Web Services with Java EE 6: An Example Using Planning in Reverse

Keith Ballantyne
University of Maryland Baltimore County, USA
kb12@umbc.edu

***Abstract*—This paper describes a Java-centric approach to the development of a service-oriented architecture. It introduces many of the technologies readily available within Java to realize a Web service architecture, and explains some of the difficulty encountered during the realization of specific Web services necessary to implement a strategic planning process. Preliminary results indicate that robust support of service-oriented architecture is available using Java technologies, but efficiently meeting non-functional requirements, such as security and platform-independence, pose design challenges for the developer.**

***Keywords*-Service-Oriented Architecture; J2EE; Web Services; Information as a Service; Planning In Reverse.**

I. Introduction

In the early part of 2011 this paper's author was approached by faculty members from Alvernia University for help in the construction of an application that would assist the users of a process they had developed. Concurrently, the author had enrolled in a graduate level service-oriented architecture (SOA) course intended to expose students to the utility of constructing applications based on services. The nature of the process defined by the Alvernia faculty lent itself well to a service-oriented approach. Ballantyne, et al. [1] outlines the need for a tool that actively incorporates all employees into the planning process.

> This book was written so that organizations – large and small, private and public, for profit and not for profit school systems or colleges and universities – can implement a process that includes all stakeholders and employees in a meaningful role in planning. It is designed so that all can be engaged in the process and become part of the organization's long-term viability by providing short-term observations.

After reading the book and conferring with its authors, several key features of the process became evident. Successful implementation relies on gathering information that is distributed among many stakeholders, resulting in a need for different types of client-side interactions. Both small and large organizations must be supported with equal utility, making scalability a key non-functional requirement. As an organization grows more adept at applying the process to its unique needs, the tool must change to accommodate the new knowledge. Finally, interaction must be ubiquitous, requiring easy access from a diverse collection of client hardware. Individually, each feature can be realized without giving much consideration to the architecture, but when taken as a whole, a service-oriented approach provides clear benefit. Adopting an *Information as a Service* approach [2] to maintaining data, and ensuring *loose coupling* between the client and business logic [3] provide both scalability and ubiquity in the final solution. A service-oriented approach also provides a path for future growth as the process matures within the organization, both through refinement and *service composition* [4].

The balance of this paper details the construction of the resulting system and some of the challenges faced along the way. It is organized as follows: Section 2 presents an overview of the Planning In Reverse as outlined by its authors, Section 3 provides an overview of Java Enterprise Edition 6 Web Services enabling technologies, Section 4 describes the current progress on implementation and pitfalls encountered, and finally conclusions and areas for future work are presented in Section 5.

II. Planning In Reverse

Scott Ballantyne, brother of this paper's author, along with Beth Berrett and Mary Ellen Wells published a book in 2011 titled *Planning in Reverse: A viable approach to organizational leadership* [1]. Citing many examples of failed strategic planning processes within organizations, the text outlines an alternative approach to strategic planning aimed at continuously updating an organization's strategic plan based on observation of current events. The process outlined in the book begins with an employee recording an observation of an event, and guides strategic planners in assessing the event, developing a plan to respond to the event, and implementing the developed plan. Planning In Reverse's authors were interested in developing software that could support the collection of information and assist strategic planners in keeping track of the process flow.

A. Process Description

At its core, PIR is a tool to facilitate the adaptation of long-term strategic plans to internal and external events that may occur. It is a methodology based on active participation of individual employees in the strategic planning process. The methodology described in the text includes a process of evaluation, implementation, and integration of the resulting strategy into the operational environment of an organization.

The process consists of several basic process steps.

- The initial step starts when an employee observes something they deem significant. News stories, newly available technologies, changes in world affairs, and even casual conversations that an employee feels could potentially impacting the organization are recorded as an *implication scan*.
- Once recorded, these *implication scans* are evaluated by a committee of strategic planners to determine their impact. Impact can be significant or insignificant, and may also be categorized as internal or external. Implications that are deemed to have a material impact are advanced to the next stage for further action.
- *Itemized action plans* are developed to address the implications. These itemized action plans define a sequence of steps for the organization to undertake to actively manage the impact. The plans include both steps to take and the logistics necessary to ensure the plan is successful.
- Finally, the plan is rolled out and integrated into the organization.

