

# Integration of Text-based Applications into Service-Oriented Architectures for Transnational Digital Government

Andréa Matsunaga, Maurício Tsugawa and José A. B. Fortes

Advanced Computing and Information Systems Laboratory

Dpt. of Electrical and Computer Eng., University of Florida

PO Box 116200, Gainesville, FL, 32611-6200, USA

1 (352) 392-4964

{ammatsun, tsugawa, fortes}@acis.ufl.edu

## ABSTRACT

Significant efforts are currently being pursued by several countries and IT providers to deploy SOA (Service Oriented Architecture) designs of digital government systems that integrate or implement workflows of multiple software services and data sources. Unfortunately, many existing applications that can be useful in digital government are not implemented as Web Services, a fact that complicates their integration and interoperation within SOAs. To address this problem, this paper presents an approach to easily wrap text-based applications into Web Services. Compared to other application-wrapping approaches, this paper's solution exposes a simpler interface to users, completely hiding the complexities of understanding and developing Web Services. The approach is motivated by, and effective for, the important case of interactive applications, which is harder than batch-oriented applications and has not been considered by other approaches or software development environments. The paper briefly reviews a transnational digital government (TDG) project that requires interoperation and integration of independently developed geographically distributed information processing tools. The characteristics of SOAs are briefly described, along with their suitability for TDG systems and how they can be developed and deployed. The applications underlying the services needed for TDG are introduced and their SOA-relevant characteristics are identified. A framework is described for turning these applications into Web Services that are secure, support interactivity as needed, and do not constrain application functionality. The use of this framework and the evaluation of its benefits are described in the context of the deployment of application services needed by the TDG project.

## Categories and Subject Descriptors

H.3.4 [Information Storage and Retrieval]: Systems and Software - *Distributed systems*; D.2.12 [Software Engineering]: Interoperability; H.3.5 [Information Storage and Retrieval]: Online Information Services – *Data sharing, Web-based services*.

## General Terms

Design, Management, Standardization.

## Keywords

System integration, system interoperability, system management, transnational information system.

## 1. INTRODUCTION

Many efforts are under way in the USA and abroad to implement Service-Oriented Architecture (SOA) approaches to new and existing government processes [43][8][13][40][11][10]. An urgent need exists for effective methods to refashion applications as Web Services that can be integrated into enterprise architectures that span entire agencies and/or enable cross-agency collaborations. This paper presents an approach to easily wrap text-based applications into Web Services. This type of applications, often also called command-line applications, are common in systems and tools that process commands or data in textual form. While this paper's approach is generally applicable, it is motivated by the need to integrate machine translation software, conversational interfaces and a database query system into an SOA for Transnational Digital Government (TDG) to enable collaboration among government agencies in different countries. Without the approach proposed in this paper, the integration of interactive software tools would, at best, take unacceptably long times and incur in large labor costs, and, in the worst case, be impossible due to missing know-how or supporting infrastructure at the government agencies.

Simply stated, an SOA for a set of government processes consists of a collection of loosely coupled services that can interoperate with each other. Interoperability is achieved by using standard languages for the description of service interfaces and the communications among services. A widely accepted technology for implementing SOAs, called Web Services (WS)<sup>1</sup> [21], uses WSDL (Web Services Definition Language) and SOAP (Simple Object Access Protocol) as the standard languages for definition of, and communication among, services, respectively. More generally, WS technologies also adopt several other standards for service discovery, workflow orchestration, reliable messaging, security, etc. The planned widespread adoption of SOAs for digital government stems from the fact that, once services are known and available, government processes are more easily created, maintained, integrated and reused. This, in turn, can enable agile government processes and facilitate new forms of

---

<sup>1</sup> The abbreviation WS is used to refer the technologies used to implement SOAP-based Web Services. When referring to a service implemented using WS this paper uses either the term Web Service or the term Service. Generic services that are not necessarily Web Services are referred as "services" (with lower-case "s").

cross-agency interactions and public access to government functions and information. SOA's and WS technologies are used at the services layer of the Federal Enterprise Architecture which has been adopted by the USA government [13].

One of the benefits of using SOA and WS is the decoupling of service interfaces from service implementations. In principle, once a service is created, it is possible to update, replace or modify its implementation without changing its interface and affecting other services. Given an existing implementation of an application to be integrated as a service into a government process, two tasks must be addressed. First, an execution environment (i.e. the hardware and other software needed to run the application) must be provided. Second, the application must be "service-enabled", i.e. additional programming is needed to "extend" the original application to behave as a service and to interact with other services. These two tasks can be particularly challenging when services based on existing applications (also called legacy applications) must be deployed across different agencies with possibly different IT infrastructures. The transnational government case that provided the motivation for the work reported in this paper is a concrete example of a scenario where these two challenges arise.

Many transnational government processes, including the case-study considered in more detail in Section 5 of this paper, require government agencies of different countries to engage in collaborations that entail accessing and sharing information. Typical requirements include agencies being allowed to query databases of other agencies, being able to invoke translation tools to enable communication across different languages and having access to dialog-oriented interfaces to enable users to easily issue queries. These are information processing tasks for which there already exist (legacy) codes which, if reusable in the context of SOA, can enable the quick development and deployment of effective systems. The alternative of redeveloping these codes would either be too time consuming, too expensive or both. However, in our experience, the available legacy applications run on different operating systems and require additional software that is not available in the agencies' IT infrastructure. In addition, the legacy application codes preceded the existence of WS technologies and could not interoperate in the context of a SOA. A naïve approach to these problems would require the acquisition of additional hardware and software (to address the first problem), and the acquisition of programming skills and labor to manually modify existing applications (to solve the second problem). This naïve approach leads to large unaffordable expenses which are recurring in nature due to the unavoidable need to update, extend and modify the system implementation.

In [22] the authors proposed the use of virtualization technology to address the first challenge, i.e. the problem of providing different execution environments for multiple services within agencies with distinct IT infrastructures. The key idea is to use readily available virtualization software (VMware [44], Xen [5], and VirtualPC [24]) to create virtual computers on which the software needed to run existing applications can be installed. This allows the encapsulation of conflicting execution environments in separate virtual machines, thus eliminating any incompatibilities among software environments already in place at the agencies and the execution environments needed by different applications. The authors have also developed software that leverages network

virtualization technologies (e.g., VLAN [30], and VPN [14]) to enable the creation of virtual networks to connect distributed resources (virtual or physical) thus making it possible to securely create and couple distributed services [42]. While machine and network virtualization address conflicting requirements at the hardware, software and networking levels, Services and SOA can be used to address software interoperability issues. Towards this goal, the focus of this paper is on the second challenge - how to enable applications as Services in order to allow them to interoperate with other Services.

