Chapter 12
A Framework for Rapid Development of REST Web Services for Integrating Information Systems

Lars Hagge, Daniel Szepielak, and Przemyslaw Tumidajewicz

Abstract Integrating information systems and legacy applications is a frequently occurring activity in enterprise environments. Service Oriented Architecture and Web services are currently considered the best practice for addressing the integration issue. This chapter introduces a framework for rapid development of REST-based Web services with a high degree of code reuse, which enables non-invasive, resource centric integration of information systems. It focuses on the general framework design principles and the role of REST, aiming to remain independent of particular implementation technologies. The chapter illustrates the framework's capabilities and describes experience gained in its application by examples from real-world information system integration cases.

Introduction

The concept of integration has been present in the software development domain in various forms for the last two decades. Over the years, integration approaches evolved from simple remote procedure calls (Brose et al. 2001) and message passing systems (Monson-Haefel and Chappell 2000) to service oriented solutions and have found their way to become integral parts of programming platforms like J2EE or .NET (Erl 2005). Recent years have witnessed an unprecedented shift in distributed computing towards Service-Oriented Computing (SOC) (Chang et al. 2006), which is gaining prominence as an efficient approach for integrating applications in heterogeneous distributed environments (Erradi et al. 2006). The most popular branch of SOC research is dedicated to advances in Service Oriented Architecture and SOAP Web services (Curbera et al. 2005), but the growing popularity of the

L. Hagge (✉)
Deutsches Elektronen-Synchrotron, Notkestrasse 85, Hamburg 22607, Germany
e-mail: lars.hagge@desy.de

Web 2.0 (Musser and O'Reilly Radar Team 2006) concept has brought increased attention to the REST architectural style as an alternative way of building service oriented environments (Howerton 2007; Vinoski 2007).

Building an integrated software environment in an enterprise often requires developing large amounts of Web services. The integration efforts can be greatly reduced by using a specialized framework for their development. Providing such tools that simplify software development in integration projects is essential for optimizing their efficiency and cost.

This paper describes a framework for rapid development of REST Web services which are suitable for integrating information systems. It first illustrates the application scenario with a simple example, which is used to explain the proposed integration architecture. Then, it introduces the framework architecture, putting particular emphasis on code reusability as basis for rapid development. The next section describes three application examples of the framework, and the final section summarizes experience gained and outlines possible next steps. The paper focuses on the general framework design principles and the role of REST, independent of particular implementation technologies.

Integrating Information Systems Using REST

One of the most important choices to make when building an integration solution is to select an appropriate integration approach and suitable technologies for its realization. These choices can vary depending on the characteristics of the software environment and the particular goals of the specific integration project. This section introduces an example integration scenario and uses it for deriving an architecture for integrating enterprise information systems. The architecture is based on a layer of REST Web services which provide unified access to the information systems of the integrated environment. The section concludes by discussing those types of integration for which REST is well suitable, and those for which it is not.

Integration Architecture

Figure 12.1 (a) shows a simplified information model for equipment documentation. It states that equipments have descriptions in terms of documents, where equipments can be complex items which are built using other equipments, and documentation can consist of various documents with cross-references and dependencies. The schema has to be adapted and specialized for each particular business, yielding an ontology of the target application area. An example is given for facility planning and plant design (b): Facilities are organized into functional subsystems. They comprise functional units, are driven by power supplies, are controlled by safety

