Using Web Services in E-Government Applications

Brahim Medjahed  Mourad Ouzzani  Athman Bouguettaya

Department of Computer Science, Virginia Tech
7054 Haycock Road, Falls Church, VA 22043
{brahim,mourad,athman}@vt.edu

(Type of paper: Long  Demo as well: Yes)

Abstract

The Family and Social Services Administration (FSSA) provides a large spectrum of welfare programs to assist needy citizens. However, collecting social benefits using current FSSA systems is an ad hoc and demanding process. Case officers must typically delve into a wealth of applications to access FSSA welfare programs. To facilitate this process and hence expediously satisfy citizens’ needs, we organize FSSA applications into Web services. The use of Web services caters for the efficient discovery and composition of welfare programs. We propose a declarative framework for composing e-government services. We define an algorithm to generate composition plans from a high level specification of the desired service. The algorithm uses a set of rules to check the compatibility of e-government services. We also provide an implementation based on emerging Web service standards including WSDL, UDDI, and SOAP.

1 Introduction

One of the major concerns of digital government is to improve government-citizen interactions using information and communication technologies [9]. This spans several government sectors including social services agencies. In our digital government project [4], we have teamed up with Indiana’s Family and Social Services Administration (FSSA). The FSSA provides a large spectrum of welfare programs to assist low income citizens, strengthen families and children, and help elderly and disabled people. However, collecting social benefits is currently a frustrating and cumbersome task. Citizens must often visit different FSSA offices located within and outside their home town. Additionally, case officers must delve into a wealth of welfare programs to determine those that best meet citizens’ needs. Assume, for example, that a pregnant teen visits FSSA to receive the benefits to which she is entitled. Such benefits include WIC (a federally funded food program for Women, Infants, and Children) and medicaid (a healthcare program for low income citizens and families). The case officer must also look for a Teen Outreach Program (TOP) to offer childbirth and postpartum educational support to the teen. This difficulty in collecting social benefits prevents citizens from becoming self-dependent with a consequent harmful impact on their welfare and health.

Collecting social benefits using current FSSA systems is an ad hoc and demanding process. Case officers must typically delve into a large number of proprietary applications (e.g., Indiana Client Eligibility System) to access FSSA welfare programs. To facilitate this process and hence expediously satisfy citizens’ needs, we organize FSSA applications into Web services. Simply put, a Web service (or e-government service) is a functionality that can be programmatically accessible via the Web [11]. An example of e-government service is medicaid; it provides a set of operations (e.g., searchPCP to look for a primary care provider) to access the medicaid healthcare program. The use of Web services caters for the dynamic discovery and composition of welfare programs. A composite service aggregates multiple e-government services (referred to as components) to provide a value added service [7]. An example of composite service is a pregnancy benefits service which combines WIC, medicaid, and TOP services. To citizens and case officers, pregnancy benefits provides a one-stop social service although it outsources from several social services located in geographically distant FSSA bureaus.

Several techniques have recently been proposed to define composite services [10]. However, these techniques generally require dealing with low level programming details. This makes the process of composing services tedious for non-expert users. For example, composers need to figure out the way operations are interconnected, services are invoked, and exchanged messages are mapped to each other.
In this paper, we propose a declarative framework for composing e-government services. Citizens and case officers specify the what part of the desired composition, but do not concern themselves with the how part. We define an algorithm to generate composition plans from a high level specification of the desired service. By composition plan, we refer to the list of components and how they interact with each other (plugging operations, mapping messages, etc.) to form the composite service. The algorithm uses a set of rules to check the compatibility of component services. We also provide an implementation based on emerging Web service standards. These include WSDL (Web Service Description Language) [3] for describing operational features of Web Services, UDDI (Universal Description, Discovery and Integration) [2] for advertising and discovering Web services, and SOAP (Simple Object Access Protocol) [1] for exchanging XML data across the Web.

The paper is organized as follows. In Section 2, we propose a compatibility model for e-government services. In Section 3, we describe our approach for composing e-government services. In Section 4, we describe our implementation. In Section 5, we provide some concluding remarks.

