Customized Delivery of E-Government Web Services

Brahim Medjahed, *University of Michigan, Dearborn*

Athman Bouguettaya, *Virginia Tech*

WebSenior, a prototype project of Virginia Tech and the Virginia Department for the Aging, uses ontologies to automatically generate Web services customized to senior citizens' needs and government program laws and regulations.

The Web is revolutionizing the way citizens interact with businesses and government agencies. Almost all key functions of modern society are being reshaped to exploit opportunities the Web opens. As part of an effort to improve government-citizen interactions, government agencies are providing a wide spectrum of online services.[1] These e-government services span several application domains, including social programs, healthcare, voting, and tax filing. Their potential benefit for senior citizens is of particular interest for two reasons. First, the population segment over 60 years old is growing dramatically in the US, and seniors' needs for assistance increase with age. It's estimated that 31.2 percent of seniors age 80 to 84 require assistance with everyday activities including access to health-related, social, and welfare programs. The percentage increases to 49.5 percent for those over age 85.[2]

Second, seniors represent one of the fastest-growing segments of Web users. To capitalize on this trend, many federal, state, and local agencies now offer online forms to access e-government services dedicated to seniors' needs. Unfortunately, such efforts have so far had little effect in the lives of seniors in terms of their receiving better care and services. Accessing public services poses bureaucratic challenges that can be daunting in any case, but seniors are challenged in ways that are fundamentally different from other age groups.

To address these challenges, Virginia Tech is collaborating with the Virginia Department for the Aging (VDA) on a project called WebSenior. A middleware infrastructure, WebSenior automatically delivers e-government services customized for seniors. We have demonstrated a first-phase prototype of the system with 25 Web services.

Service scenarios

A scenario illustrates the challenges seniors face in obtaining social services from local government agencies. The VDA subcontracts with Area Agencies on Aging to offer services to communities throughout the state. AAAs work with public and private organizations to help seniors and their families find the services and information they need.

Let's say that Mary, a handicapped and visually impaired retiree, wants to receive social services from her county's AAA. Typically, she would first have to travel to the AAA office for an interview. An agency case worker would assess Mary's needs and try to find the best matches for them among a large number of potential services. Let's say the case worker, John, determines that Mary might qualify for the following services: FasTran (transportation for the elderly and handicapped), Meals on Wheels (meal delivery), and Senior Activity Center (a facility that offers seniors a variety of activities). He advises Mary of the information she needs to submit to these organizations before she can receive their services. He might also help her prepare and transmit the information through various communication media—email, snail mail, fax, and phone. In some cases, Mary might have to personally visit the agency offering the service. In all cases, the process is ultimately manual, and processing delays are more the rule than the exception. Furthermore, under the current system, if Mary decided to move to another county, the case worker at that county's AAA would have to initiate the same manual, error-prone process.

As this scenario indicates, the current social services system suffers from intrinsic inefficiencies. This scenario becomes even more realistic when we consider that seniors are generally expected to identify

1541-1672/05/$20.00 © 2005 IEEE
Published by the IEEE Computer Society

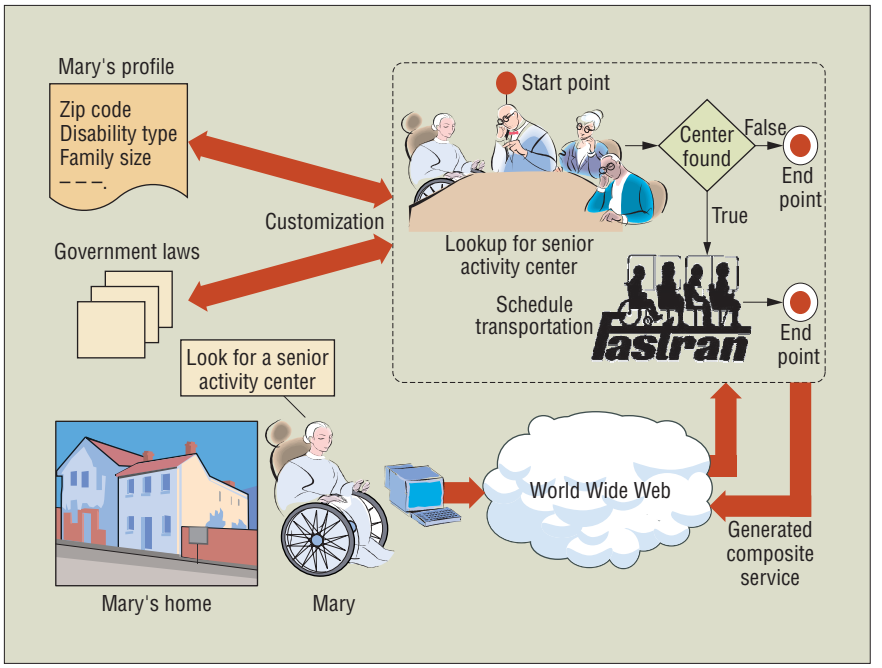


Figure 1. WebSenior process for delivering e-government services.

the necessary government services they want and to be aware of the service eligibility requirements. For example, seniors can request transportation only if they are registered with an AAA, and their eligibility for a given transportation service, such as FasTran, depends specifically on where they live.

