In Proceedings of the 2004 IEEE International Conference on Web Services (ICWS 2004).

Facilitating Automated Search for Web Services

Gerald C. Gannod*† and Sushant Bhatia
Dept. of Computer Science & Engineering
Ira A. Fulton School of Engineering
Arizona State University - Tempe Campus
Box 878809, Tempe, AZ 85287-8809
{ gannod, bhatia }@asu.edu

Abstract

Recent advances in the area of web services have enabled inter-organization sharing of data and data-oriented software services. The challenges of developing software in a service-oriented development environment include search, retrieval, and integration of services with client applications. Such applications can be dynamic in nature and may vary depending on current availability of services or on the current relationship between client and service organizations. As such, applications must be able to quickly locate and integrate different potential service components. In this paper we describe an approach for automating the process of searching for web services using signature matching and describe a new signature match criteria called the contains match.

1. Introduction

Recent advances in the area of web services have made the ability of organizations to provide access to data more accessible. Furthermore, organizations are able to specialize access to that data by providing other behavior or "services". However, at the essence of service-oriented development is the notion of software reuse. Services being provided must be integrated with client applications in a manner similar to how components are integrated in single host applications. As a result, many of the issues long faced by software developers with respect to software reuse are still present in today's world of web services including the need to address problems of *population* (e.g., where do the reusable "components" come from and how do they get added to the repository), *search* (e.g., how do you find components that are in the repository), and *integration* (e.g., how do you integrate the components into a client application).

*This author supported by National Science Foundation CAREER grant No. CCR-0133956.
†Contact Author.

In this paper we describe an approach for automating the web service search process. The approach applies the use of *signature matching* [1] as a means for facilitating web service search. The approach utilizes existing UDDI technology as a mechanism for identifying published web services and applies common tools such as full-text search engines [2] to achieve timely recall. In addition we demonstrate a number of tools that we have developed to support the approach.

2. Overview

The activities supported our approach involves three major steps. First, a UDDI server is either automatically or manually browsed to identify candidates for indexing within the UDDI database. This step is an infrequent operation or automated, depending on preference. Second, the candidates are decomposed, analyzed, and indexed in a two-part library containing service meta-data and decomposed WSDL specifications. Finally, services are retrieved using signature-based matching [1]. The first two steps occur infrequently and constitute a *population* activity where information about web services is collected. The final step occurs with each desired use of a service.

3. Repository Population

The basic process used to populate the service index is the following. First, a UDDI server is searched in order to find candidate services for storage in a local search server. Second, two parallel operations are performed: a) an index to each web service is created that captures the publisher ID, the service name, the access point, and tModel, and b) an index to each web service is created that captures signature information for each web service method including name, return type, and input parameters. Third, the indices are saved in a database. The purpose of using this approach over current web service-based development approaches is two-fold. First, the search for services from a UDDI server

can be a time-consuming activity. By using our approach, this activity can be made more infrequent and focuses primarily on identifying classes of services rather than specific services themselves. Second, the effort spent searching for specific services can be reduced at application development time and can be more readily automated if specifications are decomposed and indexed off-line.

Figure 1 shows a high-level diagram depicting the elements involved in the population process. A UDDI server is searched either automatically or manually. Once services are selected, they are analyzed and stored into a database that captures meta-data and function information.

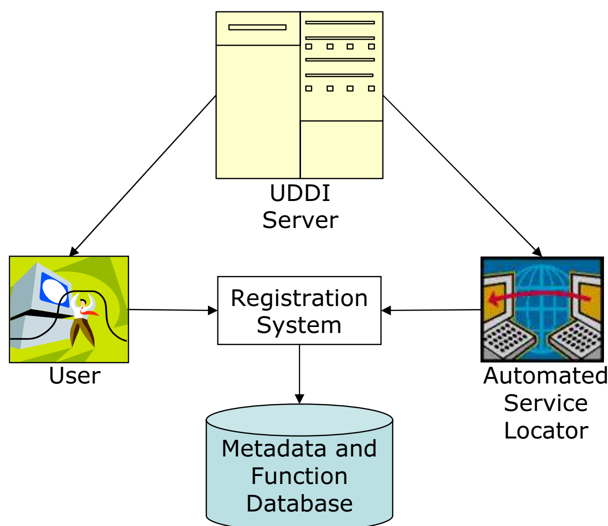

Figure 1. Repository Population Process

3.1. Automated Population

The automated search facility essentially works by searching UDDI servers for all services that meet basic search criteria. As part of the automated population operation, WSDL specifications are parsed and decomposed into function-based information. In effect, the automated facility can be used as an agent that searches UDDI servers for services on a regular basis and upon finding services that match various constraints, add those services to the local repository.

