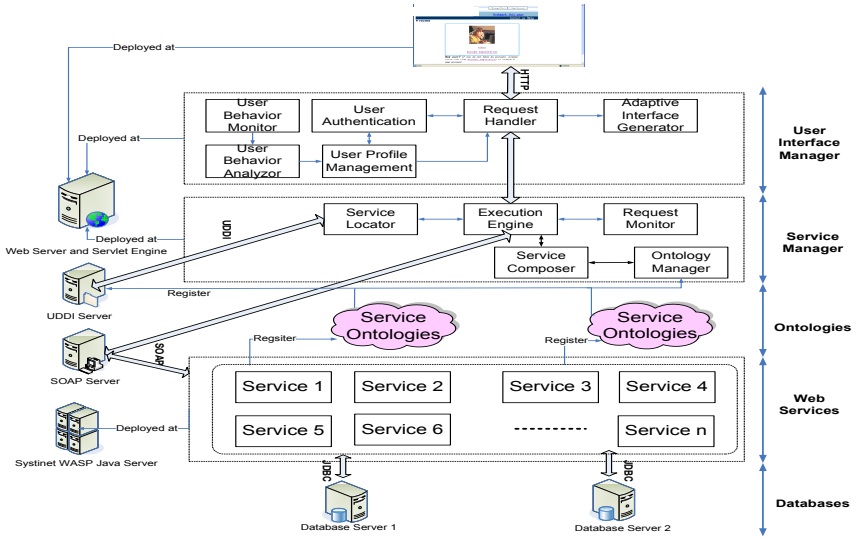

Figure 1: Proposed Web Service Architecture

of such a Web Service can provide a Web calendar script as an HTML script component. The calendar can be embedded into a Web user interface container.

User Interface Adaptation Web Service : This type of service provides instructions or templates for creating accessible user interfaces. The templates are based on the information of the user and the available components of the user interface that are to be generated. User information is gathered from the user interface container application based on a user modeling method. The returned result from the service is based on the user's model and the content of the user interface. For example, an accessibility user interface adaption Web Service will generate a template that provides contrast background and text for senior citizens with visual impairment.

Implementation

The system's architecture (Figure 1) is organized into five tiers: databases, services, service ontologies, service management, and user interface manager. The database tier consists of the databases that store the data used by the different applications. The service tier contains basic Web services. The service ontologies tier organizes services based on their domain of interests. The service management tier has four major components: (i) the service locator, (ii) the service composer, (iii) the ontology manager, and (iv) the execution engine. When a request for a service is received, the service locator interacts with the UDDI registry and discovers relevant services. If the request cannot be satisfied by a single service, the service composition module is invoked. The service composer interacts with the ontology manager to generate an optimized composite service that satisfies the user's request. The execution engine performs the final service execution. The request monitor traces the unsatisfied requests for managerial purposes. The user interface manager is to authenticate users, record and analyze their behavior, and provide the interface to interact with services. Once the user is identified, the system's adaptive user interface (UAI), will dynamically adapts to their needs, abilities, and computer skills. There are six components in this tier: adaptive user interface generator, user authentication, user behavior monitor, user behavior analyzer, request handler, and user profile management.

Conclusion

This paper describes a comprehensive WSMS, called, WebSenior. Web services are used across all system levels to efficiently deliver e-government social services. In this respect, the key feature of the system is the ubiquity of Web services: (1) they are used to model the functionality of the applications (i.e., social services for senior citizens), and (2) as a key technology for building the system itself. WebSenior enables the *automatic composition*, *optimized querying*, and *customized presentation* of Web services.

References

Alonso, G.; Casati, F.; Kuno, H.; and Machiraju, V. 2003. *Web Services: Concepts, Architecture, and Applications*. Springer Verlag (ISBN: 3540440089).

Berners-Lee, T.; Hendler, J.; and Lassila, O. 2001. The Semantic Web. *Scientific American*.

Curbera, F.; Duftler, M.; Khalaf, R.; and Nagy, W. 2002. Unraveling the Web Services Web. *IEEE Internet Computing* 6(2).

McIlraith, S. A.; Son, T. C.; and Zeng, H. 2001. Semantic Web Services. *IEEE Intelligent Systems* 16(2).

Medjahed, B., and Bouguettaya, A. 2005. A Multilevel Composability Model for Semantic Web Services. *IEEE Transactions on Knowledge and Data Engineering* 17(7).

Medjahed, B.; Benatallah, B.; Bouguettaya, A.; Ngu, A.; and Elmagarmid, A. 2003. Business-to-Business Interactions: Issues and Enabling Technologies. *The VLDB Journal* 12(1):59–85.

Ouzzani, M., and Bouguettaya, A. 2004. Efficient Access to Web Services. *IEEE Internet Computing* 8(2).