Our partnership with the VDA aims to give seniors easy and customized access to services through the Web. For example, let's say that Mary wants to visit a senior activity center for lunch and she needs transportation to get there. Figure 1 shows the online Web-based process that Mary would follow. She would access WebSenior from a computer in her home, and the system would automatically return a composite service that best fit Mary's needs. For that purpose, it uses Mary's personal information stored in a user profile and government rules and regulations defining eligibility conditions for a service. WebSenior would transparently return results for the nearest senior activity center and a convenient transportation service in Web pages appropriate to Mary's visual capacity (for example, in bigger fonts).

This example illustrates three levels of customization:

- *Citizen level*. Seniors are identified by a set of personal properties that define their profile (for example, zip code, and income). The same request issued by several citizens may be answered differently because of their dissimilar profiles. Customization at this level aims at returning Web services that are in accord with individual profiles.
- *Service level*. Executing a Web service operation often requires the execution of related operations. Dependencies between these operations usually model laws and regulations defined by governments regarding the use of their services. Customization at this level refers to automatically generating dependencies that exist between participant services.
- *User interface level*. Seniors are challenged in ways that are fundamentally different from other age groups. For example, because visual and cognitive capacities usually diminish with age, Web service interfaces must consider these constraints. Customization at this level aims at designing user interfaces that automatically adapt to a person's visual and cognitive conditions.

In this article, we address only the citizen and service customizations.

Ontology-based Web service descriptions

The Semantic Web plays a crucial role in automatic delivery of customized e-government services. It extends the existing Web by providing a framework for technologies that give meaning to data and applications for automatic processing.[3] *Ontologies*—formal, explicit specifications of a shared conceptual space[4]—are integral to the Semantic Web in facilitating knowledge sharing and reuse.

Web services constitute a related technology that has recently emerged to deal with the glut of Web applications. Simply put, a Web service is a software component that another computer program can access automatically on the Web.[5] The maturity of XML-based Web service standards such as SOAP, WSDL, and universal description, discovery, and integration (UDDI) is a prominent factor contributing to the fast adoption and deployment of Web services.[5]

A Web service is either simple or composite. A *simple* service is a stand-alone, modular component—it doesn't rely on other Web services to fulfill consumers' requests. An example of a simple service is the popular Meals on Wheels, a program funded by private as well as government contributions, which provides home-delivered meals to homebound elderly. If e-government services are set up for automatic delivery, seniors will be able to access this service directly through a user interface. A *composite* service is a conglomeration of outsourced Web services (called *participants*) working in tandem to offer a value-added service. For example, figure 1 illustrates a composite service for getting lunch from the senior activity center and transportation to and from it.

Semantic description of Web services is the key to automating customized service delivery. We focus here on business process semantics, which describe features related to Web service execution. We use these descriptions to model laws and policies that regulate access to social and welfare services. You can apply semantics to other system descriptions as well—for example, to describe message parameters at the message level or functionality at the operational level—but those levels are outside the scope of this article.

Figure 2 shows the *metadata ontology* we've defined to describe the business process semantics for e-government Web service operations. The metadata ontology provides a conceptual template that service providers, such as government agencies, can use to describe their operations.[4] We modeled the ontology as a directed graph: nodes represent Web service concepts; edges represent relationships between concepts. In addition, nodes are labeled with the cardinality of the

corresponding relationship (cardinality is the number of instances of the relationship). For example, the edge **operation → preoperation** states that an operation has zero or more preoperations. We used the DAML+OIL semantic markup language (www.w3.org/TR/daml+oil-reference) for describing the proposed ontology, but other standards, such as the Web Ontology Language (OWL, www.w3.org/TR/owl-features), could work as well.

In our approach, the service providers are responsible for describing their operations by assigning values to the operations' attributes, such as their purposes and categories. A *vertical ontology* is a key concept supporting this approach. It captures the knowledge valid for a particular domain (government social services for seniors, in our case).[4] A vertical ontology is generally defined by domain experts. For example, the VDA and related agencies, such as the Virginia Department of Health, would create a vertical ontology to define the concepts specific to social benefits for seniors. These include *zip code, income, disability, family size, approved, registered,* and *duration*. Service providers refer to this ontology while defining their operation attributes.

We use XML namespaces to prefix operation attributes with the ontology that defines them.[5] A given attribute could apply to several vertical ontologies, which would require mapping between disparate ontologies, but this issue is outside the scope of this article.

