Enterprise Application Integration Using Extensible Web Services

Venkat N Gudivada
Department of Engg. & Computer Science
Marshall University
Huntington, WV 25755
gudivada@marshall.edu

Jagadeesh Nandigam
School of Computing & Information Systems
Grand Valley State University
Allendale, MI 49401
nandigaj@gvsu.edu

Abstract

This paper describes an approach to Enterprise Application Integration (EAI) using extensible Web services. The approach is demonstrated by building a real-world application for EAI in the financial services domain. Business drivers for and approaches to EAI are presented first. The manifestation of Web services in general and their role in EAI are discussed next. Financial services domain characteristics are presented. Business drivers that entail a strong need for functional extensibility in the financial services domain are described. Our proposed architecture for EAI which addresses functional extensibility is described. This architecture is based on the notion of extensible Web Services. We then present our implementation of the architecture and practical challenges encountered in EAI. A brief discussion of how our work relates to the current research in Service-Oriented Computing (SOC) and Semantic Web concludes the paper.

1 Enterprise Application Integration

Typically large enterprises are supported by hundreds of applications. Many of these applications were written in COBOL on mainframe computers and are referred to as *legacy systems*. Enterprises critically depend on legacy systems for their day-to-day business operations. It is not unusual for large brokerage firms in the financial services sector to appropriate annual legacy maintenance budget in the order of billions of dollars.

Therefore, it is natural for the enterprises to explore ways to reduce legacy maintenance costs. Two primary approaches were pursued. The first approach involves *replacing legacy systems* with a new application. The latter is designed for better interoperability with other systems, and more importantly easier to maintain and evolve. Enterprise Resource Planning (ERP) systems were introduced to address this need. Though ERP systems were well received initially, enterprises are increasingly turning away from them due to exorbitant total cost of ownership (TCO) — licensing, implementation, and maintenance costs.

In the second approach, legacy system replacement was attempted by developing new in-house systems. This approach also met with little success. This is where Enterprise Application Integration (EAI) comes into play. Instead of replacing legacy systems, EAI leverages legacy system functionality in meeting the new requirements.

During the last few years, there has been intense thrust on leveraging legacy applications — endowing them with interoperability features for the Web environment. Recent interest in Semantic Web [3] and Service-Oriented Computing [6] has accelerated the thrust on EAI.

In this paper we describe an approach to EAI using extensible Web services. Section 2 presents approaches to EAI. Manifestation of Web services in general and their role in EAI are discussed in section 3. Financial services domain characteristics are presented in section 4 from the perspective of functional extensibility; and section 5 discusses the notion and types of extensibility. An architecture for EAI using extensible Web services and the implementation of the architecture are described in section 6. The next section concludes the paper by relating the work presented here with the current research in Service-Oriented Computing (SOC) and Semantic Web.

2 Approaches to EAI

There are two facets to EAI: *integration type* and *what is being integrated*. There are two types of integration: internal and external. *Internal integration* primarily aims at establishing interoperability among the internal systems of an enterprise, whereas *external integration* encompasses interoperability with the systems outside the enterprise as well — partner, vendor, counterparty, and customer systems. Internal integration is relatively easier in that enterprise infrastructure and other technology standards, some homogeneity in applications, and more secure operating environment

Proceedings of the IEEE International Conference on Web Services (ICWS'05)
0-7695-2409-5/05 $20.00 **IEEE**

come to the rescue.

The things that are being integrated may include just the information, business processes [4], or both. Information integration focuses on providing an enterprise-wide integrated view of data entities to promote data integrity, shared semantics and consistent usage, and rapid application development. Business process integration encompasses coordinating and executing a sequence of steps to accomplish a business task; each step may need to be executed by a different application.