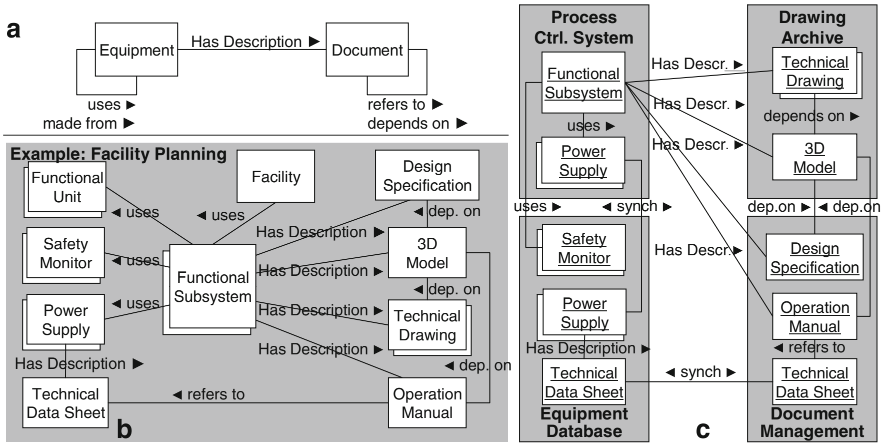


Fig. 12.1 Example scenario illustrating integrated information systems

monitors, etc., all of which are special types of equipments. They are described by a variety of technical documentation, including specifications, design models and drawings, work instructions, and operation manuals, all of which are special types of documents.

When it comes to implementation, ideally a single information system (IS) would support the entire ontology and its business processes, but in practice objects and functionalities are often spread over a number of systems. Figure 12.1 (c) shows a typical example for a deployment scenario: Operators use a process control system (PCS) for setting-up and running of the facility. Technicians use equipment databases (EDB) to keep track of the inventory and organize regular inspections and repairs. Management and staff use a central document management system (DMS) for review, approval and archival, while designers and engineers use a dedicated CAD drawing archive (CDA) for design models and drawings.

The information systems are not independent as there are business requirements which extend beyond the scope of individual systems. Consider the following examples:

1. An operator who may need to respond to an alarm in the PCS, e.g. of an over-heated power supply, would benefit from navigational support to the appropriate operation manual in the DMS.
2. A planned subsystem update, e.g. for improved performance, would require lead engineers to update specifications in the DMS, and then propagate necessary change information to different engineering groups, who then implement the change and update their equipment information and documentation accordingly. The objective would be to coordinate the entire business (change) process independent of any IS boundaries.

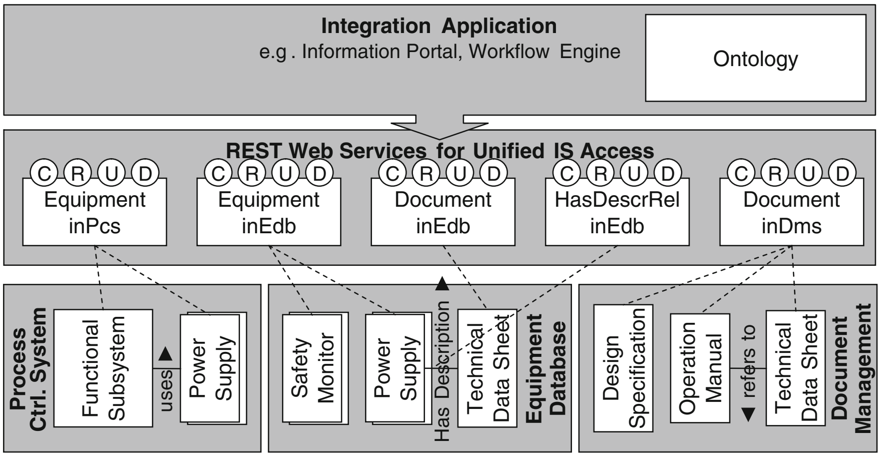


Fig. 12.2 Integration architecture using REST WS for unified IS access

The analysis reveals two characteristic groups of system integration requirements:

- Cross-system item relations: Relations need to be established across boundaries of IS, e.g. Has-Description relations between subsystem equipments in the PCS and technical documents in the DMS and CDA.
- Fragmented objects: Different aspects of the same item are stored in different IS, e.g. technical data are partially kept in the EDB (e.g. parameter values) and partially in the DMS (e.g. signed certificates).

Obviously, the IS environment has to be extended by a component which stores cross-system relations and synchronizes fragmented objects. This could be done by extending the capabilities of one or more of the available IS, or by introducing a dedicated integration component. The latter is preferable as it has the advantage of not interfering with available IS. For interfacing this integration component with the IS, an access layer should be foreseen which abstracts IS access to a uniform interface. Figure 12.2 summarizes the approach, proposing REST Web services for implementing the access layer.

This paper concentrates in the following on the REST Web services which are used for creating the unified access layer.