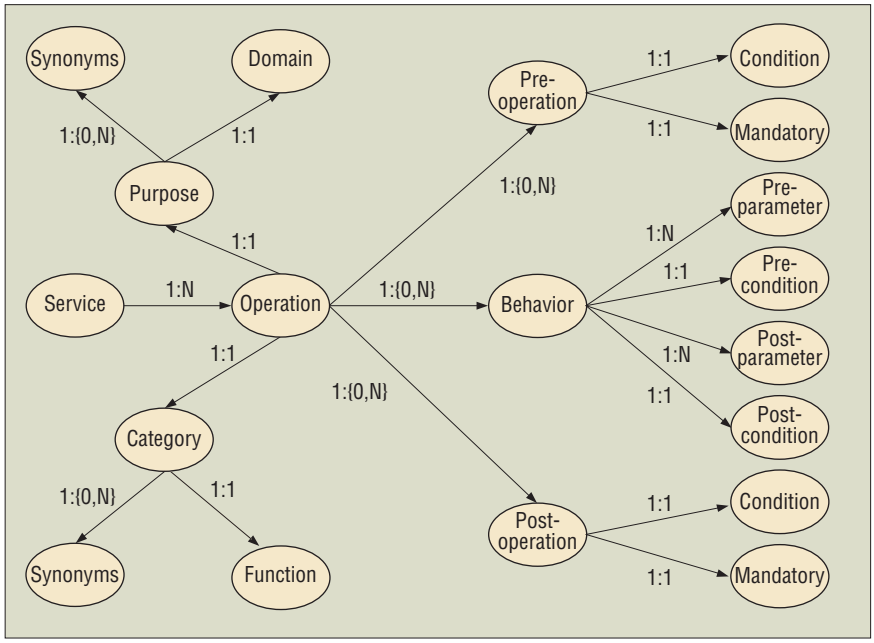


Figure 2. Metadata ontology for e-government Web services. Nodes in the directed graph represent ontology concepts. Edges represent relationships between these concepts and are labeled with the *cardinality* of the corresponding relationship—that is, the number of instances of the relationship.

$$\frac{(\{\text{income},\text{familySize},\text{zipCode}\},(\text{income} < 22{,}090 \wedge \text{familySize} \geq 2 \wedge \text{zipCode} = 22044))}{(\{\text{approved},\text{duration}\},(\text{approved} = \text{true} \wedge \text{duration} = 6))}$$

Figure 3. Pre- and postconditions of a rule associated with registerFoodStamp.

Operations: category, purpose, and business logic

As figure 2 shows, the semantic description of an operation includes two attributes: *category* and *purpose*. The category describes an operation's *domain*, or area of interest (for example, health care), and *synonyms*, or alternative domain names (for example, *medical* is a synonym of *health care*). The purpose describes the operation's *function*—that is, its business goal (for example, eligibility, registration, and mentoring), and synonyms for the function (for example, *subscription* is a synonym of *registration*).

An operation's semantic description must also specify its *business logic* as an attribute. Business logic refers to the outcome behavior expected after an operation executes given a specific condition. This attribute lets us model the eligibility conditions specific to the execution of each social or welfare program's operation. The business logic for operation op_i is defined by a set of rules. Each rule R_i^m has the following format:

$$R_i^m = \frac{(\text{PreParameters}_i^m, \text{PreCondition}_i^m)}{(\text{PostParameters}_i^m, \text{PostCondition}_i^m)}$$

WebSenior defines the parameter sets PreParameters_i^m and $\text{PostParameters}_i^m$ according to a vertical ontology. PreParameters_i^m and $\text{PostParameters}_i^m$ generally refer to an operation op_i's input and output parameters, where m is the rule number of operation op_i. However, in some cases, they can refer to parameters that are neither. For example, assume that every senior's address registered with the Department for the Aging is stored in the department's database. In this case, the address parameter shouldn't be required as input for the **orderMeal** operation because an application could retrieve its value from the database.

PreCondition_i^m and PostCondition_i^m are predicates over the parameters in PreParameters_i^m and $\text{PostParameters}_i^m$, respectively. The rule R_i^m specifies that if PreCondition_i^m holds when op_i starts, then PostCondition_i^m holds after op_i reaches its end state. If PreCondition_i^m doesn't hold, there are no guarantees about the operation's behavior.

Figure 3 is an example of the pre- and postconditions of a rule associated with the operation **registerFoodStamp**. The rule specifies that citizens with a yearly income less than US$22,090 and a minimum household size of two individuals are eligible for food stamps for a six-month period.

Pre- and postoperations

Executing an operation might require going through a predefined process that involves the execution of other operations.