Approaches to EAI range a broad spectrum: from enterprise data integration (EDI), inter-application communication, to generic shared services. EDI is better viewed as an infrastructure to support inter-application communication and generic shared services; to provide a unified and consistent view of critical data entities, and provide information connectivity across multiple platforms. Inter-application communication is often designed as point-to-point application interfaces; entails significant maintenance costs especially for large organizations. Generic and shared services approach aims at achieving EAI via an array of carefully designed and loosely coupled resilient components; tries to minimize point-to-point application interfaces. Our approach to EAI presented in this paper is based on providing generic and shared services.

Approaches to internal integration include data warehousing; virtual data warehousing; remote procedure calls (RPCs), including SOAP-based ones; method signatures — Component Object Model (COM), CORBA, and Java Remote Method Invocation (RMI) calls; XML documents; Message-Oriented Middleware (MOM) messages.

The data warehousing approach integrates disparate data sources by developing a global schema and providing a consistent and unified API to the global schema. Data warehouses tend to be large in size, expensive to develop, and entail greater risk since the development cycle tends to be longer. Currency of data is an issue since the warehouse is refreshed only once or a few times a day for performance reasons. Furthermore, data warehouses typically don't support transactions on the global schema since uniquely mapping the transaction data to the underlying data sources as a distributed transaction is not possible in the general case. For this reason, data warehouses are typically used in read-only mode to support decision support applications. We refer to this approach as *physical data warehousing*.

Virtual data warehousing was introduced to alleviate some of the problems associated with physical data warehousing — warehouse size and data currency. Virtual data warehouse also employs a global schema, but it is not populated. Applications specify data access requests as queries on the global schema. Like the physical data warehousing approach, the virtual approach is also limited in its capability to perform transactions on the global schema.

Remote Procedure Call (RPC) enables an application to invoke a function in another application. The applications typically run on two different computers. The level of abstraction manifested in RPC is low — the programmer need to explicitly address issues related to the differences in the representation of data and network communication protocols. Common Object Request Broker Architecture (CORBA), an RPC, is Object Management Group (OMG) standard for application interoperability and integration. Because of its heavy foot-print, steep learning curve, and lack of a productive development environment, industry didn't rally behind CORBA.

Component Object Model (COM)/Distributed Component Object Model (DCOM) was Microsoft's foray into component-based application development and RPC-based distributed computing. Like CORBA, COM/DCOM was also exceedingly complex and there was no provision for component versioning in deployment. COM/DCOM is limited to Windows operating system only. Java Remote Method Invocation (RMI) was introduced by Sun Microsystems for RPC-based interoperability between applications developed using the Java programming language. Though there is no platform dependence in terms of hardware and operating systems, there is the Java langauge dependence.

XML documents have become a preferred format for data representation in SOAP-based RPCs. MOM-based application integration was championed by IBM via its Message Queue (MQ) product line. Applications communicate by sending and receiving messages. An application requesting the services of another application explicitly encodes such requests as messages. One or more queues are associated with the applications and the latter monitor the queues for service requests and responses. Of all the approaches mentioned, MQ based approach has been successful since the messaging software is available for virtually any platform.

Technologies and tools for external as well as integration include message-oriented middleware (MOM); extraction, transformation and loading (ETL) tools; business process management (BPM) tools; integration broker suites — Tibco Softwares ActiveEnterprise, IBMs WebSphere MQI, SeeBeyonds e*gate, Microsofts BizTalk, WebMethods Enterprise, and Vitria Technologys BusinessWare; and Web services.

3 Web Services and EAI

A Web Service is an interoperable unit of application logic that transcends programming language, operating system, network communication protocol, and data representation dependencies and issues. It is an infrastructure for developing and deploying distributed applications. Web Services are typically intended for applications consumption.

This is in contrast with contemporary Web applications, which are meant for human users.

Web Services are based on the following industry standards: eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), Web Services Description Language (WSDL), and Universal Description, Discovery, and Integration (UDDI). XML is to data representation what HTML is to rendering of Web documents. SOAP is an XML-based message format for exchanging information between computers. WSDL is used to describe a Web service, specify its location, and describe the operations it exposes. WSDL-based document provides enough information about how to interact with the target Web service. UDDI Registry is a collection of information on all the registered Web Services. It is a free public registry — vendors publish their Web services and consumers search for appropriate Web Services.