At a very basic level, there are some key requirements for a Service to run as such. First, a container is needed, i.e., software that is needed by Services to execute as programs written in a given language (e.g., Java) to communicate and interface with other Services and to invoke the underlying application(s). In other words, a container provides a runtime environment for Services. Second, for a service to be useful, code is needed to invoke the Service (the client side) as well as to provide the Service (the server side). In theory, the code needed to implement a Service should work independently of the container used. However, in practice, Web Service developers must be very knowledgeable and careful about the containers and languages to be used. If done manually, enabling a complex legacy application as a Service can be a time-consuming expensive process. This is particularly true since WS technologies are still relatively new, lack sophisticated programming environments and require non-specialized programmers to learn new skills and tools. In contexts such as those of multiple agencies whose IT infrastructures are independently administered, since there could be different containers and languages being used, the expertise and effort needed to turn applications into Services increases in proportion to the number of agencies involved. While the virtualization approach discussed earlier can be used to alleviate this problem, it needs to be complemented by tools to automatically enable applications as Web Services.

For the above-mentioned reasons, the automatic deployment of Services is an active area of research and development in academia and industry. GAP [34], GFac [32], SoapLab [35], and VAS [23][46] are some proposed approaches developed for scientific applications to be executed in Grid computing environments. These Grid-oriented solutions lack important features needed for digital government applications, namely:

- *Easy wrapping of text-oriented applications into WS:* although existing enabling tools hide many of the complex aspects of WS deployment, it is still challenging for users to have to deal with input formats or the infrastructure required by the enabling tools themselves.
- *Support for interactive applications:* existing solutions, in general, only support batch applications.
- *Simple mechanisms for authentication and authorization:* the goal is to avoid using complex security solutions not yet available across all vendors and platforms such as Grid Security Infrastructure (GSI) [15] and WS-Security [29] when other commonly used techniques are sufficient.
- *Platform independence:* existing tools target Linux environments on x86 machines and cannot completely or easily handle applications developed for other platforms.

This paper’s approach addresses challenges faced in developing and deploying distributed SOA systems, focusing on the problem of turning existing applications into Web Services in a simple and efficient manner. The main contributions of the paper are:

- A brief review of existing WS development and deployment environments;
- A tool called CLAWS developed by the authors; when used with Java, Tomcat and Axis, CLAWS provides a framework for rapidly wrapping text-oriented applications as Web Services; the approach addresses the above-listed issues, targeting interactive and stateful applications which are common in digital government domains;
- An evaluation of the effectiveness of the proposed approach and CLAWS in the context of the Transnational Digital Government Project.

The paper is organized as follows. Existing environments for WS development and deployment are described in Section 2. Section 3 classifies applications with respect to characteristics that affect how they can be enabled as Services. Section 4 proposes a framework that facilitates the deployment of interactive text-based applications as Services, by automatically executing the necessary wrapping process once a very simple description of the application is provided as input. Section 5 presents a case study in the context of the TDG project. Section 6 describes related work and Section 7 presents conclusions.

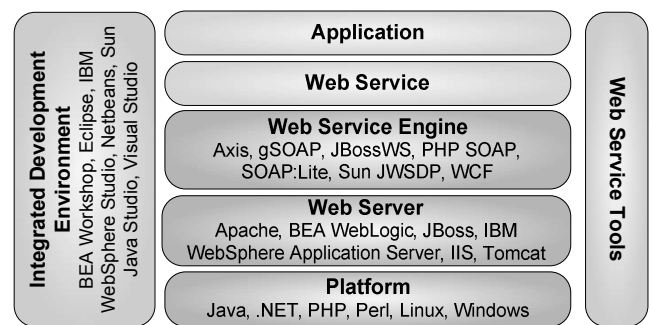
## 2. WS DEVELOPMENT AND DEPLOYMENT

A Web Services *development environment* is similar to a typical software development environment, making use of existing programming languages and compilers. The key difference is that it needs to interface with specific libraries or artifacts created by a *WS engine* vendor (see below). *IDEs* and *WS tools* facilitate the development of Web Services by exposing WS APIs, providing intuitive graphical user interfaces, generating skeleton codes, automating the deployment and testing phases of a development cycle. Figure 1 also illustrates the software components stack of the WS deployment environment (also called runtime), where Web Services actually run. The stack includes the following:

- *Application*: software that implements the functionality to be provided (e.g., a calculator, a translator, a query processor, a natural language parser). It may already exist or can be developed anew.
- *Web Service*: component that receives requests from the Web Service engine and invokes the appropriate application. Results are transmitted back to the engine. For applications developed in languages such as Java and C#, this could simply be annotations added to the existing application.
- *Web Service engine*: Web Services container, commonly a web application that implements SOAP in order to expose a standard and interoperable interface. The engine processes SOAP messages, forwarding the necessary information to appropriate Web Services in the appropriate format. It also encapsulates a Service’s response into SOAP messages to be transmitted back to the requesting client (e.g., Apache Axis[2], gSOAP [17], PHP SOAP [31], SOAP::Lite [36],

Sun Java WSDP [39], and Windows Communication Foundation [27]). WS tools are in general provided by the WS engine vendor and IDEs make use of these tools targeting a specific Web server.

- *Web server*: Web application container that implements HTTP for interactions with clients (e.g., Apache [3], BEA WebLogic [6], IBM WebSphere Application Server [18], IIS [25], JBoss [20], and Tomcat [4]). Sophisticated web servers tend to support more specific types of web applications.
- *Platform*: runtime environment of Web server, WS engine, Web Service and application (e.g., Java VM, .NET CLR, operating system with support for some specific programming language).