Let op_i and op_j be two operations: op_i is a *preoperation* of op_j if op_j can't be invoked before op_i's execution terminates. The definition of a preoperation relationship (op_i, op_j) includes the specification of a *condition* and a *mandatory* attribute. The condition is a predicate over op_i's input and output para-

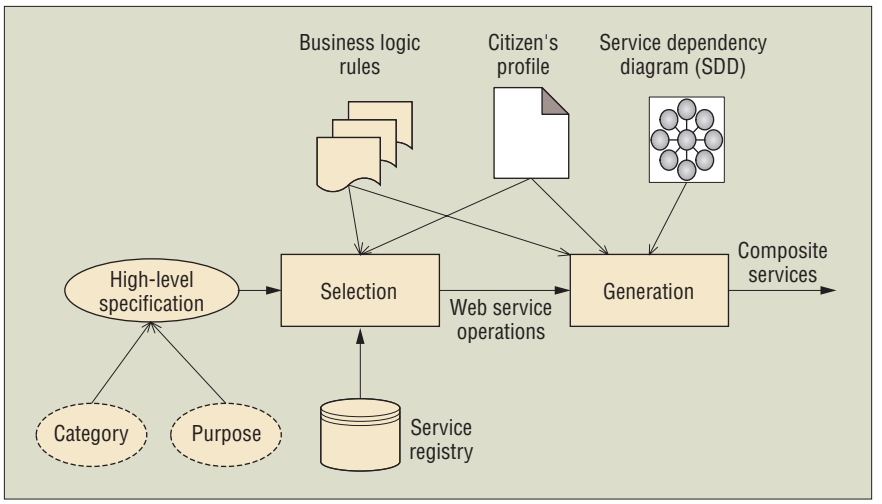


Figure 4. Block diagram of proposed WebSenior approach to automatic customization of e-government Web services.

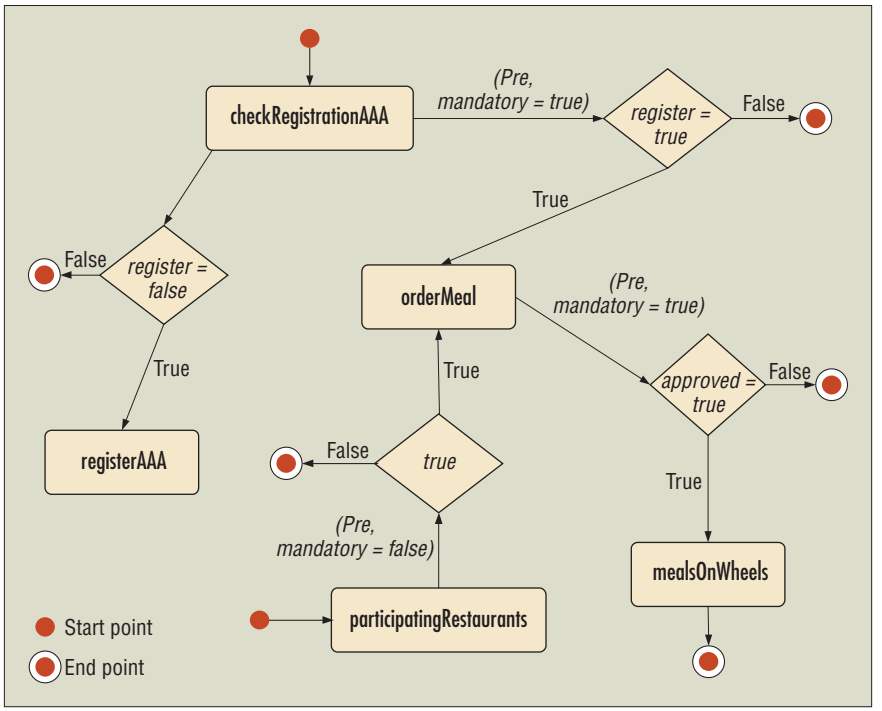


Figure 5. Example of service dependency diagram for ordering a meal.

meters, defined according to a vertical ontology. If no condition is specified for a given preoperation, then the default value is true. For example, seniors must order a meal from a participating restaurant via the **orderMeal** operation before requesting its delivery through the **mealsOnWheels** operation. The mandatory attribute takes Boolean values and specifies whether executing op_i is mandatory or optional. If this attribute is true, then the relationship between op_i and op_j is obligatory. Otherwise, it is recommended. For example, it is obligatory that seniors order meals from a restaurant before requesting home delivery from a volunteer center. However, it is recommended that they obtain the list of participating restaurants before ordering a meal.