Web Services manifest in the following ways: internal and external application integration, component-based software development [7], and software as a service. Internal integration is effected by developing SOAP interfaces to internal legacy applications. Issues here include developing a domain ontology, specification of core services, enhancing a service, and composing a new service in terms of existing services. Applications can then communicate with each other by exchanging SOAP messages using HTTP over intranets. External integration is quite similar except that security is an additional issue that need to be addressed.

Though the idea of component-based software is not new, it was dogged by problems that arose due to non-transparency of programming language, operating system, data representation, and network communication protocol related idiosyncracies and dependencies. If the components are available as Web Services, the interoperability issues will go away to a large extent. This ushers in a new paradigm for software development — composing applications by configuring and integrating truly interoperable components.

A few years ago, there was a lot of exuberance about Application Service Provider (ASP) model. Under this model, organizations can rent applications hosted by the ASP vendor without worrying about hardware purchase, software licensing and upgrades, and keeping the applications up and running round the clock. For various reasons the ASP model has died prematurely. It is expected that Web Services will bring renaissance to the ASP model. This observation is based on the fact that industry convergence on standards is crucial to the economic viability of the ASP model.

4 Application Domain Characteristics

Financial Services industry is one of the domains used by Information Technology (IT) companies as a proving ground to demonstrate that their applications are of enterprise class. It is often said that if a product withstands the vigor of the Wall Street, it can succeed elsewhere easily. The domain is characterized by massive data volumes, narrow processing windows, and stringent performance and availability requirements. Legacy systems are the norm rather than an exception. The data elements in the domain are interrelated in complex ways and so do the business processes. The situation is further exacerbated by frequent changes in regulatory and compliance requirements besides the need for functional enhancements.

The type of data maintained about financial instruments include time series (e.g., price history), real-time price quotes, fundamental or indicative data (e.g., yield to maturity, coupon rates and payment frequency), research (e.g., quality ratings, future outlook), and news. Data is also maintained about various market participants and their roles including broker/dealers, stock exchanges, Over the Counter (OTC) markets, issuers of financial instruments and their ratings, clearing houses, custodians, transfer and paying agents, depositories, and regulatory and compliance organizations. The disparate IT systems used by the market participants come into play in the life cycle of a trade and post-trade operations — order entry and routing, execution, confirmation, clearing and settlement, corporate action processing, custody and portfolio management.

Currently, the life cycle of a trade is three business days after the trade has been executed (referred to as $T+3$). The longer it takes for a trade to settle, the higher is the risk for the market participants. For this reason, there is a mandate from the Security Industry Association (SIA) that a trade life cycle be reduced to one day after the trade (i.e., $T+1$) by the year 2007. During the course of the life cycle, trade information flows through several disparate systems, both in-house and external. A large portion of the cost of a trade is attributed to the manual processes involved in feeding data from one system to another. An industry-wide initiative — referred to as Straight Through Processing (STP) — has been in place for several years to automate these manual processes to reduce the number of trade failures as well as to realize the $T+1$ initiative. However, progress has been slow even for internal integration. Thus, Web Services technology has tremendous potential in achieving STP by integrating in-house and external systems.

Given frequent changes to business processes and the need for incorporation of regulatory and compliance requirements, it is rather mandatory for IT systems in financial services industry to feature *functional extensibility*. The latter is an architectural mechanism to extend the functional

capabilities of a system with minimal or no changes to the code. Functional extensibility is also needed to address the following: multi-entity processing, user-specifiable accounting options, and account service options, among others. Multi-entity processing enables transactions of the same kind be processed differently to reflect the business processes of the organization that originated the transaction. For example, all Visa credit card transactions are uniformly processed by the same system, yet the processing reflects the terms and conditions of the account related to the transaction; and accounts are issued by various financial institutions. In other words, functional extensibility is a means for *mass customization* of information systems.