**Figure 1: A typical runtime environment for a Web Service is composed of a platform where a web application server runs Web Service engines used to make an application available as a Web Service. IDEs and WS tools facilitate the Web Service development.**

As previously mentioned, Web Services are defined using WSDL, the Web Service Definition Language. There are two main approaches to develop Web Services:

- *WSDL-to-code*: From the WSDL description of the service interface, this approach generates the code for both the server and the client sides of the Service.
- *Code-to-WSDL*: This approach starts from existing code and then derives the WSDL for the selected application programming interface (API) to be exposed as Service. In this context, the “existing code” could be one of the following: the source code of an existing application; binaries of applications written in a language that offers reflection (e.g., Java and C#); or the source code of the API in the case of applications to be developed anew.

WS developers recommend the first process, as the likelihood of Services working across different implementations of WS engines and Web servers is highest when the WSDL interface is defined following all standard specifications, in particular, meeting the WS-I Basic Profile 1.1. When exposing existing applications as Services, often times it is easier to start from the source code and automatically generate the WSDL files using some tool offered by the WS development environment. However, this automated process may still lead to interoperability issues due to the generation of WSDL constructs that are not fully supported in different environments. This is evident from the fact that WS engines from different vendors are not fully interoperable, even when developing Services for applications using the same

programming language (e.g., Axis and Sun JWSDP). In spite of the existence of tools that facilitate the creation of Services, in many occasions, the development of Services is still a hard task for the following reasons:

- WS development tools known to a developer may not support the language used to develop the application.
- Required modifications in the source code of applications can be too complex for programmers who are unfamiliar with the application or are not advanced WS programmers.
- The source code of the application may not be available.
- Automatically generated code may not be complete, and specialized knowledge is required to add the missing code and/or fine-tune the resulting Web Service code.

Recent versions of Integrated Development Environments (IDEs) support the development of Web Services. However, in general, they tend to concentrate on a specific version of WS technologies and target a specific Web server container. For example, Visual Studio 2005 [26] is an IDE that supports .NET Framework 2.0 and targets the Internet Information Services (IIS) container. IDEs facilitate not only the development but also the deployment and test of the Services. Solutions that integrate the various WS tools such as soapUI [37] also exist. These tools facilitate the creation of WS artifacts targeting a specific version of WS engine. Still IDEs and integration tools do not solve issues pointed above or may not offer facilities for the developer in certain environments. For instance Netbeans 5.5 [28] offers support for Sun Java WSDP to be deployed on either Tomcat or Sun Application server containers. While this IDE enables the developer to test a Web Service deployed to a Sun Application Server from the IDE itself without the need to create new code, the same is not true when the desired web server container is Tomcat.

Commercial and open source systems that support Enterprise Application Integration (EAI) (e.g., BEA WebLogic, IBM WebSphere, JBoss) focus on modernizing and consolidating applications in an enterprise, exposing them not only as WS but possibly as RPC, RMI, MOM, and JCA among others (see [19] for an overview of these technologies). When dealing with legacy applications, complex tasks of configuration and programming of adapter components for each legacy application are required. Once again, the cost of EAI systems, the necessary support from the vendors and large labor costs might not be affordable. When using SOA for EAI, exposing existing applications as Services faces the problems discussed above. In this context, vendors provide ready-to-use adapters only for known legacy protocols (e.g., IMS, CICS, enabling COBOL and PL/I applications). The solution proposed in this paper complements SOA-based EAI by simplifying the process of WS-enabling text-based legacy applications independently of the programming language used and without requiring the existence of data access mechanisms.

The conclusion from this section’s brief review of the issues faced in the development and deployment of Web Services is that developers still need to find workarounds that address potential interoperability issues. When possible, developers can tweak the WS technologies, add features to the existing tools, and propose changes to be incorporated by the vendor, but these are time-consuming steps that are not for the faint of heart.

The solution proposed in this paper targets text-based applications, including interactive ones (specific characteristics of these applications are discussed in the next section). It does not require source code to be available, thus achieving programming language independence and avoiding the need to modify existing applications. An important advantage of the proposed solution is that it enables IT staff to keep Services operational even when applications are updated. This is accomplished even when IT staff have minimal or no knowledge about the particularities of a particular WS engine or WS standards.

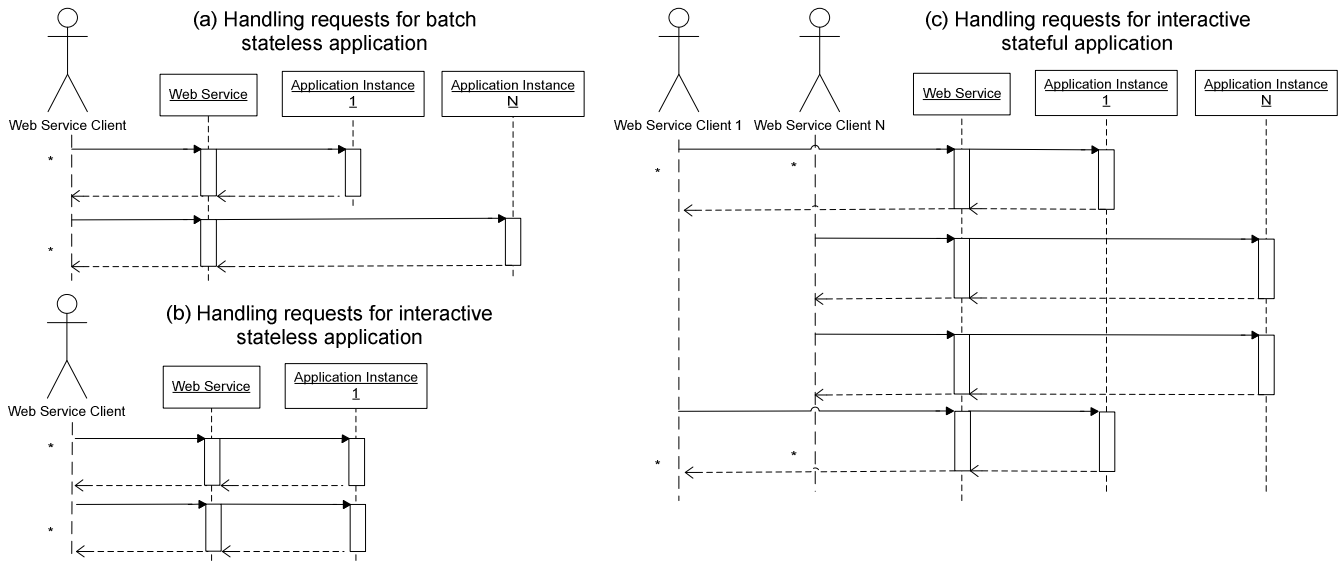
### 3. APPLICATION CLASSIFICATION

The approach and coding needed to wrap an application as a Service depends on the application’s characteristics, in particular on how it behaves at runtime. This section differentiates several types of applications in order to identify the features that an application wrapper service (proposed in the next section) needs to provide. Applications can be invoked via a graphical user interface (GUI), a text-based interface (process standard input and output) or an application programming interface (API). Focusing on the particular type of application that is supported by the solution presented in this paper – text-based applications – they can be further classified as shown in Table 1.