Identifying and Defining Resources

Integration of information systems based on REST architectural style is resource-centric in its nature as standard REST operations are tightly aligned with the CRUD pattern. Thus, the first step in building an integrated environment involves identifying and defining the resources in that environment. The resource-centric

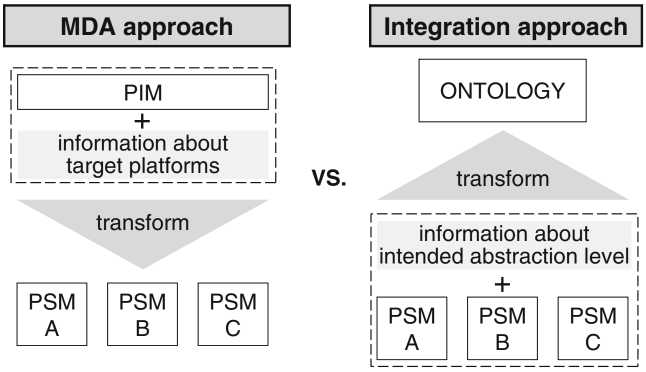


Fig. 12.3 Top-down design vs. bottom-up integration

approach requires a common vocabulary, which contains definitions of all resource types that are used by the participants. Such a vocabulary can be realized in many different forms. One of the most effective ways of formal knowledge representation and sharing it in a coherent and consistent manner among interacting software agents is ontology (Dietz 2006; Guber 1993).

At this point it should be noted, that the initial example can be read in two directions: Figure 12.1 may be the result of a top-down business design process (a–c), which deploys the overall information model to the best-suited available application platforms, or it may be the result of legacy applications which were independently introduced and afterwards brought bottom-up into co-operation (c–a). The former is similar to the MDA-approach (MDA 2003), which transforms high-level platform independent models (PIM) to target systems represented as platform specific models (PSMs). The latter, which is typical for integration projects, implies that the information model would have to be inferred from a bottom-up analysis of the different information systems.

Figure 12.3 illustrates and compares the two approaches. The ontology corresponds to the PIM in MDA terminology, and the models of the individual ISs correspond to PSMs. In MDA, the PIM is the first model to be created, followed by the generation of PSMs for specific platforms (Kleppe et al. 2003). Integration of existing information systems requires reversing this process, i.e. the PIM has to be derived from a set of PSMs of the existing information systems (Szepielak 2007). This involves:

- Comparative analysis of all PSMs to identify similar resource classes in multiple PSMs and ensure they are abstracted into the same concepts.
- Analysis of relations between resources across IS boundaries.
- Analysis of all PSMs to reveal overlapping resource classes and create according cross PSM model mappings.

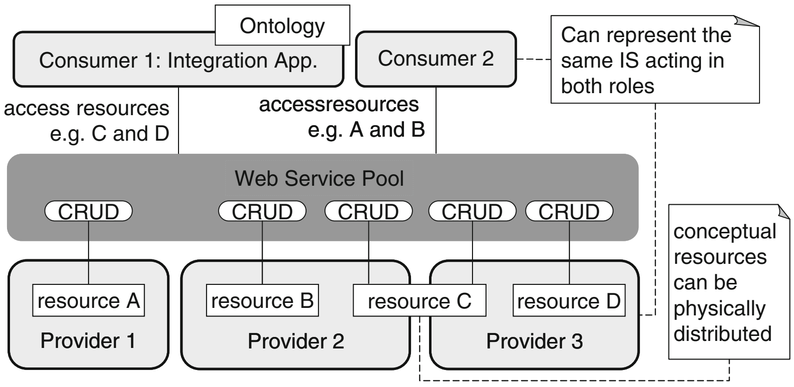

Ontology
Consumer 1: Integration App.
Consumer 2
Can represent the same IS acting in both roles
access resources e.g. C and D
accessresources e.g. A and B
Web Service Pool
CRUD
CRUD
CRUD
CRUD
CRUD
resource A
Provider 1
resource B
resource C
Provider 2
resource D
Provider 3
conceptual resources can be physically distributed

Fig. 12.4 Integration architecture

The emerging ontology has to be checked for consistency and compliance with the original IS data models. It will stabilize in an iterative process. Figure 12.4 redraws the integration architecture from a resource-centric perspective and emphasizes, that ISs can act both as resource providers and consumers.