We say that op_i is a postoperation of op_j if the termination of op_j precedes the invocation of op_i. For example, a senior who registers for a food stamp program (**registerFoodStamp**) registers, as a consequence, for a nutritional counseling course (**registerNutritionCourse**) as well. If op_i is a preoperation of op_j, then op_j is not necessarily a postoperation of op_i. For example, **checkRegistration** is a preoperation of **orderMeal**. However, **orderMeal** isn't a postoperation of **checkRegistration**. Indeed, seniors don't need to order meals whenever their registration with an AAA is checked.

As for preoperations, we associate a condition and a mandatory property with each postoperation relationship (op_i, op_j). A postoperation may also be mandatory or optional. For example, **registerAAA** is a postoperation of **checkRegistrationAAA**. Additionally, the postoperation relationship (**checkRegistrationAAA**,**registerAAA**) is optional (mandatory = false). Indeed, seniors don't necessarily have to register with an AAA if they're unwilling to do so.

WebSenior customized services

The semantic description of e-government Web services lets us define a novel approach for automatically generating customized services. Figure 4 shows a block diagram of our approach. It consists of two conceptually separate phases, selection and generation, and uses Unified Modeling Language activity diagrams to model pre- and postoperation service dependencies.

Service dependency diagram

Pre- and postoperations provide the means to specify predefined business processes. We adopted UML activity diagrams to model pre- and postoperation relationships. Activity diagrams are the most widely used process-modeling techniques in conventional interaction technologies (for example, workflows) as well as Web services.[5] Their popularity stems from their ease of use and simplicity for modeling business processes. Several tools are available for designing business processes using activity diagrams (for example, Rational Rose). Additionally, UML has become the de facto standard for representing application architecture and design models.

Modeling pre- and postoperations with UML activity diagrams results in a *service dependency diagram*. Figure 5 shows an example WebSenior SDD. SDDs are defined by the government agencies providing the services (in this case, the VDA and Department of Health). They can contain several start and end points, like the one in figure 5.

Each edge in the diagram is labeled with a relationship attribute. This attribute takes either a Pre or Post value to specify whether the edge models a pre- or postoperation. For example, the edge checkRegistration → orderMeal models a preoperation relationship. It specifies that users (in our scenario, this means seniors or case workers) must execute checkRegistration before invoking the orderMeal operation. Edges are also labeled with a mandatory attribute. For example, orderMeal is a mandatory preoperation of mealsOnWheels. However, users can invoke orderMeal even if participantRestaurants hasn't previously been executed.

SDD diagrams can also indicate that one operation conditionally follows or precedes another. For example, the diamond between the checkRegistration and orderMeal operations states that the value returned by the register parameter must be true before invoking orderMeal.

Selection phase

The input to the selection phase is a high-level specification of a senior citizen's need. The specification doesn't refer to an actual service; instead, it reflects the senior's (or AAA case worker's) selection of an operation's category and purpose. Let's assume that at the initial WebSenior screen, a senior chooses "senior activity center" and "visit" for the desired category and purpose attributes, respectively. WebSenior would interpret this request as "visiting a senior activity center." During the selection phase, the Web service composition process would execute the algorithm in figure 6.

Next, WebSenior accesses the service registry to find the operations whose category and purpose are compatible with those defined in line 2 of the specification. We assume that each operation stored in the registry has a category and purpose assigned to it. Each selected operation op_i has a set of business logic rules $\mathcal{R}$ associated with it. The composition engine then checks that each rule in $\mathcal{R}$ is valid (lines 4-8). A rule R_i^m is valid if Precondition_i^m is true given the senior's profile. Since business logic rules model eligibility conditions, the validity test ensures that the senior is eligible for the selected operation's social or welfare program. If a senior isn't eligible, there's no need to determine the operations dependent on op_1 because op_1 won't be returned to that person. Note that these dependencies can still be determined (by accessing the SDD diagram) if another senior who is eligible for op_1 invokes it.

The selection process might return several operations. Quality-of-Web-service parameters offer a promising way to select the "best" operation.[6] Example QoWS parameters include cost, availability, and reputation. The aim is to return operations with the best values for meeting those parameters. WebSenior computes the overall quality of the operation on the basis of its QoWS parameters and adjusts the returned results using user-supplied weights for each parameter. For example, one senior might be interested in low-cost or free operations and services. Another might prefer operations and services that rank high in terms of preserving citizens' privacy.

```
Algorithm Selection (input profile) {
(01) specification = get(category, purpose);
(02) 𝒪 = discover(registry, category, purpose);
(03) selected = Ø;
(04) for each op_i ∈ 𝒪 do {
(05)   isvalid = true;
(06)   for each rule ∈ op_i.R
(07)     if ¬ check_validity(rule.precondition, profile) then isvalid = false;
(08)   if isvalid then selected = selected ∪ op_i; }
(09) return(selected); }
```