**Table 1. Text-based application classification**

Non-interactive (or batch)	Applications that run without any user interaction. Input parameters are specified when starting the application and output is collected at the end of execution.
Interactive	Applications in which users enter individual commands and control the execution flow. Commands are processed and results are returned immediately.
Stateless	Applications in which the processing of a command does not depend on data computed by the application in response to past commands
Stateful	Applications in which the outcome of a command depends on data computed by the application in response to past commands

Well-known commands used in operating systems (OS) can be used as examples that illustrate the types defined in Table 1. Non-interactive applications are necessarily stateless (e.g., echo, ls, dir), while interactive applications may be stateless (e.g., cat, copy files) or stateful (e.g., sh, matlab). From a WS perspective it is important to understand the programmatic means of accessing an application at runtime. Text-based applications are not invoked directly by the clients. They are invoked by WS engines once client requests are processed. Particular care needs to be exercised when turning a text-based application into a Web Service, since the Service invokes and interacts with an application that is executed outside of the WS container. Further, for interactive applications it is important for the Web Service to identify its users and decide how to maintain application state(s). In some scenarios it is desired for application state to be shared among all Service users (in which case the Service invokes and interacts with one instance of the application). In other scenarios, the application state must be independently maintained per user or session identification (in which case the Service instantiates one application per user or per session identification).



**Figure 2: Sequence diagrams depicting the necessary interactions between WS client, Web Service, and application for different types of application. For batch stateless applications (a), one application instance is created for each request whereas for interactive stateless applications (b), an application is instantiated once and re-used over several requests for multiple clients. However, for interactive stateful applications (c), an application instance is created and shared among requests from a specific user or a group of users, according to the access control mechanism chosen while WS-enabling the application.**

Depending on how they are invoked, applications can have more than one type of usage mode. For example, the machine translation system used in the TDG project (see Section 5 for details) provides two executable programs: one that translates from English to Spanish (eng2spa) and another from Spanish to English (spa2eng). It accepts the text to be translated as input (acting as batch stateless application) or it can expose a console-based user interface, in which case it translates multiple sentences following user's input (acting as interactive stateless application). The communicator system (CS) used in the TDG project is an example of interactive stateful application. CS exposes a console-based user interface where users can type queries in natural language (e.g., 'who entered Belize in December 2006?'), which are parsed, generating queries that are sent to the actual databases. CS keeps the conversation state, i.e. a subsequent query could be 'list the passengers coming from USA' (among those who entered Belize).

The Unified Modeling Language (UML) sequence diagrams in Figure 2 provide the sequences of interactions that occur between a Web Service client and a Web Service for the three types of application identified above. One instance of batch stateless application must be created in order to handle each request from Web Service clients (Figure 2a). Interactive applications (Figure 2b, Figure 2c) are instantiated and reused over several requests. In case of interactive stateless applications, one or more instances of applications can be created, and requests, independently of the requesting Service client, can be directed to any application instance, opening opportunities for load balancing. In Figure 2b only one application instance handles all requests, thus the Service is responsible for forming a queue and only allowing one request to proceed at a time. Handling interactive stateful applications requires the Web Service to identify the requesting Service client in order to direct requests to the appropriate application instance that keeps state for the client. Figure 2c

shows that requests from the Service client 1 are always directed to application instance 1, whereas requests from Service client N are always directed to application instance N. In this case a Service client can be a single user of a group of users.

#### 4. CLAWS

The tool proposed in this paper, called Command-Line Application Wrapper Service (CLAWS), focuses on interactive text-based applications (but also supports batch applications), and automatically wraps, builds and deploys an application onto a web-server container hosting a particular WS engine (Axis). CLAWS enables its users (in general, IT staff) to easily generate a Web Service for existing interactive text-based applications through a web form without expecting the user to have knowledge about the various WS standards or to write additional code.

The information that needs to be provided to CLAWS to initiate the automated process consists of a

- *Service Name*: name of the Service to be generated;

and, for each operation to be provided by the Service, a data set composed of the following:

- *Operation*: name of the Service operation, along with the name of input attributes for the Service operation; the input attribute name can be referenced when specifying the command-line, the environment variables or the standard inputs (see below); support for arrays of inputs is also available, enabling the creation of parameter-sweep applications, i.e. applications that are executed for a range of parameter values instead of a single value.
- *Command-line*: entire command-line exactly as invoked by the deployed Web Service.

- *Environment variables*: set of environment variables to be set before the execution of the application.
- *Current working directory*: indicates the directory from where the application will be invoked.
- *Standard Input*: information sent to the standard input of the command-line application process.
- *Interaction end-detector pattern*: interactive applications accept inputs from standard input and output results; as the Service proxies the interaction between the Service client and the application, it needs to detect, based on the output of the application, when the processing of an input finished; the end of an interaction is in general detected when a specific pattern is output (e.g., a prompt); this parameter indicates how the Service can detect the end of an interaction.
- *Rewrite rules*: pairs consisting of a pattern and a value so that if the pattern matches parts of the output, each part is replaced by the corresponding specified value; this is especially useful when a prompt is part of the output of the application, but it should not be part of the Service response.
- *Mode*: indicates the mode of operation of an application - as detailed in Section 3, it can be batch, stateless interactive or stateful interactive (Figure 2); when an application is configured as interactive, the waiting time, in the form of number of tries to match the end pattern with the interval between attempts is also required; the deployed Service will monitor the output of the application, and will try to match the end pattern the specified number of times; note that an application that is interactive could be also executed as non-interactive (since, for each interaction, the application needs to be initialized, this is not recommended for applications with high initialization overhead).
- *Security configuration*: a Service may require access control mechanisms, in which case user authentication is necessary; when authentication is required, secure and well established HTTP basic authentication with Transport Layer Security (TLS) is currently supported by CLAWS; supported access control modes are as follows: no authorization scheme, user-based authorization, or role-based authorization.