2 Compatibility Model for E-Government Services

Composing e-government services requires the description of each component service so that other services can understand its capabilities and learn how to interact with it. In our approach, we use WSDL standard [3] for describing e-government services. Major IT companies involved in research and development (IBM, Microsoft, Ariba, etc.) are participating in WSDL development. Hence, it is most likely that WSDL will gain considerable momentum as the language for Web service description.

One of the issues when defining composite services is whether component services are compatible. For example, it would be difficult to invoke an operation if there is no mapping between the parameters requested by the operation (e.g., data types or number of parameters) and those transmitted by the client service. This issue is particularly important in automated service composition. In this section, we define a set of rules to check the compatibility of e-government services.

2.1 Syntactic Compatibility

An e-government service is invoked through WSDL operations. Each operation has an input and/or output message. It also has one of the following signatures: one-way, notification, request-response, and solicit-response [3]. In a one-way (notification) operation, the service receives (sends) a message. In a request-response (solicit-response) operation, the service receives (sends) a message and sends (receives) a correlated message. An example of request-response operation is WIC::checkEligibility. This operation receives information including citizen’s income and family size. It returns a result indicating if the citizen is eligible and a list of additional benefits (e.g., medicaid) to which the citizen is eligible. WIC::eligibilityExpiration is an example of notification operation. It notifies citizens about the termination of their eligibility period (usually 6 months).

For two operations to be “plugged” together, they must have “dual” signatures [5]. A notification operation at one service must be connected to a one-way operation at a partner service. Similarly, a solicit-response operation maps to a request-response operation at a partner service.

**Definition 1** - Two operations $op_{ik}$ and $op_{j\ell}$ are signature compatible if (i) $\text{Signature}(op_{ik})$ = “notification” and $\text{Signature}(op_{j\ell})$ = “one-way”; or (ii) $\text{Signature}(op_{ik})$ = “one-way” and $\text{Signature}(op_{j\ell})$ = “notification”; or (iii) $\text{Signature}(op_{ik})$ = “solicit-response” and $\text{Signature}(op_{j\ell})$ = “request-response”; or (iv) $\text{Signature}(op_{ik})$ = “request-response” and $\text{Signature}(op_{j\ell})$ = “solicit-response”. □

A characteristic of WSDL services is the independence of their messages and operations from the underlying transport protocols. In fact, the same service may support different transport protocols. Hence, it is important to make sure that component services understand each other. We define below a compatibility rule to check whether component services support a common protocol.

**Definition 2** - Let $S_i$ and $S_j$ be two e-government services. $S_i$ and $S_j$ are transport protocol compatible if $\text{Protocols}(S_i) \cap \text{Protocols}(S_j) \neq \emptyset$. □
2.2 Semantic Compatibility

E-government services interact by exchanging messages. A message contains a set of parameters (called parts in WSDL), each parameter is associated with a type. We adopt XML Schema [6] as a canonical type system. XML Schema data types are either user-derived or built-in. User-derived types are those defined by individual schema designer. Built-in types are those pre-defined in the XML Schema specification. They can be either primitive or derived. Contrary to primitive types (e.g., string, decimal), derived types are defined in terms of other types. For example, integer is derived from the decimal primitive data type [6].

One important aspect to consider during message exchange is compatibility between the types of parameters sent by a service and the types of parameters expected by its partner. We define two data type compatibility methods: name and derived compatibility. Two parameters are name compatible if they have the same type. A parameter \( p_i \) is derived compatible with a parameter \( p_j \) if the type of \( p_i \) is derived from the type of \( p_j \).

**Definition 3** - A message \( M_i \) is type compatible with a message \( M_j \) if for each parameter \( p_{ik} \) of \( M_i \), there is a parameter \( p_{jk} \) of \( M_j \) so that \( p_{jk} \) is name or derived compatible with \( p_{ik} \).