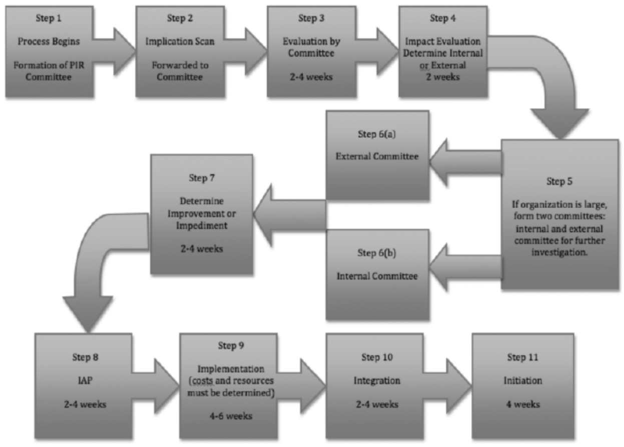

Figure 16.1. Timeline for implementation. PIR = planning in reverse.

Fig. 1. Planning in Reverse, [1]

.

Figure 1 shows a more detailed view of the process outlined within the text. Depending on the size organization, separate committees may be formed to better meet demand. Also note that implication scans can result in both positive (improvements) and negative (impediments) that must be dealt with.

B. PIR Example

To better understand the process flow consider the following example. An employee reads a news article that oil prices are likely to rise over $100 a barrel for the next 12 months. The employee submits an implication scan. The scan is entered into the system, and the next time the impact evaluation committee meets, they review it. They determine that higher energy prices will increase the overhead rates for their factory. These increased overhead rates will make the product they produce more expensive. The additional expense means their product will not be price competitive within the market. Realizing that they are not going to meet their annual operating profit, they develop an itemized action plan to increase the sales of an alternate product that will remain price competitive while reducing the energy usage throughout the factory. This itemized action plan is subsequently developed into an implementation plan by considering the logistics needs and defining the timeline. Finally, the implementation plan is integrated into the organization, and they are ultimately able to meet their annual operating targets.

III. Web Services

The term *web services* broadly refers to service-oriented architectures where services are provided over transport mechanisms such as HTTP, and service clients are available as webpages. As an architectural concept, SOA provides *loose coupling* [5] between the implementation of business logic and the consumers of those implementations. This loose coupling also allows significant capacity for scaling, either through the distribution of service endpoints across multiple platforms, or through the scaling of business logic implementation using multiple cores, distributed computation, or cloud computing. Prolific use of Web services is largely the result of standards that define the mechanisms of interface definition and service invocation.

A. Web Services Definition Language

Web Services Definition Language (WSDL) [6] is the primary mechanism used to define the interface to Web services. Written using the Extensible Markup Language (XML), WSDLs provide a complete encapsulation of the service interface available to the client. WSDL XML documents contain a complete description, including the methods available from the service and the data types that are exchanged. The key elements within a WSDL are defined below. The collection of information is a complete interface specification, allowing programmatic discovery and use of the service.

PortType	defines operations and their associated **Messages**.
Types	references an XML schema document used to define datatype elements and the namespace.
Message	defines a Web service method and **Parts** of the message.
Parts	identify parameters used in invoking the service method.
Binding	binds each **Operation** with its transport mechanism and datatypes.
Service	binds the service name to the port and the physical location at which the service may be found.

B. Simple Object Access Protocol

Simple Object Access Protocol[1] [7] (SOAP) is an XML specification that defines the structure of information passed to and from Web service *endpoints*. Specified by the W3C organization, the protocol is both prolific and well understood.

[1]More recent versions of the specification have dropped the phrase, referring only to the acryonym SOAP

SOAP messages provide client-initiated bidirectional communication with a service. A request is sent from the client to the Web service endpoint, and a SOAP response containing the information is returned from the Web service endpoint to the client. Whereas WSDLs specify the interface to the service, SOAP is used to transmit and receive service data.