Modelling Workflow as Resources

The rigorously applied resource-centric approach should completely avoid any form of thinking in categories of processes. All communication among information systems in the integrated environment should be performed with the help of resources only. From a REST perspective, the appropriate way of implementing processes is to represent them as sequences of CRUD operations which are executed on the resources of the ontology.

This approach is feasible for simple transactional workflows. Example 1 from "Integration Architecture", navigating from an alarm in the PCS to the corresponding operation manual in the DMS, could e.g. be written as

1. RETRIEVE status information of Equipment
2. RETRIEVE connected Has Description relation
3. RETRIEVE connected Document

More complex workflows will need richer expressions. Example 2, organizing an engineering change process, could start as

1. CREATE change request
2. APPROVE change request
3. UPDATE specification
4. APPROVE specification
5. …

In this example, approving items denotes that they are read, signed and this way endorsed by responsible persons. Such approvals or sign-offs are common functionalities of information systems, often provided as workflows. To remain compliant with the REST approach, such workflows also have to be represented and treated as resources. This requires translating all functional aspects of workflow into data structure and defining it as an ontology class. For manipulating workflow, the standard CRUD operations can be used with the following interpretation:

- *Create* – start workflow
- *Retrieve* – check workflow status
- *Update* – alter workflow execution
- *Delete* – abort workflow

The above example would then be re-written as

1. CREATE change request (cr-id, title, description, author, . . .)
2. CREATE approval workflow (cr-id. reviewer-1, reviwer-2, . . .)
3. UPDATE specification (. . .)
4. . . .

If it turns out that reviewer-1 is not available, an alternative reviewer may be assigned by updating the approval workflow. Authors may inquire how many reviewers have already processed the request by RETRIEVing the workflow status, and in case they discover mistakes, withdraw the request for approval by DELETing the workflow.

The described scenario shows how standard CRUD operations can be used to manipulate workflows within an information system.

The scenario neglects that in a "real" business process, the different actions would be conducted by different users. This would require additional steps of routing information to process participants and asking them to perform their actions and acknowledge their completion. Routing, acknowledging, etc. can be treated in the same way as described above for the approval workflow, which leads to the conclusion that any business process can be implemented with this schema.

In case of complex processes, the granularity of resources should be carefully considered. Defining too unspecific resources can lead to insufficient control over a process, while too detailed resources may impose too many actions on the IS users and thus become inefficient. On the other hand, building a library of general-purpose process building-blocks will allow quick and easy future process modification by simply rearranging items in the process sequence.

Suitability of the Proposed Integration Approach

The proposed integration approach has been specifically developed for integrating business information systems. It assumes an existing environment of legacy information systems, which are characterized by transactions-type processing of business objects. In such cases, the strategy of introducing CRUD Web services for accessing business objects and executing transactions is applicable. In other environments, such as e.g. agent-based systems, the applicability of the approach needs to be reviewed.

The effective application of the proposed approach requires careful consideration of the granularity of the information system resources which are exposed as Web services. With the growing number of Web services necessary for intersystem communication, the level of coupling increases, and the environment becomes harder to manage in case of future updates or changes. To avoid the necessity of creating and managing a too large number of Web services, it is advisable to define the ontology on the highest possible level of abstraction that still meets the needs of the IS that need to be integrated. In typical enterprise environments, this condition should be moderately easy to achieve. The proposed method can still be used if fine-grained Web services are necessary, but in such cases it is worth considering if those systems which require such tight integration would benefit from other integration techniques.

As the REST architectural style is highly resource-centric, it is very effective for integrating environments where the primary focus is on data access and manipulation, as representing data as resources is a very straight forward procedure. Environments which focus primarily on complex processes, which span over multiple information systems, are hard to integrate using REST. This is related to the fact that creating a resource interface to a process grows in complexity with the process complexity and the number of involved information systems. While it is always possible to provide access to a complete process through multiple resource interfaces to parts of the process, again the then high number of Web services can lead to tight coupling and according difficulties in maintenance.