User-specifiable accounting options include multi-currency or single-currency, principal and income separation, choice of amortization and accretion methods, lot accounting, carrying and cost value types desired, and lot disposition methods. Account service options comprise chosen risk management strategy, owner and regulatory compliance, discretionary and non-discretionary portfolio management, performance measurement and attribution, account valuation methods, and period-end statements. In summary, IT systems for financial services industry needs to be highly extensible and configurable to address the above functional requirements.

5 Notions of Extensibility

The genesis of extensibility traces back to the programming languages of the 1970s. User-defined data types and abstract data types were introduced as an extensible mechanism to extend the intrinsic data types of the language. The extensibility notion also appeared in relational database management systems in the area of optimizing queries that involve user-defined data types. ERP systems architecture is centered around functional extensibility via configuring the application modules to reflect an organization's business processes. Of late, extensibility is also investigated in the area of operating systems — dynamically configuring kernel services to suit application-specific characteristics performance-wise [9].

There are two types of extensibility: dynamic and static. In *dynamically extensible systems*, code components can be added to a running system in an arbitrary way. Web browser is an example of a dynamically extensible application — can load and execute plug-ins and applets on the fly. Security is of paramount concern is such systems. *Statically extensible systems*, in contrast, achieve extensibility by employing an array of techniques ranging from configuring initialization parameters to developing new code.

Functional extensibility is viewed as a continuum. At one end of this continuum is a technique referred to as *white labeling* or *personalization*. Internationalization issues such as language and locale come under white labeling. This is the easiest to achieve via resource files as exemplified in numerous Windows applications. The next one in the continuum is based on *user modeling* and *profiling*. Using this technique, an application's behavior in certain functional areas is dynamically governed by cumulative accumulation and adjustment of past user actions and preferences. These aspects are best exemplified in amazon.com web site's book recommendations feature. The next in the continuum pertains to functional extensibility via code changes. Applications development by making use of class libraries and design patterns makes this approach a little more palatable. True component-based application development achieves functional extensibility primarily by adding and configuring new components. It is essential that such components be interoperable, configurable, and extensible. ERP applications achieve *pre-defined* functional extensibility via rules-driven processing governed by business process-specific meta-data. Extensibility by integrating product-lines and domain-specific languages is presented in [2].

At the other end of the continuum are approaches that enable functional extensibility via a declarative specification framework. More specifically, our approach to EAI and functional extensibility is achieved via the notion of *extensible Web services*, which is described in the next section. Since functional accuracy is the essence of information systems in the financial services domain, it is necessary to employ static extensibility.

6 An Architecture for EAI

Our approach to EAI and functional extensibility is realized by using an architecture named EWSA (Extensible Web Services Architecture). First we describe the operational context for EWSA. The latter needs to integrates several legacy applications. Each application sports its proprietary API. However, all applications use the same relational data model. The data model has in excess of one thousand tables and some of the tables have over 200 columns. The data model is comprehensive for the domain, but it is complex. The tables are highly interrelated (via foreign key relationships) — beneficial for data integrity but counterproductive for transaction rates. We term this data model as Comprehensive Relational Data Model (CRDM). Ad hoc queries is a primary mode of retrieving information by end-users from CRDM since the information content requested in the queries keeps changing over a period of time. There is a steep learning curve for the users even to understand a subject area in CRDM. Updates to the CRDM database are always routed through the legacy systems, as the latter abstract business processes and update semantics. Prior to the implementation of EWSA, application developers need

to master the APIs of the legacy systems as well as the CRDM. Various internal departments in the enterprise develop applications, which leverage the legacy systems; and these applications operate in an Intranet environment.

Therefore, the objectives for EWSA are:

- Leverage legacy systems.
- Provide an integrated functional view of legacy systems and CRDM as domain-level, task-oriented, sharable atomic services.
- The services should be delivered via multiple delivery channels.
- Services should be configurable at multiple levels of granularity — service level, service group level, and system level.
- Services should be extensible.
- It should be possible to create a new service by modifying an existing service, or by declaratively composing the existing services.
- The integration architecture should promote platform independence and offer deployment options.
- It should be possible to programmatically use services with any industry standard programming languages.