C. Java support of Web Services

Traditionally, Web services have been developed using a manual process. WSDLs were constructed by hand, requiring a manual specification of each service endpoint and each operation. During development, the WSDL was manually updated each time a change to an operation was made. Since the resulting WSDL file is machine readable, a logical progression was the development of tooling that generated these files automatically and contained a mechanism to ease deployment. Java provides a robust implementation of all of the necessary components for Web service development.

D. Java 2 Enterprise Edition 6

The Java 2 Enterprise Edition 6 (J2EE 6) [8] specification for enterprise-level Java ensures that the requisite facilities for Web service development and deployment exist within the platform. A comprehensive tutorial is available from the Oracle website [9]. This tutorial includes information on nearly every enterprise edition feature, including Web service development. Enterprise edition 6 is more than a simple collection of compiler tools. This standard specifies services, libraries, and many other facilities that are useful for enterprise-level service deployment. J2EE 6 is not a specific implementation, but rather a specification that certified implementations comply with. Until recently, compliant implementations were only available commercially. However, with the release of GlassFish Version 3 [10], an open source J2EE 6 compliant implementation became available.

Java enterprise edition Web services are defined in JSR-109 [11]. This specification includes detailed information about developing and consuming Web services within Java code. The specification details how WSDLs and the services themselves are accessed. A related specification, JSR-172 [12], details the same thing for J2ME devices.

1) GlassFish: The GlassFish server bundles all of the necessary elements to deploy Java enterprise applications, including Java Web services applications. GlassFish provides administration utilities, application containers, extensive scalability, database management and connection facilities, and centralized user management. As a framework for deploying large-scale enterprise applications, GlassFish contains all of the necessary components in a single installation package. The server itself may be deployed on both Windows and Linux platforms, and is agnostic to the specific version or architecture of the host operating system.

E. Java Architecture for XML Binding

Consumption of Web services in Java relies on the Java Architecture for XML Binding (JAXB) client programming model [13]. This model provides a realization for transmission and reception of both Java Web service data and Java remote procedure call invocation. Though Web services under Java can be produced and consumed using Java remote procedure calls, the implementation within PIR relies solely on the JAXB Web services (JAX-WS) implementation.

JAXB exposes several core functions that are necessary in any Web services realization using Java. First, it provides annotation-driven binding that directly maps XML elements into Java objects. This binding is extensible, allowing implementers to alter both marshaling and un-marshaling of data via callback methods. Second, JAXB provides intrinsic Java type-to-XML mapping, combined with facilities to extend the mapping as necessary. Finally, JAXB provides facilities that allow customization of the XML schema to Java representation.

1) JAX-WS: As noted above, PIR relies solely on the JAX-WS implementation specified in JSR-224 [14]. This implementation includes both client and service facilities to produce and consume Web services. Typical usage involves annotating code using a special syntax to define the Web service, operations within the Web service, and the datatypes used in connection with the web service. A code snippet showing a small portion of the annotation is included below.

```
@WebMethod(operationName = "getLateImplications",
       action="getLateImplications")
public List<Implications> getLateImplications(
       @WebParam(name = "userID") int userID,
       @WebParam(name = "daysLate") int daysLate) {
```

Note the **@WebMethod** and **@WebParam** annotations above. These provide all of the information necessary for the JAX-WS architecture to properly generate the operations associated with the Web service. The annotations significantly reduce the amount of effort required to define the WSDL and SOAP interfaces for the service. Instead, the developer is free to focus on implementing the appropriate logic required within the service. At compile time, the annotations are parsed, and proper WSDL, XSD, and SOAP envelopes are automatically constructed.

Consuming a Web service is even easier. In either the Eclipse [15] or NetBeans [16] integrated development environments, the Web service can be selected from the tree hierarchy on the left pane and simply dropped into a Java class file. In both cases the result of the operation is an automatically generated stub that the developer can call directly. Both integrated development environments are sophisticated enough that a change to the service (WSDL) will be automatically reflected when the IDE is updated. The developer need only delete the prior stub and repeat the drag-and-drop operation. Unlike earlier manual Web service development, modern tools provide both simplicity and expediency when constructing service-oriented architectures.