While there are situations when the described integration approach is not the best way to proceed with an integration project, it also has its undisputable advantages. First, it allows for a non-invasive integration of existing information systems. The REST Web services that allow for accessing and manipulating information system resources are built on top of the existing code base and have no direct impact on the systems. This ensures that operation of an existing IS (provider) can continue without any interruptions, and users do not experience any side effects to the way they used to work with the system. On the other hand it still allows other IS (consumers) to interact with it through its new interfaces.

Second, in the proposed approach all information systems in the environment can be accessed with the same consistent API, which compared to situations where each IS has its own API greatly reduces development efforts and cost. In particular, the proposed approach allows developers to realize complete integration scenarios without the necessity to learn individual IS APIs, as more IS are added to the environment.

Framework for Rapidly Developing REST Web Services

This section introduces a REST Web services framework with specific emphasis on rapid development. Building an integrated enterprise environment, which often consists of dozens of individual applications that should cooperate, requires developing large families of Web services. The described framework can considerably speed up the development process by achieving high levels of code reuse and easing code maintenance. The section presents the framework architecture, introduces the strategy for code reuse, and describes how the framework is effectively used in large environments.

Objectives

Building an integration solution based on the proposed approach requires developing families of Web services for all resource classes which are defined in the ontology. A development framework shall satisfy the following requirements:

- The framework shall minimize development efforts and time, as the expected number of Web services which have to be provided may be high.
- The framework shall support changing and adapting Web services when the environment evolves; e.g. new ISs are introduced or existing ISs updated.
- The framework shall ensure the Web services are uniform in their structure as far as possible, to make them easier to understand, use and maintain.

As general strategy, the framework shall attempt to achieve an as-high-as-possible degree of code reuse, as code reuse is an efficient and established method of increasing productivity and reliability (Gui and Scott 2006), which significantly accelerates and reduces the development cost of new software (Boxall and Araban 2004). Among other advantages, it:

- Reduces the amount of code to be developed, and thus effort and time.
- Keeps the code base small, and thus eases quality assurance.
- Minimizes code duplication, and thus side-effects of changes.

Strategy for Code Reusability

Reusability is defined as the degree to which a software module or other work product can be used in more than one computing program or software system. The proposed framework facilitates reuse for developing families of Web services by providing code blocks that can be generalized for all or a subset of Web services.

Figure 12.5 illustrates the approach: It shows three Web services, one for retrieving equipment information of power supplies from the PCS, and two for

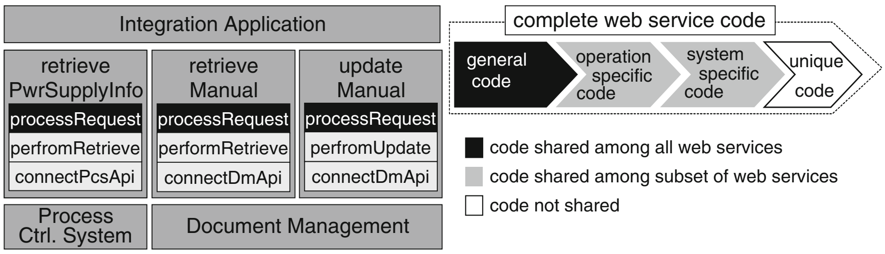


Fig. 12.5 Approach to code reusability

retrieving and updating documents, in particular operation manuals, in the DMS (left). A closer look reveals that:

- The Web services shall provide uniform access to the underlying IS, hence al three of them should respond to similar URIs and provide similar response.
- Two Web services are providing retrieve operations, hence they should have similar internal sequencing.
- Two Web services are serving the same business object, hence they should use the same data definition.
- Two Web services are connecting to the same IS, hence they should use similar calling sequences of the IS API.