For every pair of “plugged” operations, we define a compatibility rule to compare the types of input and output messages. The idea is to check that each input of an operation is type compatible with the output of the other operation. The parameters of each input message map to all or some of the parameters contained in the output message of the other operation. For example, assume that the composer defines a solicit-response operation foodProgramEligibility for pregnancy benefits service. This operation checks citizen’s eligibility for supplemental food program. Assume that the composer wants only to know whether the citizen is eligible for such program. In this case, the input message of foodProgramEligibility would contain one parameter, that is eligibilityStatus. This message is type compatible with the output message of WIC::checkEligibility although the latter contains additional parameters.

**Definition 4** - Two operations \( op_{ik} \) and \( op_{jl} \) are message compatible if (i) Input\((op_{ik})\) is type compatible with Output\((op_{jl})\); and (ii) Input\((op_{jl})\) is type compatible with Output\((op_{ik})\).

To capture semantic features of e-government services, we associate two attributes to each operation: type and category. The type describes the business function offered by the operation. Examples of types include “eligibility”, “counseling”, and “mentoring”. The category describes the domain of interest of the operation. Examples of categories include “health”, “pregnancy”, and “adoption”. Additionally, we define a Category attribute for each e-government service. This is especially important for modeling composite services. Indeed, the domain of interest of a composite service may differ from the domains of interest of its operations. For example, the pregnancy benefits composite service is related to the “pregnancy” category. However, it includes operations related to “insurance”, “nutrition”, and “housing”.

To test the semantic compatibility of interconnected operations, it is important to make sure that their types and categories are similar. For example, it would be semantically “incorrect” to interconnect a “counseling” operation with an “eligibility” operation since these operations offer different types of functionalities.

**Definition 5** - Two operations \( op_{ik} \) and \( op_{jl} \) are operation semantics compatible if (i) Type\((op_{ik})\) = Type\((op_{jl})\); and (ii) Category\((op_{ik})\) = Category\((op_{jl})\).

2.3 Compositional Compatibility

One important aspect to consider in service composition is whether combining services in a specific way provides an added value. For that purpose, we define a compatibility rule to test whether a composite service is sound. Intuitively, a composite service is sound if it provides an added value. For example, combining WIC and medicaid services allows low income pregnant women to collect social benefits for which they are eligible.

To check the soundness of a composition plan, we first introduce the notion of composition template. A Composition template gives the general structure of a composite service. It is modeled by a directed graph \((V, E)\) where \( V \) is a set of service categories. A special vertex corresponds to the composite service
and has the special value “CS”. An edge \((v_i, v_j) \in E\) means that a service of category \(v_i\) precedes a service of category \(v_j\).

**Definition 6** - An e-government service \(S_i\) precedes \(S_j\) if there are operations \(op_{ik}\) and \(op_{jl}\) belonging to \(S_i\) and \(S_j\) respectively so that (i) \((\text{Signature}(op_{ik}) = \text{"notification"})\) and \(\text{Signature}(op_{jl}) = \text{"one-way"})\); or (ii) \((\text{Signature}(op_{ik}) = \text{"solicit-response"})\) and \(\text{Signature}(op_{jl}) = \text{"request-response"})\). □

We define a particular type of templates called *stored templates*. These are templates defined *a-priori* by government agencies. For example, social services agencies would agree that a pregnancy benefits composite service combines food, health, and children services. Since stored templates provide in essence added values, they are used to test the soundness of composition plans. Intuitively, a composition plan is *sound* if it provides part or all of the interactions modeled by a stored template.

**Definition 7** - A composition plan is *sound* if its template is a subgraph of a *stored* template. □

### 3 Composing E-Government Services

The process of composing e-government services consists of two phases: specification and matchmaking. The specification phase allows composers to provide high level descriptions of the desired compositions. We define an XML-based language, called CSSL (*Composite Service Specification Language*) to cater for the the specification of composite services. By using CSSL, composers only need to have a general idea about the service they are interested in offering. They are not required to be aware of the full technical details such as descriptions of component services, their characteristics (e.g., data types), and how they are plugged together.