F. Java Persistance Application Programming Interface

Among its many features, the J2EE environment provides several methods of connecting to a database. GlassFish is equipped with Java Database Connectivity (JDBC) connectors to many of the most popular databases. Even when the connector is not present in the base installation (such as the MySQL Connector/J [17]), installation is simply a matter of copying a Java archive file (JAR) into the appropriate directory. Once a JDBC driver is installed, the GlassFish server provides several mechanisms of allowing applications within it to access the database.

Though any application within GlassFish is capable of direct connection to a database, the most popular mechanism is the use of a connection pool. Connection pools allow any application that shares the same data source to use a commonly specified connection. An application requiring access to the data is pointed at the connection pool rather than a specific JDBC connection. In this way, server administrators can update the entire installation to a new database by simply updating the server's connection pool settings. No change is required to the service code.

JDBC connections allow direct queries of relational databases. Though this is frequently sufficient for data manipulation, it results in stronger coupling between the underlying database query engine and the application. J2EE includes other mechanisms to persist data within the database. The Java persistence Application Programming Interface (JPA) [18] is used throughout the PIR service implementation when possible. In addition to providing the functions typically necessary when manipulating a relational database, JPA represents the database rows as Java objects (called *entities*). Both Eclipse and NetBeans allow automatic construction of these entity objects by directly querying the database. The automatically generated models include many-to-one and one-to-many object relationships. Hence the object representation within Java is identical to the relationship within the database. This enables a cohesive, object-centric approach to data management within a Java enterprise application.

G. Security

One of the most difficult aspects of Web service development for PIR relates to securing the Web service. In 2004 the Web Services Security (WS-Security) specification was produced by OASIS [19]. Subsequently updated in 2006, the document details how SOAP messages may be secured. Most Web services are still secured by using the security mechanism built into the underlying web server. *HTTP basic auth* is frequently used as the identity mechanism. With or without secure socket layer (SSL) support, this basic mechanism presents several problems for the realization of PIR. Most notably, authentication to any service operation is cumbersome to an operator. Without some form of persistence, each SOAP request would require reentry of the username and password. JAXB provides mechanisms that allow the client to specify both username and password, but it still leaves several security weaknesses in Web service deployment. The WS-Security specification addresses these issues by standardizing several newer mechanisms of security within Web services.

Within the PIR application it is important that any call to the database via a Web service is secure. This means that the data transmitted and received via the Internet must be secure from a man-in-the-middle attack. The WS-Security specification enables both X.509 certificates and Kerberos tickets to be used for this purpose. When using security mechanisms such as these, the entire body of the SOAP message is encrypted using the ticket then decrypted by the client. Thus the certificate mechanism provides acceptable security for the protection of corporate strategic planning data.

IV. Implementation

In order for the PIR process to be successfully implemented, two critical objectives must be met. First, it must be easy for an employee to submit an implication scan. The PIR authors desire a ubiquitous application, with broad availability to every member of an organization. The intent is that an employee, regardless of their location, can readily submit a scan into the system. The second objective is that both the employee and the organization's committees are informed of the information and given frequent feedback about progress. PIR requires accountability for ensuring that every implication scan is processed in a timely manner. The process authors want every participant to be aware that their contribution is being considered and acted upon, and that submitters know the current status of their implication scans. These requirements, along with others from alternate stakeholder perspectives, are enumerated below.

A. Threshold Requirements

THR(1) The service(s) shall provide individual contributors with rapid and ubiquitous capability to submit implication scans.

THR(2) The service(s) shall provide individual contributors with routine updates on the progress of their implication scans throughout the process.

THR(3) The service(s) shall provide impact evaluators with the facilities necessary to manage implication scan evaluation.

THR(4) The service(s) shall provide impact evaluators deadline information for implication scan evaluation per process temporal constraints.

THR(5) The service(s) shall provide the capability to develop itemized action plans (IAP).

THR(6) The service(s) shall provide the capability to develop implementation plans.

THR(7) The service(s) shall provide the capability to develop initiation plans.

THR(8) The service(s) shall provide the capability to retrieve and view the history of items from implication scan through initiation.

THR(9) The service(s) shall provide the capability to tailor the process to fit organizational needs.