Or, more general, the Web services exhibit three major sources of reusability:

- General code that can be shared among all Web services
- Code shared among Web services which are performing the same operation (e.g. code shared between all "update" services)
- Code shared among Web services for accessing a specific IS

The code for a particular Web service can thus be separated into general code, operation specific code, target-IS-specific code, and unique code for that Web service (Fig. 12.5, right). Except for the unique code, the code can be provided by the framework, which is designed to represent each source of reusability by a code block which encapsulates the according functionality:

- A *Request-Response Processor* (RRP) provides functionality which is specific for all Web services, such as request parsing, response formatting and exception handling
- *Operation Controllers* (OC) provide the generic sequencing for specific operations, which can be used for any type of object. E.g. the generic "update" can be written as:
  - open a transaction,
  - find an object using a given set of filtering attributes,
  - if the object exists, update it with a set of new attribute values,
  - on success commit transaction,
  - otherwise, rollback

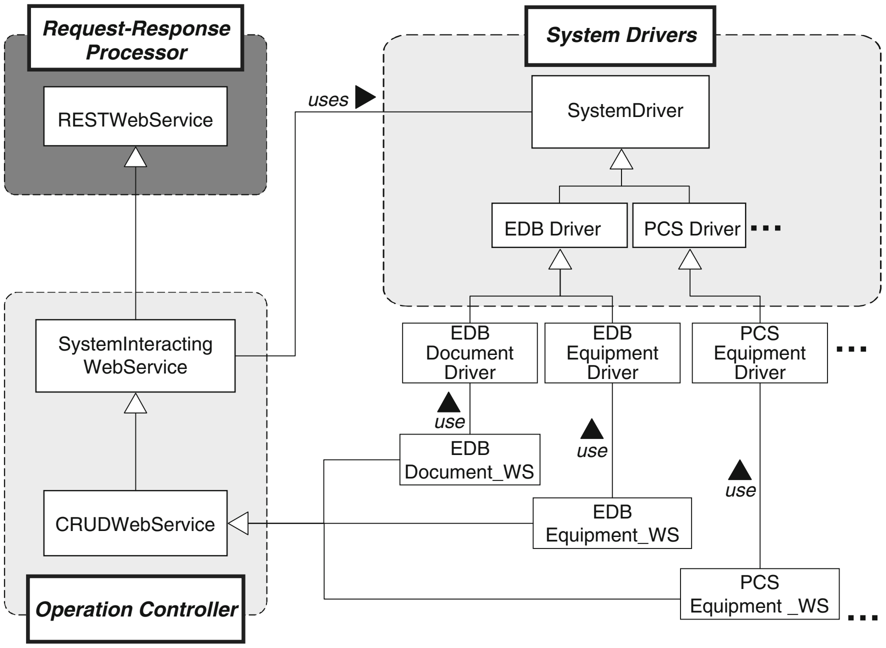


Fig. 12.6 Framework structure

- *System Drivers* (SD) provide functionality for accessing specific ISs, such as connecting to and disconnecting from the IS, beginning and finalizing transactions, and creating, locating, retrieving, modifying and deleting resources. *System Drivers* encapsulate the IS APIs

Framework Structure

Figure 12.6 presents the structure of the framework and the mapping of its classes to functional layers. The left side of the class hierarchy contains classes which implement the operational skeleton of a Web service, while the right side represents system specific code. Figure 12.7 illustrates the interplay of the different classes in an activity diagram. Partitions represent framework classes, and the allocation of actions shows their implementing classes. Structured activity blocks spanning multiple partitions represent abstract methods.

The topmost *RESTWebService* abstract class encapsulates functionality which is common to all REST Web services and realizes the *Request–Response Processor*. The class is responsible for handling incoming requests, processing them and passing their parameters together with stored configuration to the abstract *action()* method for operation-specific processing. Upon successful completion of the operation, the results are formatted and returned. In case of failure, exception handling takes place and error messages are returned.