The above-listed information is sufficient for the generated Service to offer exactly the same capabilities as the original application. The user of CLAWS can also choose to input different information so that the Service generated by CLAWS has a reduced set of capabilities (by not exposing options that the original application accepts), an increased set of capabilities (e.g., offering a parameter-sweep capability) or is a Service that combines several applications (e.g., inputs could determine which of several different programs would be invoked).

The process that CLAWS automates consists of the following steps:

- *Code generation*: generates the implementation of the Web Service responsible for instantiating the application according to the information given by the user enabling the application. The generated code assembles the command-line as specified, invokes the application according the mode of

operation specified, handles inputs and outputs and provides authorization when required.

- *Deployment descriptor generation*: specific deployment descriptors are generated depending on the targeted WS engine to instruct the engine on how to deploy the Service.
- *Building*: depending on the language used, the generated code is compiled before the Service can be deployed.
- *Deployment*: the compiled code is deployed on the targeted Web Service engine.

Service Name:	MyService
Operation 1:	echo message
Command 1:	cat
Input 1:	<message><end of interaction>\n
End pattern 1:	(?s).*<end of interaction>\n\$
Rewrite rule 1:	<end of interaction>\n\$
Mode 1:	Interactive and stateless
Security configuration 1:	User-based user1 user2
Operation 2:	store text filename
Command 2:	cat > C:\temp\<filename>
Input 2:	<text>
Mode 2:	Non-interactive and stateless
Security configuration 2:	Role-based users

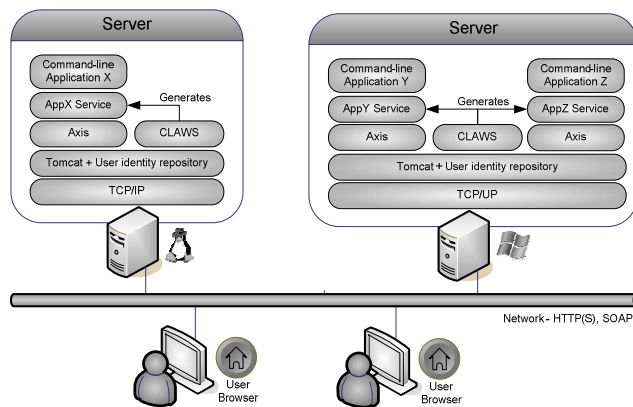
**Figure 3: Example of input to CLAWS to wrap an interactive text-based application as a Web Service called MyService. MyService offers 2 operations: echo message and store files. Both Service operations, when deployed, invoke a simple application, the OS command ‘cat’.**

To exemplify the CLAWS wrapping process, consider the commonly used OS command ‘cat’ as an application that concatenates files and prints them on the standard output. Suppose that the goal is to deploy a Service with two operations: one that copies the input data to output and another that copies the input to a file. Figure 3, shows the information needed to deploy a Service called ‘MyService’ with an operation ‘echo’ available for users ‘user1’ and ‘user2’ and an operation ‘store’ available for role ‘users’. For both operations the deployed Service invokes the ‘cat’ command. The first operation expects the cat application to be used interactively and each Service request from clients will specify different input messages. Messages are passed to standard input of the application with an extra string to facilitate the detection of the end of an operation (the example in Figure 3 - ‘(?s).\*<end of interaction>\n\$’ – illustrates the support for Java style regular expression for added flexibility). The second operation executes ‘cat’ in non-interactive mode and expects two inputs from the Service client: the text to be stored (parameter ‘text’) and the name of the file that will store the text (parameter ‘filename’). The folder where the file is stored was pre-defined by

the application enabler as 'C:\temp\' and the text input is sent to standard input of the application. Since 'cat' is a command that does not require environment variables to be set and it can be executed from any location, the current working directory and environment variables have been omitted in the figure.

The first operation 'echo' can be configured as non-interactive, and without the need to define an end pattern, it can still perform the same functionality of the example. This illustrates the flexibility offered by CLAWS which can support applications in both interactive and non-interactive modes. Furthermore, a tool like CLAWS can be configured in order to create a pool of stateless interactive applications processes to serve simultaneous requests from clients without serializing the requests (as shown in Figure 2b).

Figure 4 depicts how CLAWS, developed using Java 1.5, is deployed as a web application on Tomcat using JSP and servlets technology to deliver a web form similar to Figure 3 for its users. It is intended for generating and deploying Axis Web Services 1.4 into a Tomcat Web server container. This is a combination that can be easily deployed in most platforms (it was tested in both Linux and Windows), but if other combinations of programming languages, web servers and WS engines are required, CLAWS concept can be applied to the particular environment - this is planned as future work if and when needed. Note that although developed in Java, CLAWS can wrap text-based applications developed in any existing language and without requiring their source code or the manual development of new source code. Testing of the newly created Service can also be done through a web user interface.



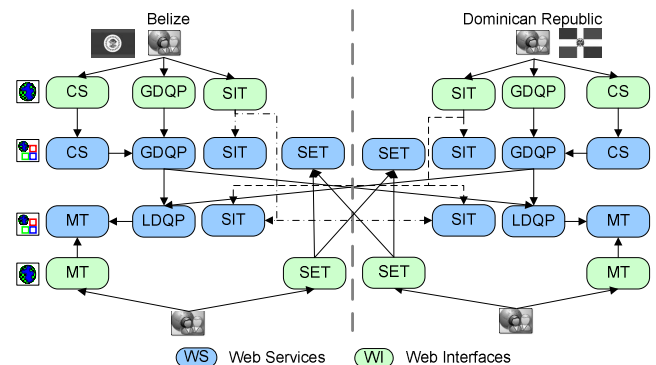
**Figure 4: The CLAWS framework offers a web interface for the user to enter information about the application to be enabled as Web Service for use by any other application on the network. Once the information is validated, the Service is generated, built and deployed in the same container as CLAWS, independently of the execution environment.**

Currently, CLAWS makes use of the commonly used basic HTTP authentication-based access control mechanism. This is sufficient and secure (when combined with TLS) in many cases of interest in digital government, including those faced in the TDG project. More complex solutions could also be implemented but are not currently supported by CLAWS. For example the Globus [15] project offers security features based on delegation and standard X.509 certificates. Security standards in WS are still evolving and

they are not fully supported in many stable WS engines. Another issue is the added expertise needed to maintain other aspects of these complex solutions - for example Globus-enabled resources.