B. Objective Requirements

In addition to the threshold requirements, the author identified a few objective requirements that the architecture should support, even if the actual requirement was not supported in the first-pass implementation. These requirements follow in the list below.

OBJ(10) The service(s) shall provide search and retrieval capability of all historical data.
OBJ(11) The service(s) shall provide interfaces on portable devices.
OBJ(12) The service(s) shall provide Implication Scan submitters with the ability to interact with evaluators.
OBJ(13) The service(s) shall enable future capability growth.

C. Architecture

The requirements above specify an abstracted view of the desired application. They touch on the need for ubiquity and rapid feedback to stakeholders throughout the process. Service-oriented architecture provides a convenient method of enabling such ubiquity. Since the data repository itself can be realized as a service [2], client implementations are free to submit and retrieve data independently of the specific platform. Furthermore, Java standard edition applications, Web applications, and Java micro edition applications can be developed using the same service model, enabling the desired ubiquity. Once these applications are constructed, additional logic can be included in the service layer to monitor progress and enable rapid feedback. Finally, by moving the business logic into the services instead of the application, improvement and automation are possible without requiring client application upgrades.

The basic enabling technologies outlined above provide significant capabilities to a service-oriented application developer. As a framework, these technologies expose a broad collection of components that improve efficiency in adopting a service-oriented approach to an application. However, like all frameworks, their use requires design tradeoffs in the actual realization of the application. What serves well as a broad abstraction may limit what can be accomplished.

The PIR service-oriented architecture is shown in Figure 2. This architecture provides the capability to meet all of the threshold and objective requirements outlined above. The **PIR Services** process in the center represents the core collection of services. Services such as scan submission, current status, and backlog are realized through this interface. All consumers of PIR services are directed through the PIR Services endpoint. This includes both hand-held and desktop web services clients. Users accessing the service through a web browser also consume the PIR Services endpoint, but gain access through a server-side application (noted as the **PIR Web** process) that processes HTML requests.

The architecture accommodates a broad array of clients, from strategic planners at their desktop to individual contributors using a handheld[2]. Pairing of client and device places unique constraints on the realization of functionality. For example, a key difference between an enterprise edition Web service client (JSR-109) and a micro edition client (JSR-172

[2]Research In Motion's Blackberry Bold was chosen as a target handheld.

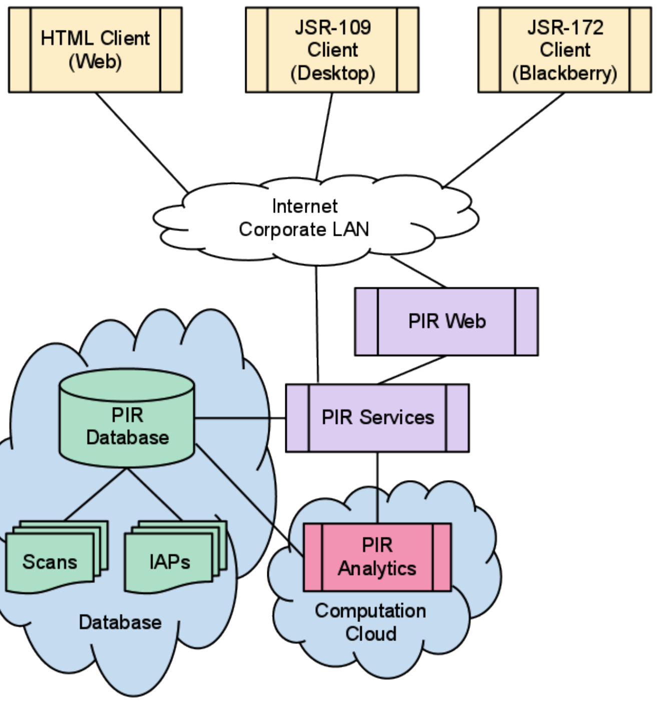

Fig. 2. PIR Architecture.