Figure 6. Selection phase Web service composition algorithm.

```
Algorithm Generation (input selected) {
(01) CS = Ø;
(02) for each op ∈ selected do
(03)   if ¬ found (op_i, SDD) then CS = CS ∪ {op_i};
(04)     else
(05)       { subdiagram = compute_paths(op_i, SDD);
(06)       iscomposite = true;
(07)       for each op_j ∈ subdiagram do {
(08)         isvalid = true;
(09)         for each rule ∈ op_j.P
(10)         if ¬ valid(rule.precondition, profile)
(11)         then { isvalid = false;
(12)           break; }
(13)         if ¬ isvalid then
(14)         if mandatory (op_j, subdiagram)
(15)           then { iscomposite = false;
(16)             break; }
(17)           else remove (op_j, subdiagram); }
(18)       if iscomposite then CS = CS ∪ subdiagram; }
(20)   return(CS); }
```

Figure 7. Generation phase Web service composition algorithm.

Generation phase

The generation phase aims at returning composite Web services that fit with the senior's specification. The idea is to use the operations returned by the selection phase and augment them with pre- and postoperation relationships. WebSenior uses each such operation to generate a potential composite service. During the generation phase, the Web service composition process would execute the algorithm in figure 7.

Let op_i be an operation returned by the selection phase. WebSenior's first step is to search for op_i in the SDD diagram. For that purpose, we use the well-known breadth-first algorithm. If op_i isn't found, then the generation phase returns op_i as an answer for the senior's request in line 3. Otherwise, we

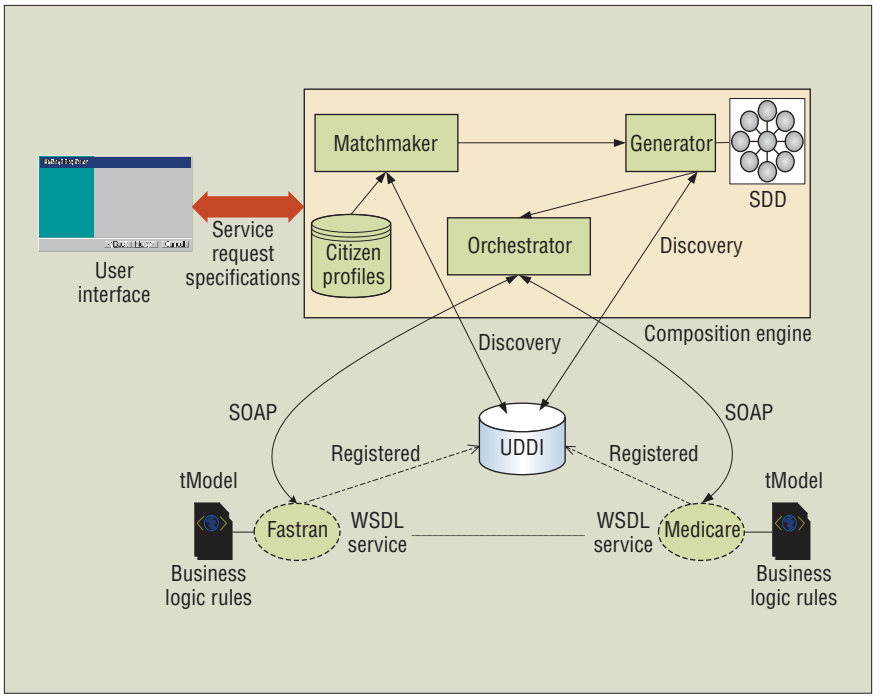


Figure 8. Composition engine architecture for WebSenior.

determine all operations that are related to op_i via pre- and postoperation relationships (lines 5-17). For that purpose, we determine the set $\mathcal{P}$ of all paths in the SDD diagram that contain op_i as a vertex. We use the Floyd-Warshall dynamic programming algorithm for computing these paths. The composite service potentially returned by the generator is defined by the SDD subdiagram that contains all paths in $\mathcal{P}$.

Given a generated subdiagram, the generation algorithm then checks the validity of the business logic rules of all operations in that subdiagram (except for op_i, which has been checked in the selection phase—lines 7–12).

On occasion, a rule associated with an operation op_j of the composite service isn't valid. In this case, we look at the mandatory attribute of the pre- or postoperation relationship in which op_j is involved (lines 14–17). If this attribute is false, then we remove op_j from the composite service because op_j is not required. Otherwise, WebSenior discards the SDD. The generator then executes the same algorithm for the next operation returned by the selection process.

Prototype implementation

We have implemented a WebSenior prototype using SOAP, UDDI, and WSDL standards for invocation, discovery, and description of Web services.[5] Figure 8 depicts the composition engine's architecture. It has three major modules: the *matchmaker*, *generator*, and *orchestrator*.