### 5. CASE STUDY

The goal of the TDG project [38] was to deploy an information system that enables the sharing of immigration information among border-control agencies that collect and use such information in Belize and the Dominican Republic. The requirements of such a system included the ability to transparently query databases in different countries, the automatic translation of information expressed in a language other than the user's, conversational interfaces to facilitate queries by border agents, and supporting tools for global schema construction and other configuration tasks. For economic and other practical reasons, the information system had to be implemented without requiring modifications of the IT infrastructure of the national immigration agencies. In addition, the availability of software already developed by the project leaders for machine translation and conversational interfaces lead to the decision to reuse these legacy applications as the basis for the corresponding capabilities of the TDG system. The other components were developed anew. By design, the TDG system was conceived as an SOA whose key components were to be implemented as Services.



**Figure 5: Interactions among Web Services and Web interfaces built for the TDG information sharing project: an immigration agent in Belize issues questions in English using the communicator system (CS) Web interface, which in turn connects to CS Service to generate a global query to the global distributed query processor (GDQP) Service that splits the request to local distributed query processor (LDQP) Services in Belize and the Dominican Republic; the results are translated from Spanish to English (if necessary) using machine translator (MT) Services and returned to the officer.**

More specifically, the TDG system components that needed to be integrated as Web Services included: a Machine Translator (MT) system [9] which translates natural language texts from Spanish to English and from English to Spanish, a Communicator System (CS) [33][45] capable of maintaining a textual conversation with a user typing queries in natural language to the system, and a Distributed Query Processor (DQP) system [38] which integrates information from databases whose schemas are different. DQP consists of the following subcomponents: a Global Distributed Query Processor (GDQP), a Local Distributed Query Processor (LDQP), a Schema Export Tool (SET) and a Schema Integration Tool (SIT). As shown in Figure 5, each of these components was wrapped as a Web Service in order to run in the computing

environment required by the application and still be accessible to other components. Web interfaces were created for the end-users of these tools and integrated into a portal that runs in Belize and in the Dominican Republic.

The design of CLAWS (and the need for such a tool) is a direct result of the authors' experience in developing the TDG system through its two versions. For the first version, all Services were manually created, a time-consuming process that required exchange of information among groups at different locations and hard-to-coordinate schedules. Since MT and CS were programmed using the language C and developed for execution on Linux-based platforms, the Services originally created to wrap these applications were developed for such an environment. In a second version of the TDG system, since Windows systems were the dominant system at immigration agencies, an attempt was made to create and support MT and CS binaries for Windows. While CS was successfully ported to Windows, such port was not practical for MT as several incompatibilities were detected -- it was decided to instead concentrate the team's efforts on improving translation accuracy. During this process, Service wrappers had to be modified to support particularities of Windows for both CS and MT, as partial implementations of the MT became available. Moreover, during the course of this project, WS interfaces and the WS client library had to be maintained manually as new version of the applications became available. This labor-intensive redesign and maintenance activity, with constantly changing versions of components, is only possible if a WS specialist is available, which increases development cost and is not realistic at deployment sites. Once developed, CLAWS greatly enhanced the team's ability to cope and integrate these modifications (e.g., a new option, a new command location, removal or addition of an error message) into the TDG system, making it possible for it to be quickly updated and continuously available for testing over the Web.

Service Name:	Translator
Operation 1:	translate from to message genre
Command 1:	<from>2<to> -q
Input 1:	:genre <genre> <message>\n
End pattern 1:	(?s).*\n\$
Mode 1:	Interactive and stateless
Operation 2:	translate from to messages[] genre
Command 2:	<from>2<to> -q
Input 2:	:genre <genre> <messages>\n
End pattern 2:	(?s).*\n\$
Mode 2:	Interactive and stateless

Figure 6: Wrapping MT command-line as Web Service.

To exemplify the use of CLAWS in the TDG project, Figure 6 presents the information that was necessary to enable the MT application (a similar approach was used to enable CS application). Basically there were two command-line applications to be deployed in Belize and Dominican Republic servers: eng2spa, and spa2eng (which provides translation from English to

Spanish and from Spanish to English respectively). These commands are formed joining the 'from' and 'to' input parameters of the Service. The result of the translation can be obtained passing the 'message' parameter as input of the invoked application.

As pointed out before, MT is a stateless application that could be executed in a non-interactive way. However, as it has to load a large database with example sentences to tune the translations for a specific domain, initialization of the application takes more time than a single request for translation. Thus, it is best executed if implemented as an interactive application. MT Service is also an example of a Service that integrates more than one interactive text-based application into a single Service.

The net result of having CLAWS available is that Services exactly with the same characteristics of manually created ones can now be easily deployed. WS complexities are completely hidden, and software component updates are simplified.

## 6. RELATED WORK

The wrapping of existing applications as Services is a problem addressed by many projects in grid computing and scientific applications domain. Noteworthy approaches include: SoapLab[35], GridLab's Grid Application Toolkit (GAT)[1], Virtual Application Services (VAS)[23][46], Generic Application Service (GAP)[34], and Generic Application Factory (GFac)[32]. The main difference between CLAWS and these grid-oriented systems is that applications that needed to be supported in the TDG project present interactive and/or stateful characteristics that are not supported by previous work. In the scientific application domains that are typical of grid computing, large numbers of resources are usually involved in long-running jobs. In such scenarios, issues such as notification of execution progress and delegation of credentials become very important. CLAWS does not support these complex features, but it is flexible enough to support a wide range of applications necessary in digital government. The focus of CLAWS is on simplifying the processes of Web Services development/deployment by completely hiding the complexities of Web Services technologies and frameworks. For example, Web Services expertise is not required when maintaining the information-sharing system described in Section 5.