The major tasks in realizing EWSA are: establishing a domain ontology; specifying and implementing core services; implementing the EWSA infrastructure; and developing a tool set for automating the various subtasks in the specification, implementation, and maintenance of services. These tasks are described in the following subsections.

6.1 Domain Ontology

Since our domain is complex and vast, establishing and consistently using a shared vocabulary across disparate applications is crucial to the integration effort. Such a vocabulary is referred to as *domain ontology*. Ontologies encompass more information than what is found in data dictionaries — data element name; semantic meaning; context-dependent usage; relationship to other elements; constraints and conditions (e.g., edit and validate checks); how the element value is derived (e.g., enumeration, executing a SQL query, a database stored procedure, or an external procedure invocation); and CRDM place holder (i.e., table/column name), if applicable. The domain ontology is central to EWSA. The elements of the ontology are referred to as Business Common Names (BCNs). The ontology is implemented as an Oracle database with suitable API. Industry standards or activities related to our ontology include ISO 15022 Data Field Dictionary, SWIFT Standards Financial Dictionary, and Market Data Definition Language (MDDL). However, the focus of these activities is narrower in scope.

6.2 Specification of Core Services

One of the difficult tasks in EWSA is identifying and specifying a set of core services. The services should be neither too primitive nor too coarse. There is no practical value if the services are too granular. If they are too coarse, their use in composing new services is greatly diminished. The situation here is somewhat akin to fundamental transformations in computer graphics, where any complex transformation can be expressed in terms of three primitive but fundamental transformations — translation, scaling, and rotation.

Given the complexity of the domain, it is rather unusual for any one business analyst to understand the domain in its entirety. A collective effort of business analysts and system developers was required for identifying meaningful set of core services. We have taken a role-based, task-oriented approach to the identification and characterization of core services. For example, the various domain tasks performed by a corporate action processing end-user provides enough information and context to come up with a set of services for this sub-domain.

Specification of a service encompasses identifying the BCNs needed for the task and associated business process that manipulates or uses these BCNs. Services are specified using the domain ontology in XML notation. The set of BCNs retrieved by a service are collectively referred to as an *information block*. Once the set of core services are identified for the various sub-domains, they were reconciled and consolidated. The effort involved in implementing a service depends on whether the service simply retrieves information from the CRDM or updates the information as well. In the latter case, the updates need to be processed through the legacy applications (more details on this later).

6.3 Extensible Web Services Architecture

EWSA is shown in Figure 1. EWSA services are delivered via three channels — Web service, message queues, and .NET assembly. Corresponding to each of the delivery channels is an adapter component residing in *ServiceProvider*. The latter functions as an entry point into EWSA. It intercepts all service requests coming through the delivery channels, and hands them over to *ProcessOrchestrator*, which has total responsibility for processing a service request. ProcessOrchestrator has the requisite knowledge to process all services hosted by the EWSA. Depending on the service request type, fulfilling the request may comprise several steps, and executing these steps in a specific order.

There are two categories of services: query and update. *Query* services just retrieve the data from CRDM. *Update*, on the other hand, alters the CRDM data via legacy and other applications. Consider the query-type service re-