The matchmaker receives seniors' service request specifications as input and implements the selection algorithm to determine relevant service operations. For that purpose, matchmaker provides access to a UDDI registry that stores WSDL descriptions of e-government services. We implemented the UDDI repository with Systinet's Web Applications and Services Platform (WASP) DDI Standard 3.1. We used a Cloudscape 4.0 database to create the registry. The matchmaker implements the UDDI Inquiry Client using a WASP UDDI API and provides access to a seniors' profiles repository stored in an mSQL database.

Each Web service has a set of business logic rules associated with it. We designed specific tModels (technical specifications defined for Web services) for business logic rules. The matchmaker forwards the list of selected operations along with the senior's profile to the generator, which implements the generation algorithm. The generator uses the SDD, implemented as a directed and labeled graph, to generate composite services; it accesses the UDDI registry to retrieve the business rules of SDD's operations.

Generated composite services are forwarded to the orchestrator for execution. The orchestrator invokes all operations included in composite services in the SDD-specified order. SOAP envelopes encapsulate all communications (invocations and replies). We use Apache SOAP to provide the tools necessary to deploy SOAP messaging.

The "Related Work in Web Services Delivery" sidebar presents information about other approaches.

The work presented here is ongoing. We currently have an in-house implementation of WebSenior in Virginia Tech's E-Commerce and E-Government Research Lab. In cooperation with our partners from VDA, we're extending the implementation to include an additional 25 Web services. The first version of the prototype includes the composition engine's matchmaker and generator, which we've demonstrated to our partners. We're currently working on the second version, which will include the orchestrator and techniques for customized user interfaces. Our future plans include testing the prototype with select groups of seniors and using their feedback to refine the prototype. We are also planning to deploy WebSenior in our partners' systems.

Although our focus in this article was on customization at the citizen and service levels of WebSenior, efforts are under way at the interface level to automatically generate user interfaces that will adapt to seniors' physical challenges, such as offering a choice of type size. We are also investigating techniques that are based on monitoring seniors' changing physical and cognitive abilities so that we can provide a more adaptive user interface. ■

Acknowledgments

Athman Bouguettaya's research was supported by the National Institutes of Health's National Library of Medicine grant 1-R03-LM008140-01.

References

1. A. Bouguettaya et al., "Managing Government Databases," *Computer*, vol. 34, no. 2, 2001, pp. 56–64.

2. W.C. Mann, "The Aging Population and its Needs," *IEEE Pervasive Computing*, vol. 3, no. 2, 2004, pp. 12–14.

Related Work in Web Services Delivery

Several techniques and projects relate to our approach.

Semantic Web services

DAML-S (DARPA Agent Markup Language-Services) is a major effort aimed at enabling the semantic description of Web services (www.daml.org/services). However, DAML-S gives little support for the business process semantics of Web services. For example, it doesn't allow the specification of pre- and postoperations for automatically generating composite services. Nor does it explicitly define notions of behavior and business logic.

The Web Service Modeling Ontology describes various aspects related to Semantic Web services (www.wsmo.org). WSMO uses the Web Service Modeling Framework as a starting point, and refines it by developing a formal ontology and language. Like DAML-S, WSMO provides little or no support for specifying interoperational relationships.

Standardization efforts for Web services

Several efforts are under way to define standards for Web services.[1] BPEL4WS (Business Process Execution Language for Web Services) is a language for specifying composite services. A BPEL4WS composite service (also called a *process*) consists of several steps called *activities*. BPEL4WS defines a collection of primitive activities, such as *invoke*, as well as structures for combining them, such as *sequence* and *while*.

Our approach complements BPEL4WS and other standardization efforts, such as WS-Coordination and WS-Transactions.[1] We focus on generating customized composite services, while BPEL4WS focuses on defining language constructs to specify composite services. Our approach doesn't preclude developers from adopting BPEL4WS to describe composite services.

The W3C's Web Service Architecture is another emerging standardization effort for Web services (www.w3.org/TR/ws-arch). WSA provides a conceptual model for understanding services and the component relationships for managing them. The architecture doesn't attempt to specify how Web services are implemented, nor does it detail how to combine services or specify their semantics. WSA merely identifies the global elements in a Web service environment that ensure interoperability.

Automatic service composition

A recent trend in automatic service composition is to use AI planning techniques. The composition engine treats service composition as a planning problem. Ideally, given a user's objective and a set of Web services, a planner would find a collection of service requests that achieves the objective. For example, SHOP2 (Simple Hierarchical Ordered Planner 2) adopts the Hierarchical Task Network concept as a planning methodology.[2] The planning system decomposes tasks into smaller and smaller subtasks, until it finds primitive tasks that can be performed directly.

AI planning techniques differ from our approach in two main aspects. First, AI techniques are knowledge based, while our approach is policy and rule driven. Second, AI techniques provide little support for Web service standards, while our approach is built on standards such as SOAP, UDDI, and WSDL.