[20]) is that serialization and deserialization of objects via SOAP is limited under JSR-172. Hence, construction of the handheld PIR client places constraints on the services themselves. Where JSR-109 provides robust data type serialization, careful attention is required to ensure the service exposed for use by a Blackberry (JSR-172) can safely communicate with the device. The effect is handled in one of two ways: either objects are carefully constructed to use atomic data types that are supported by both standards; or custom logic is written to decompose complex objects into atomic types and reconstruct them after transmission. The PIR services require a combination of both, depending on whether the object can be cleanly represented as atomic types. As an example, textual elements within the database can be used as-is, but date-time stamps require special treatment.

D. Database Considerations

The PIR services themselves are straightforward implementations in Java. Since the services are predominantly about exposing basic database functions, most service implementation code is trivial. JDBC provides an abstraction for many database backends and JPA provides an object-centric view of the database. When combined, basic storage and retrieval operations are reduced to a few lines of code. As an example, code for a prototype scan implementation is included below.

```
@WebMethod(operationName = "submitScan",
     action="submitScan")
public int submitScan(@WebParam(name = "trigger")
String trigger, @WebParam(name = "description")
String description) {
 EntityManagerFactory emf =
 Persistence.createEntityManagerFactory("PIR");
 EntityManager em = emf.createEntityManager();

 em.getTransaction().begin();

 People person = em.find(People.class, person);
 Implications impl = new Implications(0,
        new java.util.Date(), trigger, false);
 impl.setDescription(description);
 impl.setUserId(person);

 em.persist(impl);

 em.getTransaction().commit();

 em.close();
 emf.close();

 return impl.getId();
}
```

The JPA has abstracted away any notion of table retrieval or joins. Instead, table rows are object instances, and the relationships established between them are expressed naturally in the language. The API simplifies manipulations of the database, but there are some caveats to using it. Frequently tables within a database have a many-to-one or many-to-many relationships. Unchecked, these relationships have the potential to recursively enumerate large portions of the database in a single SOAP response. Frequently the solution is to identify when the persistence manager should ignore the relationship by annotating the relationship with the **@XmlTransient** tag. Doing so prevents the entity manager from enumerating the table rows specified by the relation.

JPA also raises a question about achieving the proper level of abstraction. The database backend, whether it is MySQL, Oracle, SQL Server, or some other implementation, is abstracted in Java using JDBC. JPA adds another layer above JDBC, providing an object-oriented view of the database and its relationships. One can argue that this additional abstraction is more of a hinderance to developers than a benefit [21]. Since SQL itself is intended to be an abstraction of the implementation, layers above the query language may actually obfuscate what is going on. Long term code maintainability by someone familiar with database manipulation may be better served by directly encoding the queries rather than relying on the JPA abstraction. Since JDBC already accommodates switching to a different backend with minimal impact, JPA's benefit may be minimal.

E. Security Considerations

Similar questions arise about security. While the WS-Security specification standardizes the mechanisms for securing web services messages, there is not a comprehensive web services security mechanism built into Java EE 6. Several standards have evolved that may be combined in support of a secure solution, but a comprehensive package for securing them does not exist. For example, certificate-based mechanisms incorporating timestamps provide the necessary strength, but exchange and management of certificates is difficult to administer [22]. The problem is compounded when developing a system where centralized identity management outside of the Java enterprise edition environment is essential. As of this writing, the author is still seeking an elegant solution to securing the web services that works well for both hand held applications and desktop environments.

F. Application Design

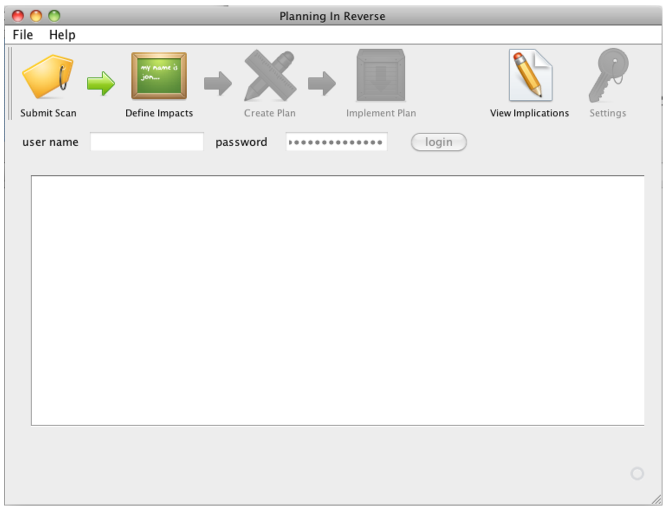

Fig. 3. PIR Desktop Client

.