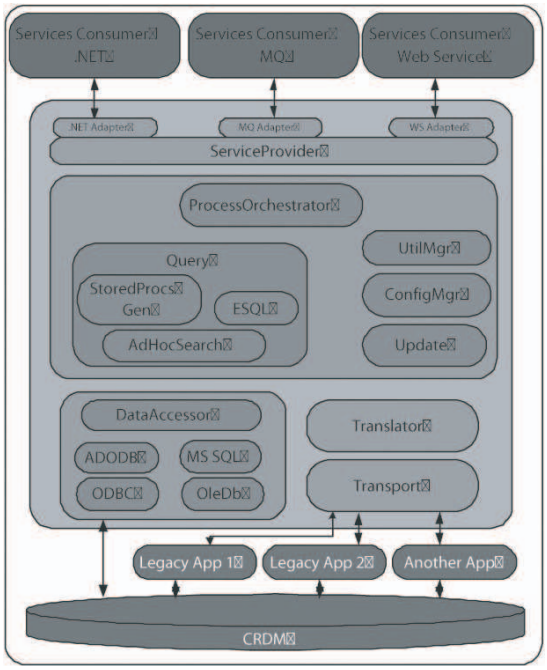

Figure 1. EWSA Architecture

quests. They are implemented by one of three ways: executing a pre-defined SQL query, executing a SQL query received as service parameter data — dynamic SQL, and executing a database stored procedure. The query service requestor can specify any of the following three formats for receiving the results: ADO record set, .NET dataset, and hierarchical XML. The last format is not just turning a tabular format data by inserting appropriate XML tags. It involves constructing a truly hierarchical XML document by discovering multi-level parent-child relationships in the data. This not only eliminates the redundancy in the data returned, but more importantly adds value by explicitly delineating hierarchical relationships among the data elements.

AdHocSearch component generates a SQL query on the fly for retrieving a given set of BCNs. Besides BCNs, AdHocSearch requests optionally specify filter conditions on the BCNs (e.g., MaturityDate $>$ Jan-10-2005). However, the service requester is relieved from specifying the CRDM table/column mappings for the BCNs and database table join conditions, which are automatically discovered by the AdHocSearch component.

The *DataAccessor* component provides a generic mechanism for executing SQL and database stored procedures against OLEDB, ADODB, and ODBC data sources besides Microsoft SQL Server.

Next consider the update category service requests. Since the updates are required to go through the legacy applications, they are more complex relative to query-type services. The *Update* component employs two other components: Translator and Transport. Recall that the APIs available for accessing the legacy applications are unique to them, and often are based on proprietary message formats. The APIs require that the input parameters be encoded using pre-specified message structures. The principal function of the *Translator* component is to generate a suitable message structure to invoke the relevant API in the target application. Although the overall message structure is pre-determined for a specific update service, the actual structure and length of an instantiated message depends on the parameter values supplied by the service invoker. Message structures are declaratively specified using meta-data. Thus, message structures for new services can be introduced without any changes to the source code of EWSA.

Once the actual message is generated, it is transported to the application using a network protocol that the application is compatible with. This is the core function of *Transport* component. For some applications, more than one choice is available (e.g., TCP/IP, IBM MQ). The transport protocol to be used is specified during the service configuration. The Update component is capable of communicating with applications in both synchronous and asynchronous modes. The mode is actually dictated by the application. Typically, asynchronous mode is used for communicating with message-based APIs; synchronous mode is used otherwise.

ConfigMgr component is used for configuring a service, a group of services, or all the services as a unit (i.e., system level). Configuration information includes database server type, connection string, database driver, maximum number of records to be returned, location of XML files for system initialization, trace flags, and specifying what information goes into a trace file when the trace is enabled. Trace feature is used in debugging services.

UtilMgr is a collection of utility classes for accessing and manipulating XML Schemas, and XML and trace files.

6.4 Enhancing an Existing Service

An existing service can be declaratively enhanced or customized in two ways. First, the data returned by a service can be configured to include only a subset of the BCNs associated with a service. By default, all the BCN values included in the service specification are returned. If the service requestor needs only a small subset of the BCNs, this modification helps to improve the response time. Services can also be declaratively enhanced by associating pre- and post-processing operations. The operations are implemented as .NET assemblies, and are executed before and after the service execution. These assemblies are externally developed by implementors desiring to customize an existing service. Pre-processing is typically used for special