Ninja (a framework for network services) introduces a technique called *automatic path creation* to support service composition.[3] When APC receives requests for composite service execution, it creates a path that includes a sequence of operators. The operators perform computation on data and the connectors that provide data transport between operators. Ninja focuses mostly on fault tolerance by replicating services on multiple workstations. Its operator functional classifications for automating selection are limited. It doesn't consider the business logic and relationship between operations.

SWORD (a developer toolkit for Web service composition) is a technique that uses a rule-based expert system to automatically determine whether a desired composite service can be constructed using existing services.[4] SWORD doesn't focus on relationships between Web services or customized composite Web service generation based on user profiles.

Other techniques for composing Web services include WISE, eFlow, and CMI.[5] These techniques mostly provide constructs for specifying composite services. They provide little support for automatic Web service composition.

E-government Web service projects

To our knowledge, WebSenior is one of the first projects to provide support for e-government Web services. The Organization for the Advancement of Structured Information Standards (OASIS) has an e-government technical committee. The TC is a forum for governments to express their requirements with respect to XML-based standards. It provides a mechanism for creating best-practice documents and promotes the adoption of OASIS specifications. The US federal government published a draft document based on a TC recommendation to enhance interoperability by adopting a common data search standard. However, TC efforts to support e-government Web services are still in their infancy.

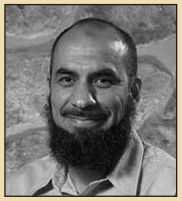
MyNJbusiness, another project that applies Web service technologies for e-government,[6] deals with customized service delivery in the context of the US Small Business Development Center. It specifies governmental regulations as a condition-action rule base to identify component services. The rule base is a collection of statements about regulatory, functional, geographical, and temporal regulations. However, MyNJbusiness doesn't deal with the business logic of operations and pre- and postoperation relationships.

References

1. G. Alonso et al., *Web Services: Concepts, Architecture, and Applications,* Springer, 2003.
2. D. Wu et al., "Automating DAML-S Web Service Composition Using SHOP2," *Proc. Int'l Semantic Web Conf.*, LNCS 2870, Springer, 2003, pp. 195–210.
3. S.D. Gribble et al., "Scalable, Distributed Data Structures for Internet Service Construction," *Proc. 4th Symp. Operating Systems Design and Implementation*, Usenix Assoc., 2000, pp. 319–332.
4. S.R Ponnekanti and A. Fox, "SWORD: A Developer Toolkit for Web Service Composition," *Proc. WWW 2002 Conf.*, Elsevier, 2002, pp. 83–107.
5. B. Medjahed et al., "Business-to-Business Interactions: Issues and Enabling Technologies," *VLDB J.*, vol. 12, no. 1, 2003, pp. 59–85.
6. S.A. Chun, V. Atluri, and N.R. Adam, "Dynamic Composition of Workflows for Customized eGovernment Service Delivery," *Proc. 3rd NSF Conf. Digital Government Research*, US Nat'l Science Foundation, 2002, pp. 383–389.

The Authors

Brahim Medjahed is an assistant professor in the Department of Computer and Information Science at the University of Michigan-Dearborn. His research interests include data integration, the Semantic Web, Internet computing, Web services, and bioinformatics. He received his PhD in computer science from Virginia Polytechnic Institute and State University (Virginia Tech). He is a member of the IEEE and the ACM. Contact him at Dept. of Computer and Information Science, Univ. of Michigan, 4901 Evergreen Rd., Dearborn, MI 48128; brahim@umich.edu.



Athman Bouguettaya is an associate professor in the Department of Computer Science at Virginia Tech, where he is also director of the E-Commerce and E-Government Research Lab. His current research interests are in Web databases and Web services focusing on e-government and e-commerce. He received his PhD in computer science from the University of Colorado at Boulder. He is a senior member of the IEEE and serves on the editorial boards of the *Distributed and Parallel Databases Journal* and the *International Journal of Cooperative Information Systems*. Contact him at the Dept. of Computer Science, Virginia Tech, 660 McBryde Hall, Blacksburg, VA 24061; athman@vt.edu.

3. T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," *Scientific American*, vol. 284, no. 5, 2001, pp. 34–43.
4. D. Fensel, *Ontologies: A Silver Bullet for Knowledge Management and Electronic Commerce*, Springer, 2003.
5. G. Alonso et al., *Web Services: Concepts, Architecture, and Applications,* Springer, 2003.
6. M. Ouzzani and A. Bouguettaya, "Efficient Access to Web Services," *IEEE Internet Computing*, vol. 8, no. 3, 2004, pp. 34–44.

COMING NEXT ISSUE
January/February

AI's Cutting Edge

This issue will examine a variety of topics at the forefront of AI research, including data mining, natural language processing, multiagent systems, machine learning, and smart environments.