There are other specific differences between CLAWS and GAP, VAS, and GFac. Users of CLAWS are not expected to provide the information about an application as an XML file, as some users are still not comfortable with producing and editing XML files even with the use of XML editors. As CLAWS could include support for more configuration options to the users, it seems better to evaluate various solutions until a good tradeoff is established. For example, VAS and GAP can accept definitions of complex command-line options whereas GridLab and GFac only deal with simple command-line arguments. For this TDG project, in order to handle interactive applications, it was essential for CLAWS to support referencing names used in defining Web Services operations so to enable the construction of the command-line to be issued as well as having more control over the input. Other solutions also allowed more detailed description of Service inputs and outputs. This information would not be visible to end-users of TDG project as all the Services used by them were

located behind customized Web portals. In other words, if this information were to be truly used to automatically construct portals as in GAP, or GFac, there would be a need to allow the application configurations to be bilingual, since the TDG project needs to deal with two languages.

When compared to VAS and GAP, the Web Services interface supported in CLAWS is stricter with respect to acceptable parameter types – instead, a simpler framework that only works with strings was preferred. Complex types can be encoded as strings and processed by Web Services before forwarding to applications or multiple operations could be constructed to support the needed complexity.

The Web Services deployed by using CLAWS, or any other solution, require full network connectivity. In previous work we advocated (and implemented) the use of virtual networking, in particular ViNe [41], to enable connectivity in highly firewalled network environments, such as those existing between Belize, Dominican Republic and US in the context of the TDG project.

Finally, it is possible to envision CLAWS as a tool that can be used to complement toolkits currently available for a variety of Web Services frameworks, thus extending their Web Service enabling capabilities to include textual applications. This is possible because CLAWS has no dependencies on application programming language.

## 7. CONCLUSIONS

The integration of unmodified applications as Web Services is a common requirement of SOAs which are increasingly being adopted for digital government purposes. CLAWS, the tool described in this paper, automates this process so that its users do not require any knowledge of Web Services technologies and frameworks. As a result, significant savings in development and deployment time and cost can be achieved by using CLAWS as described in this paper. CLAWS can also be used to regenerate Services when applications or software are updated or modified, thus enabling additional savings in maintenance costs. These considerations are particularly important in transnational digital government systems. In the TDG project discussed in this paper two applications (for conversational interfaces and machine translation) developed initially to work isolated from other applications were required to be part of integrated immigration border control information system. CLAWS enabled the integration of these applications as Services and, indirectly, the implementation of intuitive and effective user interfaces in two countries with different languages and IT infrastructures.

## ACKNOWLEDGEMENTS

Research reported in this paper is funded in part by NSF awards EIA-0107686 and EIA-0131886. José Fortes was supported in part by the BellSouth Foundation. The authors acknowledge the members of the TDG project (for contributions to different aspects of the system): S. Su, T. R. Kasad and M. Patil, (DQP system); V. Cavalli-Sforza, J. Carbonell, and P. Jansen (MT); W. Ward, R. Cole and J. Brenier (CS); D. Towsley and W. Chen; A. I. Antón and Q. He; C. McSweeney, R. Bol, F. Rabbani (University of Belize) for the Belize IS; L. de Brens, J. Ventura, and P. Taveras (Pontificia Universidad Católica Madre y Maestra) for the Dominican Republic IS, R. Connolly, C. Ortega, and B. Piñeres (Organization of American States); O. Brooks (National Drug

Abuse Control Council, Belize); and M. Herrera (National Drug Council, Dominican Republic). Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the NSF.

## 8. REFERENCES

- [1] Allen, G. et al. The grid application toolkit: toward generic and easy application programming interfaces for the grid. *Proceedings of the IEEE*, March 2005, v.93, n.3, 534-550.
- [2] Apache Axis, from <<http://ws.apache.org/axis>>, or <<http://ws.apache.org/axis2>>.
- [3] Apache HTTP server project, from <<http://httpd.apache.org/>>.
- [4] Apache Tomcat, from <<http://tomcat.apache.org/>>.
- [5] Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM Symposium on Operating Systems Principles (SOSP)*, October 19-22, 2003, Bolton Landing, NY, USA.
- [6] BEA WebLogic Server. <<http://www.bea.com/framework.jsp?CNT=index.htm&FP=content/products/weblogic/server/>>.
- [7] BEA Workshop for WebLogic Platform, from <<http://www.bea.com/framework.jsp?CNT=index.htm&FP=content/products/workshop/workshop/>>.
- [8] Burk, D. The Government's SOA Roadmap. *InfoWorld SOA for Government Executive Forum*, September, 2005, from <[http://www.infoworld.com/event/soa/gov/docs/Pres\\_Dick\\_%20Burk\\_Handout\\_20050914.ppt](http://www.infoworld.com/event/soa/gov/docs/Pres_Dick_%20Burk_Handout_20050914.ppt)>
- [9] Cavalli-Sforza, V., Carbonell, J.G., and Jansen P.J.J. (2004). Developing Language Resources for Transnational Digital Government Systems: A Case Study. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation (LREC 2004)*, Lisbon, Portugal, May 24-30, 2004, v.III, 945-948.
- [10] California Enterprise Architecture Program – Program Charter. October 2005, from <[http://www.cio.ca.gov/ITCouncil/Committees/PDFs/project\\_charter\\_FINAL.pdf](http://www.cio.ca.gov/ITCouncil/Committees/PDFs/project_charter_FINAL.pdf)>
- [11] Danish Ministry of Science, Technology and Innovation. *The Architecture of the Danish OIO Service Oriented Infrastructure*. Draft version 0.8, Ministry of Science, Technology and Innovation, March 2006, from <[http://www.oio.dk/files/OIO\\_SOI\\_-\\_Architecture\\_v0.8\\_ENG.pdf](http://www.oio.dk/files/OIO_SOI_-_Architecture_v0.8_ENG.pdf)>.
- [12] Eclipse – an open development platform, from <<http://www.eclipse.org/>>.
- [13] FEA Program Management Office. *Federal Enterprise Architecture*. OMB, Executive Office of the President, February, 2004, from <<http://www.whitehouse.gov/omb/egov/a-1-fea.html>>
- [14] Gleeson, B., Lin, A., Heinanen, J., Armitage, G., and Malis, A. A framework for IP-based virtual private networks. *RFC2764*, Feb. 2000.