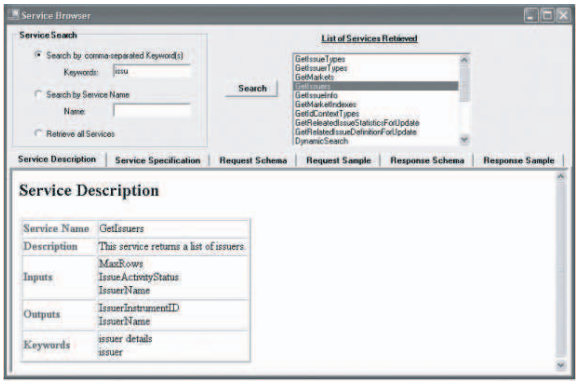

Figure 2. ServiceBrowser

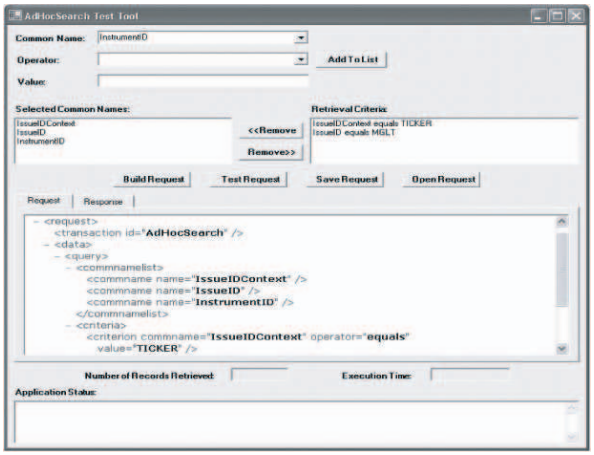

Figure 3. BCN Based Ad Hoc Database Search

transformations on XML encoded parameter data. Likewise, post-processing operations are used to transform the service results to suit local needs.

6.5 Composing a New Service

A new service is composed from existing services in two ways, which are referred to as static and dynamic composite services. Users declaratively specify *static composite service* in the form of a script (in XML). Such a script essentially lists various services to be executed and specifies how the results from executing one service are mapped as inputs to other services. This is a powerful feature to efficiently realize functionally higher-level services tailored to a specific purpose. *Dynamic composite service* is similar to its static counterpart, except that the composition script is not known until the run-time.

In both types of composition, it is possible to simply execute a list of services as a unit and return multiple result sets in one XML document. Under this scenario, results of executing a service are not mapped as input parameters to other services in the unit. The resulting XML document includes information to correlate service requests with their corresponding execution results. This also enables specifying input parameters that are common to multiple services only once.

6.6 Integrated Tool Set

As EWSA is meta-data driven, it features graphical user interface (GUI) tools for meta-data generation and maintenance. It also provides tools for service specification; browsing BCNs and services; service configuration; composing an ad hoc SQL query using the BCNs; and automated stored procedure generation.

ServiceBrowser is a tool for searching and browsing existing services and is depicted in Figure 2. Search can be based on keywords associated with the service or service name. It is also possible to browse through all the services in alphabetical order of service names. Clicking on any service name resulting from the search, yields the following information: service description (service name, a brief description, BCNs that are required as input parameters, BCNs that are returned in the result, and keywords associated with the service); service specification (an XML based specification of the services including its configuration information); request schema (an XML schema for the document, which is required as a parameter to invoke the service); request sample (an instantiation of request schema); response schema (an XML schema for the document, whose structure is used to return the execution results); and response sample (an example XML documents showing the returned results). All of the above pieces of information is meant for the consumers of the service, except the second one, which is of interest to the service implementors.

Tools are also available for searching and browsing the BCNs, configuring services, BCN based ad hoc search, database stored procedure generation, and testing of services. BCN based ad hoc database search tool is shown in Figure 3. It allows a user to select one or more BCNs for retrieval, and specify constraints on them — predicates on the selected or other BCNs. Using this information, it constructs an appropriate SQL query on the fly and generates service request XML document corresponding to the SQL query. It also enables the user to test the service request by invoking EWSA. The interface also allows for saving the ad hoc query requests and replaying them at a later time.