Two primary client interfaces are required in the PIR implementation, one on a desktop the other on a handheld device. Though both rely on the same set of PIR services, their intended purposes are different. The handheld application is primarily used for implication scan submission and as a mechanism to track progress. As a result, the interface is quite simple, providing minimalist facilities to login, submit a scan (title and description), and check status. These three actions are collected in the application's menu.

The more complex application being developed allows strategic planners to manage the PIR process. The author elected to model the desktop-client user interaction as a process flow that follows the PIR workflow. Figure 3 shows the prototype desktop interface. The icons across the top represent steps in the PIR process. Like the handheld client, the desktop application uses the same core set of PIR services. Database access and business logic is abstracted by the services to provide a uniform interface.

G. Services

Below is a partial list of PIR functions decomposed into services. These services are currently incomplete, but the list provides insight into the basic functionality required. Perhaps the most significant benefit of an SOA implementation is the room for growth. As new functionality is identified that will improve the organization, it can be incorporated into the architecture with minimal impact.

submitScan: submit an Implication Scan for assessment.

getImplications: retrieve the contents of an Implication Scan.
submitImpact: submit an Impact Assessment for one or more Implication Scans.
getImpact: get an Impact Assessment.
getLateImplications: get Implication Scans that have aged too long.
closeImplication: close out an Implication Scan.
openImplication: reopen an Implication Scan.
closeImpact: close an Impact Assessment.
authenticate: authenticate a user for a session.

V. Conclusion and Future Work

The development of a Java-based Web services application provided significant opportunity to understand the facilities available under Java enterprise edition 6, and the progress made in facilitating service-oriented architecture development. It is clear that advances in technology, both from within the platform, and external to it in the development environment, have simplified the complexity associated with service-oriented architectures. Despite these advances, however, development of service-oriented architectures in Java still present some challenges. Security issues, platform compatibility, and API abstractions require analysis beyond simple design choices. Though the initial PIR application will remain in development for several more months prior to commercial use, this survey provided background in all of the technologies necessary to develop and deploy the final application.

The development and publication of new Web services provides ample opportunity for future expansion of the PIR application. Though the initial goal was simply to develop an information service that facilitated the use of the PIR process, future development efforts are likely to incorporate capabilities that become available as services. Both high-performance and cloud computing environments provide processing power necessary to perform complex artificial intelligence operations that may assist businesses in improving their strategic plans. The **PIR Analytics** box in Figure 2 encapsulates this expansion. Though these services may be undefined right now, it is clear that as computational capability increases, so too will its application to computationally difficult problems. In the future, strategic planning problems may be reduced to a specification that is transmitted to a service, processed, and returned back to PIR users. The elegance of a service-oriented architecture is that it is well positioned to embrace these potential capabilities without the need for substantial rework.

References

[1] S. Ballantyne, B. Berret, and M. E. Wells, *Planning In Reverse: A Viable Approach to Organizational Leadership*. New York: Rowman and Littlefield Education, 2011.

[2] A. Dan, R. Johnson, and A. Arsanjani, "Information as a service: Modeling and realization," in *Systems Development in SOA Environments, 2007.*, May 2007, p. 2.

[3] D. Kafzig, K. Banke, and D. Slama, *Enterprise SOA: service-oriented architecture best practices*. Upper Saddle River, NJ: Pearson Education, 2005.

[4] S. Staab, W. van der Aalst, V. Benjamins, A. Sheth, J. Miller, C. Bussler, A. Maedche, D. Fesnel, and D. Gannon, "Web services: been there, done that?" *Intelligent Systems, IEEE*, vol. 18, no. 1, pp. 72–85, Jan-Feb 2003.