The advantage of using this automated population technique is that the repository that is constructed via this process can be easily updated with new services. A potential issue, however, is the fact that the automated search of a UDDI server can have a “floodgate” effect, thus populating the database with potentially superfluous, redundant, or undesired services. To alleviate this issue we are currently working on using UDDI registry information to help narrow the pool of services that are included in the repository. Furthermore, as described below, our approach supports the use of user guidance for identifying web services for inclusion in the database.

3.2. User Guided Population

The process for manually adding services to the repository is similar to that used in the automatic population case. The primary difference is in the front-end search of the UDDI server. We have developed a web-based application that is used to facilitate browsing of UDDI servers for services. The application uses a tree-based representation to support rudimentary search for services and provides functions for describing and saving services in the service repository.

4. Search

The second major aspect of the approach described in this paper is the use of signature matching [1] to facilitate search and integration during service-oriented application development. A typical development process for creating service-oriented applications might look something like the following. A developer is interested in building an application so they visit a UDDI server to determine whether a service exists that meets their application needs. Upon finding a service, the developer creates their application so that it conforms to the signature of the given service.

Contrast the above with the approach being supported here. Instead of searching a UDDI server to find a specific service, the developer identifies a class of services from which they wish to find a candidate service. For instance, they might be looking for a weather service. The class of service is used to launch the automatic service locator in order to include available services into their local service database. If the developer has developed applications within this domain in the past, in all likelyhood the local service database will already have information about potential services contained in the index and thus, does not need to perform the above population operation and can move onto application development. Next, using the design for their current application, they identify the expected types that might be used by a service that provides the information they are looking for. Using those types, the developer creates a signature search string that is used to search the local database for services that match that signature. Upon finding a match, the developer incorporates the given service into their application. The advantage of this approach is that several candidate services can be considered, and by using a facade pattern, a single interface provided to all of the candidates. As such, the choice of service can be determined at run-time rather than at development time.

4.1. Full-Text Search

Full-Text Search is an indexing mechanism that allows users to perform queries against character data [2]. It also allows an index to be updated as a background process. In our approach we use a full-text search that is enabled by

the use of a pre-compiled index. Unlike a query that might use a predicate based search, this allows for efficient search in catalogs that contain at excess of over one million rows or items. There are two components involved with full-text search: full-text *indexing* and full-text *querying*. Full-text indexing is responsible for the initial population of the index as well as updating the index when tables are modified. Full-text querying takes a full-text predicate (Contains, Freetext, ContainsTable, FreeTextTable) from a database that supports full-text search and transforms it into a command tree that is sent to a search service.

Within this framework, we support *Exact Match* and *Transformation Match*, both introduced by Zaremski and Wing [1]. We have also developed a new form of signature-based match called *Contains Match*.

4.2. Exact Match

When using *Exact Match*, a user provides an expression for the signature they want to search for in the repository. If the expression exactly matches a row of in the repository then the conditions for Exact Match have been met and that function is added to a list of candidate services.

To illustrate exact match, consider a function that is part of a Railways web service called `GetRailTicketQuote` with the following definition in a C# or Java like syntax:

```
String GetRailTicketQuote (String Location,
String Destination, DateTime Departure,
DateTime Arrival)
```

The corresponding expression stored in the function repository would appear as follows:

```
String, String, DateTime, DateTime -> String
```

Later, when a user provides a search string that has the exact same signature as the one given above, information regarding the details of the service (and others that match the signature) are retrieved including the name of the module or service, the name of the function within the service, the matching signature, and the tModel id.

Since we are using a full-text search to perform the matching, the search activity is reduced to matching the search string against the pre-compiled full-text index. As such, the retrieval is limited only by the capabilities of the full-text search engine.

The advantage of an exact match is that the set of returned signatures is very concise. The disadvantage is that the developer must have very specific knowledge about services that are being sought including knowledge of the exact types used by a service and the ordering of those types.

4.3. Transformation Match

Transformation Match, like its name suggests, transforms the search expression provided by the user into all of its possible permutations and performs search based on the permuations. For instance, given an expression "`String, Int32, String -> String`", transformation match takes permutations of the inputs "`String, Int32, String`" to perform the search. As such, functions with the following signatures would be returned as matches on the original search string:

```
Int32, String, String -> String
String, Int32, String -> String
String, String, Int32 -> String
```

The advantage of a transformation match is that it relaxes the constraint present in the exact match case. That is, the exact form of the desired signature is not required. However, there remains a restriction that the developer must have at least knowledge of the types involved in the desired operation.