CSSL uses a subset of WSDL elements and extends this subset to enable the description of semantic features of composite services. The definition of WSDL-like language for composite services makes the description of composite services as simple as the description of basic (i.e., non-composite) services. Additionally, it allows the support of recursive composition of services. Composite services specified in CSSL can be used as components for new compositions. We give below a subset of the CSSL specification of the pregnancy benefits composite service:

```xml
<service name="pregnancy benefits" category="pregnancy"/>
  <protocol name="SOAP"/>
  <message name="profile">
    <parameter name="monthlyIncome" type="decimal"/>
    <parameter name="familySize" type="short"/>
  </message>
  <message name="reply">
    <parameter name="eligibilityStatus" type="boolean"/>
  </message>
  ....
  <operation name="foodProgramEligibility"
    signature="solicit-response" type="eligibility"
    category="nutrition">
    <input name="profile"/>
  </operation>
```

The abovementioned specification states that *pregnancy benefits* service uses SOAP protocol. The service is related to “pregnancy” domain as mentioned in the category attribute. The specification includes a *foodProgramEligibility* operation. This operation allows to check eligibility (*type attribute*) for food programs (*category attribute*). It sends a message containing citizen’s profile (*output element*) and receives a response containing the eligibility status (*input element*).

The second phase of the composition process is the *matchmaking* phase. The aim is to use compatibility rules for generating composition plans that are conform to composers’ specifications. For that purpose, we propose a *matchmaking algorithm*. The input of the algorithm are the CSSL service specification and a
repository of WSDL services. We use UDDI business registries [2] to store WSDL descriptions. The general premise of the algorithm is to map each operation \( op_i \) of the composite service with one or more WSDL service operations \( op_j \). The algorithm looks for services with a category similar to the category of \( op_i \). We use UDDI inquiry interface [2] for discovering such services. Then, the algorithm checks that interacting services have compatible transport protocols (Definition 2). For every pair of operations \((op_i, op_j)\), it also checks signature compatibility (Definition 1), message compatibility (Definition 4), operation semantics compatibility (Definition 5), and composition soundness (Definition 7).

4 Implementation

We provide an implementation of the proposed approach on top of WebDG prototype [8, 9]. Figure 1 depicts the extended WebDG architecture. In the following, we focus on the new features added to WebDG (service composition, UDDI registry, and SOAP invocations). Details about the other features can be found in [8].

Figure 1: WebDG Architecture

Users (citizens and case officers) access WebDG via a Graphical User Interface (GUI). The GUI is implemented using HTML/Servlet. Users’ systems interactions go through the Request Handler (RH). Users discover WSDL services of interest via the Service Locator (SL) which provides access to the UDDI business registry. We use Systnet’s WASP UDDI Standard 3.1, a UDDI API 2.0 compliant registry implemented in Java. WASP UDDI Standard allows automated service lookup through the standard SOAP protocol. Once a service is discovered, its operations can be invoked using SOAP through the RH module.

CSSL specifications are handled by the Composite Service Manager (CSM). The CSM forwards each specification to the Service Description Parser (SDP) to parse the corresponding XML documents. The SDP uses JAXB (Java API for XML Processing). It returns back a data structure containing composite service operations, messages, and bindings. The SCM forwards this data structure to the Syntactic and Semantic Matchmaker (SSM). The SSM module implements syntactic and semantic compatibility rules.
If no plan is generated, it returns an error message to the composer via the CSM module. The SSM sends the category of each operation to the Service Locator (SL). Only services with a category similar to the operation’s category are retrieved. The SL forwards the WSDL description of each located service to the SDP module for parsing. The SDP returns back a data structure containing component service operations, messages, and bindings. The SL forwards this data structure to the SSM module. Each generated plan is sent to the Compositional Matchmaker (CM). This module checks the soundness of the corresponding template. CM accesses the Stored Templates (ST) repository (mSQL database). Stored templates are kept in a table containing a template number, source and target categories. The CM returns composition plans with their compatible stored templates (if any) to the CSM.

5 Conclusion

We proposed a declarative framework for composing e-government services. We also presented a prototype implementation based on emerging Web service standards (WSDL, UDDI, and SOAP). We are currently defining Quality of Composition (QoC) parameters to help composers in selecting a composition plan that best meets their expectation.

Acknowledgment. This research is supported by grant 9983249-EIA from the National Science Foundation’s Digital Government program.

References