- [15] Globus. *GT 4.0: Security*. Retrieved December 2006, from <<http://www.globus.org/toolkit/docs/4.0/security/>>.
- [16] Global Grid Forum. *Global Grid Forum*. Retrieved December 2005, from <<http://www.ggf.org/>>.
- [17] gSOAP: C/C++ Web Services and Clients, from <<http://gsoap2.sourceforge.net>>.
- [18] IBM WebSphere Enterprise Service Bus. <<http://www-306.ibm.com/software/integration/wsesb/>>.
- [19] Janssen M, Cresswell A. Enterprise Architecture Integration in E-Government. In *Proceedings of the 38th Annual Hawaii International Conference on System Sciences*, 2005.
- [20] JBoss Application Server, from <<http://www.jboss.org/products/jbossas>>.
- [21] Kreger, H. *Web Service Conceptual Architecture (WSCA 1.0)*. IBM Technical White Paper, May 2001. Retrieved December 2005, from <<http://www-306.ibm.com/software/solutions/webservices/pdf/WSCA.pdf>>.
- [22] Matsunaga, A., Tsugawa, M., and Fortes, J.A.B. Virtual Machines in Transnational Digital Government: a Case Study. In *Proceedings of the 6th National Conference on Digital Government Research*, May 2005, 255-256.
- [23] Matsunaga, A., Tsugawa, M., Adabala, S., Figueiredo, R., Lam, H., and Fortes, J. Science gateways made easy: the In-VIGO approach. *Concurrency and Computation: Practice and Experience*, Wiley Press, October 2006 (online).
- [24] Microsoft. *Using Microsoft® Virtual PC 2007 for Application Compatibility*. White paper, August 2006. Retrieved December 2006, from <<http://www.microsoft.com/windows/virtualpc/techinfo/appcompat.msp>>.
- [25] Microsoft Internet Information Services. <<http://www.iis.net>>.
- [26] Microsoft Visual Studio, from <<http://msdn.microsoft.com/vstudio/>>.
- [27] Microsoft Windows Communication Foundation, from <<http://wcf.netfx3.com/>>.
- [28] Netbeans IDE, from <<http://www.netbeans.org>>.
- [29] OASIS. *Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)*. OASIS Standard Specification, February, 2006.
- [30] Passmore, D. and Freeman, J. *The Virtual LAN Technology Report*. White Paper, 1996. Retrieved December 2006, from <[http://www.3com.com/other/pdfs/solutions/en\\_US/20037401.pdf](http://www.3com.com/other/pdfs/solutions/en_US/20037401.pdf)>.
- [31] PHP SOAP extension, from <<http://www.php.net/soap>>.
- [32] Kandaswamy, G., Fang, L., Huang, Y., Shirasuna, S., Marru, S. and Gannon, D. Building Web Services for Scientific Grid Applications. *IBM Journal of Research and Development*, 2006, 50(2/3):249-260.
- [33] Pellom, B., Ward, W., and Pradhan, S. The CU Communicator: An Architecture for Dialogue Systems. In *Proceedings of the 6th International Conference on Spoken Language Processing (ICSLP)*, Beijing, China, October 2000, v.2, 723-726.
- [34] Sanjeevan, V., Matsunaga, A., Zhu, L., Lam, H., Fortes, J.A.B. A Service-Oriented, Scalable Approach to Grid-Enabling of Legacy Scientific Applications. In *Proceedings of 2005 International Conference on Web Services (ICWS-2005)*, July 2005, 553-560.
- [35] Senger, M., Rice, P., Oinn, T. Soaplab - a unified Sesame door to analysis tools. In *Proceedings of UK e-Science All Hands Meeting*, September 2003, 509-513.
- [36] SOAP::Lite for Perl, from <<http://www.soaplite.com>>.
- [37] soapUI, from <<http://www.soapui.org/>>.
- [38] Su, S., Fortes, J., Kasad, T. R., Patil, M., Matsunaga, A., Tsugawa, M., Cavalli-Sforza, V., Carbonell, J., Jansen, P., Ward, W., Cole, R., Towsley, D., Chen, W., Antón, A. I., He, Q., McSweeney, C., de Brens, L., Ventura, J., Taveras, P., Connolly, R., Ortega, C., Pineres, B., Brooks, O., Murillo, G. A., and Herrera, M. (2005). Transnational Information Sharing, Event Notification, Rule Enforcement and Process Coordination. In *International Journal of Electronic Government Research (IJEGR)*, April-June 2005, v.1, n.2, 1-26.
- [39] Sun Java Web Services Developer Pack, from <<http://java.sun.com/webservices/jwsdp/>>.
- [40] Treasury Board of Canada Secretariat. *The Government of Canada Service-Oriented Architecture Strategy – Statement of Direction*. Government of Canada, Enterprise Architecture and Standards Division, February, 2006, from <[http://www.tbs-sct.gc.ca/cio-dpi/webapps/architecture/sd-eo/sd-eo\\_e.pdf](http://www.tbs-sct.gc.ca/cio-dpi/webapps/architecture/sd-eo/sd-eo_e.pdf)>.
- [41] Tsugawa, M., and Fortes, J. A Virtual Network (ViNe) Architecture for Grid Computing. In *Proceedings of 20th International Parallel and Distributed Processing Symposium (IPDPS 2006)*, Greece, April 2006.
- [42] Tsugawa, M., Matsunaga, A., and Fortes, J.A.B. Virtualization Technologies in Transnational DG. In *Proceedings of the 7th Annual International Conference on Digital Government Research*, May 2006, 456-457.
- [43] United Nations. UN Global E-government Readiness Report 2005: From E-government to E-inclusion. United Nations publication, 2005, 270p, from <<http://unpan1.un.org/intradoc/groups/public/documents/un/unpan021888.pdf>>.
- [44] VMware, Inc. *Introducing VMware Virtual Platform*. Technical white paper, February 1999.
- [45] Ward, W. Understanding spontaneous speech: the Phoenix system. In *Proceedings of the Int. Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, April 1991, v.1, 365-367.
- [46] Zhu, L., Matsunaga, A., Sanjeevan, V., Lam, H., Fortes, J.A.B. Application Modeling and Representation for Automatic Grid-enabling of Legacy Applications. In *Proceedings of First International Conference on e-Science and Grid Computing*, December 2005, 24-31.