[5] T. Erl, "Service loose coupling," 2009. [Online]. Available: http://www.soaprinciples.com/service_loose_coupling.php [Accessed: September 10, 2011]

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana, "Web services description language (wsdl) 1.1," March 2001. [Online]. Available: http://www.w3.org/TR/wsdl [Accessed: September 10, 2011]

[7] M. Gudgin, M. Hadley, N. Mendelsohn, J.-J. Moreau, H. F. Nielsen, A. Karmarkar, and Y. Lafon, "Soap version 1.2 part 1: Messaging framework," April 2007. [Online]. Available: http://www.w3.org/TR/soap12-part1/ [Accessed: September 10, 2011]

[8] Oracle, "Your first cup: An introduction to the java ee platform," March 2011. [Online]. Available: http://download.oracle.com/javaee/6/firstcup/doc/ [Accessed: September 10, 2011]

[9] E. Jendrock, I. Evans, D. Gollapudi, K. Haase, and C. Srivathsa, "The java ee 6 tutorial," March 2011. [Online]. Available: http://download.oracle.com/javaee/6/tutorial/doc/ [Accessed: September 10, 2011]

[10] Oracle, "Glassfish server open source edition 3.1 application development guide," April 2011. [Online]. Available: http://download.java.net/glassfish/3.1/release/glassfish-ose-3.1-docs-pdf.zip [Accessed: September 10, 2011]

[11] J. Kotamraju, "Web services for java ee, version 1.3," December 2009. [Online]. Available: http://download.oracle.com/otn-pub/jcp/websvcs-1.3-mrel2-evaluate-oth-JSpec/websvcs-1_3-final-spec.pdf [Accessed: September 10, 2011]

[12] J. Ellis and M. Young, "J2me web services 1.0," October 2003. [Online]. Available: http://download.oracle.com/otn-pub/jcp/j2me_web_services-1_0-fr-oth-JSpec/j2me_web_services-1_0-fr-spec.pdf [Accessed: September 10, 2011]

[13] K. Kawaguchi, S. Vajjhala, and J. Fialli, "The java architecture for xml binding (jaxb)," December 2010. [Online]. Available: http://download.oracle.com/otn-pub/jcp/jaxb-2.2-mrel2a-oth-JSpec/jaxb-2_2-mrel2-spec1.zip [Accessed: September 10, 2011]

[14] J. Kotamraju, "The java api for xml-based web services (jax-ws) 2.2," December 2009. [Online]. Available: http://download.oracle.com/otn-pub/jcp/jaxws-2.2-mrel3-evalu-oth-JSpec/jaxws-2_2-mrel3-spec.pdf [Accessed: September 10, 2011]

[15] D. Williams, "Eclipse web tools platform project," 2011. [Online]. Available: http://www.eclipse.org/projects/project_summary.php?projectid=webtools [Accessed: September 10, 2011]

[16] Oracle, "Netbeans ide 7.0 features: Web service development," 2011. [Online]. Available: http://netbeans.org/features/web/web-services.html [Accessed: September 10, 2011]

[17] MySQL, "Mysql connectors," 2011. [Online]. Available: http://www.mysql.com/products/connector/ [Accessed: September 10, 2011]

[18] L. DeMichiel, "Java persistence 2.0," November 2009. [Online]. Available: http://download.oracle.com/otndocs/jcp/persistence-2.0-fr-eval-oth-JSpec/ [Accessed: September 10, 2011]

[19] A. Nadalin, C. Kaler, R. Monzillo, and P. Hallam-Baker, "Web services security: Soap message security," Feb 2006. [Online]. Available: http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf [Accessed: September 10, 2011]

[20] BlackBerry, "Blackberry java application version 5.0 fundamentals guide," April 2010. [Online]. Available: http://docs.blackberry.com/en/developers/deliverables/9091/JDE_5.0_FundamentalsGuide_Beta.pdf [Accessed: September 10, 2011]

[21] T. Neward, "The vietnam of compuer science," Blog entry, June 2006. [Online]. Available: http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx [Accessed: September 10, 2011]

[22] A. Singhal, T. Winograd, and K. Scarfone, "Guide to secure web services: Recommendations of the national institute of standards and technology," National Institute of Standards and Technology, Tech. Rep. 800-96, 2007.