4.4. Contains Match

The last form of match that we currently support is known as *Contains Match*. In this form of match, the returned signatures *contain* the types found in the search string. For instance, the search string `"string, string, boolean"` could possibly return the following signatures:

```
String, String-> Boolean
String, String, String,String -> Boolean
String, String, Boolean -> ResponseResult
Boolean, String -> String
```

Notice that this form of search is not constrained by input types and output types. As such, the types in the search string given above need only be contained within the resulting signatures. Furthermore, by including the "->" operator, the search can be qualified to indicate input and output relationships between the given types. The advantage of the contains match is that the developer need not be concerned with the input and output relations, but has the flexibility to include the relationships when needed. Additionally, the contains match allows for partial match with less stringent requirements being placed on the search criteria. Finally, since we are using a full-text search engine, the contains match can be qualified using many different criteria including the use of wildcards, location in the query string, and weighting of types. The disadvantage of the contains match is that in general, queries may return many more signatures than are actually required. As a result, the more refined a search query, the smaller the set of returned signatures. Conversely, the more general a search query, the larger the result set. Regardless of size, the user will ultimately be required to perform the final selection of a service since input and output relationships may not be known in a contains match.

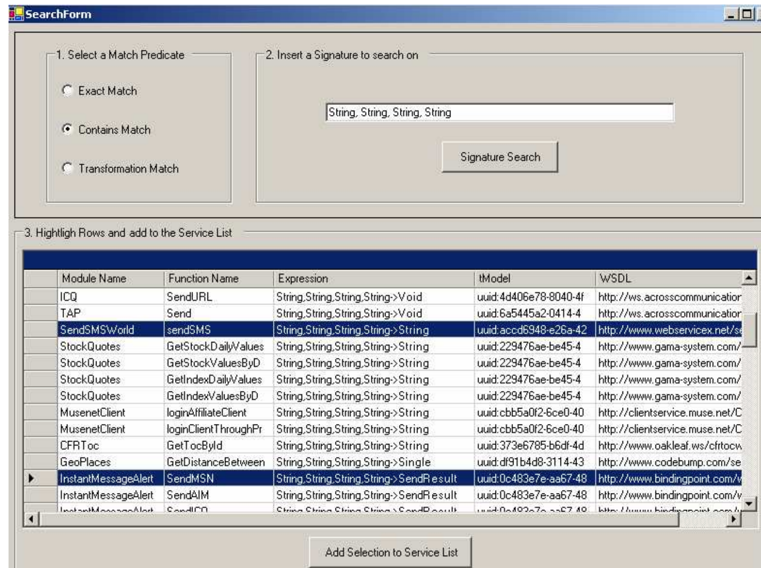

Figure 2. Messenger Signature Search

4.5. Other Match Criteria

Our current toolset supports each of the search methods described above. In addition, we are currently working on supporting the use of type-subtype relationships (e.g., generalization and specialization matches [1]) so that types contained in signatures can be replaced with either subtypes or supertypes.

5. Tool Support

To support the activities described in this section, we have developed a toolset upon the Microsoft .NET platform. The toolset supports automatic and manual population of a service meta-data and function repository, and supports exact, transformation, and contains signature matching. Figure 2 contains a screen capture of the results from performing a contains match on "`String, String, String, String`". The application provides a form for specifying signature match queries (see top portion of the interface), and a result area for displaying the returned matches. Services can be selected on the bottom portion and added to a list of services to be used by the client application.

6. Related Work

Many approaches for coping with library-based reuse issues have been suggested, including the signature-based reuse work described earlier [1], and semantic-based reuse [3], where component matches are achieved via the use of computationally expensive proofs that relate preconditions and postconditions of required and stored components. More recently, work has focused on the use of the semantic web [4]. The work described in this paper differs from the former in the use of full-text search as the mechanism for achieving signature match and differs from the latter two approaches in the use of lightweight match criteria.

7. Conclusions and Future Work

Service-oriented programming and development is increasingly gaining attention as a way for organizations to more readily achieve inter-organization and enterprise-level collaboration. A challenge to using services in an effective way is the identification and integration of available services with client applications. In this paper we described a search-based mechanism and toolset for facilitating retrieval of services. A primary feature of this approach is the use of full-text search [2] to achieve signature-based match.

References

[1] A. M. Zaremski and J. M. Wing. Signature Matching: a Tool for Using Software Libraries. *ACM Transactions on Software Engineering and Methodology*, April 1995.

[2] Purna Gathani, Sade Fashokun, and Rashid Jean-Baptiste. Full-Text Search Deployment. [Online] Available http://support.microsoft.com /default.aspx?scid= /support/sql /content/2000papers /fts_white%20paper.asp.

[3] Y. Chen and B. Cheng. A semantic foundation for specification matching. In M. Sitaraman and G. Leavens, editors, *Foundations of Component-Based Systems*. Cambridge University Press, 2000.

[4] K. Sycara, M. Paolucci, A Ankolekar, and N. Srinivasan. Automated Discovery, Interaction and Composition of Semantic Web Services. *Journal of Web Semantics*, 1(1), December 2003.