The tool for testing services is shown in Figure 4. The

Figure 4. ServiceTest

tool generates an XML request document for a chosen service, executes the service, and displays execution results.

7 Conclusions

The task of identifying core business abstractions or services at the right level of granularity turned out to be the most challenging. An incremental, role-based, task-oriented approach we embarked upon proved quite successful.

EWSA based EAI demonstrated significant productivity improvements for application developers. It effectively transcended programming language, operating system, and network communication related issues. The services provided a true business-oriented abstraction of the domain, thus freeing the developers to focus more on the system implementation. Two applications based on EWSA are in production.

Because of the data-intensive nature of the application, some services can take significant amount of time to execute. Under such situations, asynchronous mode of service execution seems desirable. Though asynchronous invocation of services is intrinsic to the IBM MQ based delivery channel, we haven't investigated issues that might arise in asynchronous invocation for the Web Services delivery channel. Another issue that remains to be investigated is incorporating state information into Web Services. Currently, all the services are stateless. Exploring the approaches used in the recently proposed Business Process Execution Language for Web Services (BPEL4WS) is a good starting point in this direction.

Another factor that would help with performance is caching service execution results in EWSA. Especially services that retrieve reference data are good candidates for caching. Reference data doesn't change frequently. Implementing a cache synchronization mechanism will be considered in a future release of EWSA. Currently, our approach to EAI requires considerable technical expertise for composing a new service. From a system developer perspective, though the approach entails significant productivity gains, our goal is to evolve the approach so that business analysts themselves should be able to compose services.

The work presented in this paper complements the recent research in Semantic Web, Service-Oriented Computing (SOC), and composing Web services [1, 5, 8]. In the Semantic Web environment, applications are dynamically weaved by composing loosely coupled shared services using domain ontologies. A primary aspect of SOC is in creating loosely coupled services, which publish their characteristics — both functional and non-functional — in a standardized, machine readable format.

References

[1] S. Arroyo, R. Lara, J. M. Gomez, D. Berka, Y. Ding, and D. Fensel. Semantic aspects of web services. In M. P. Singh, editor, *The Practical Handbook of Internet Computing*, pages 31–1 – 31–17. Chapman & Hall/CRC, 2005.

[2] D. Batory, C. Johnson, B. MacDonald, and D. von Heeder. Achieving extensibility through product-lines and domain-specific languages: A case study. *ACM Transactions on Software Engineering and Methodology*, 11(2):191–214, April 2002.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[4] F. Casati and A. Sahai. Business process: Concepts, systems, and protocols. In M. P. Singh, editor, *The Practical Handbook of Internet Computing*, pages 32–1 – 32–15. Chapman & Hall/CRC, 2005.

[5] F. Curbera, R. Khalaf, N. Mukhi, S. Tai, and S. Weerawarana. The next step in web services. *Commun. ACM*, 46(10):29–34, 2003.

[6] M. N. Huhns and M. P. Singh. Service-oriented computing: Key concepts and principles. *IEEE Internet Computing*, 9(1):75–81, 2005.

[7] R. B. Kieburtz, L. McKinney, J. M. Bell, J. Hook, A. Kotov, J. Lewis, D. P. Oliva, T. Sheard, I. Smith, and L. Walton. A software engineering experiment in software component generation. In *ICSE '96: Proceedings of the 18th international conference on Software engineering*, pages 542–552. IEEE Computer Society, 1996.

[8] E. Sirin, B. Parsia, and J. Hendler. Filtering and selecting semantic web services with interactive composition techniques. *IEEE Intelligent Systems*, 19(4):42–49, 2004.

[9] A. C. Veitch and N. C. Hutchinson. Kea – a dynamically extensible and configurable operating system kernel. In *ICCDS '96: Proceedings of the 3rd International Conference on Configurable Distributed Systems*, page 236. IEEE Computer Society, 1996.