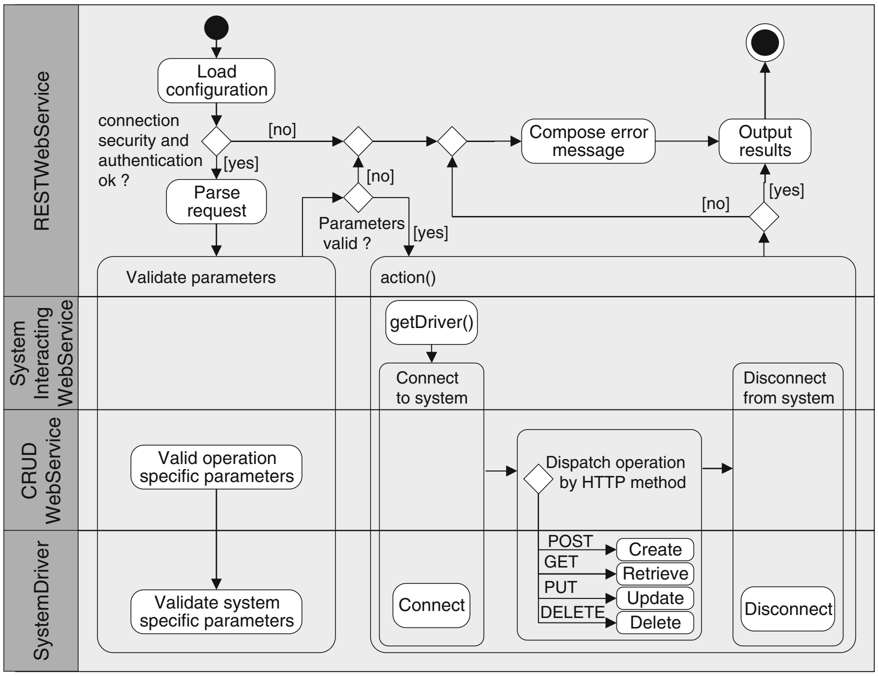


Fig. 12.7 Activity flow and implementation distribution of the framework

The abstract *action()* method is narrowed down in the *SystemInteractingWebService* class, which isolates system interactions common for all Web services. An instance of a *SystemDriver* is created through a call to an abstract factory method *getDriver()* and uses it to connect to the system. Finally, the *CRUDWebService* class performs the dispatching of operations into Create (POST), Retrieve (GET), Update (PUT) and Delete (DELETE) depending on the HTTP method used for the request. Each of the four basic operations is decomposed into a sequence of atomic method calls, which can be overloaded if necessary by an extending subclass.

The *SystemInteractingWebService* and *CRUDWebService* classes realize the *Operation Controller* functionality. In case of operations other than CRUD *SystemInteractingWebService* should be extended by a class that implements a controller for a specific operation.

The *SystemDriver* abstract class acts as a flexible interface for the enclosed set of atomic interactions. It enforces the implementation of basic interactions (like connect and disconnect), but allows for partial implementation of the CRUD functionality and runtime checking of driver capabilities. Such implementation is useful for practical reasons, as not all system resources allow or require the full set of operations. The *SystemDriver* class is a parent class for all specific information system drivers. Single information systems can contain many different types of resources, which require different behavior of the driver at a system specific level.

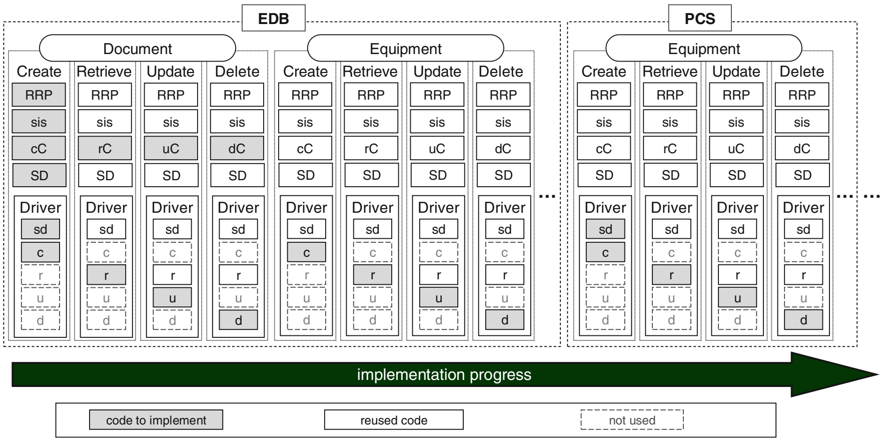


Fig. 12.8 Effective code reuse in Web Service development

Therefore, drivers for specific ISs can have sub-hierarchies of drivers for specific types of resources. The structure of the hierarchy is not enforced by the framework and can be built according to the specification of a given IS.

The actual Web service classes for each of the resource types extend the *CRUD-WebService* class (e.g. *EDB_Equipment_WS*), thereby inheriting the full operational skeleton, and implement the *getDriver()* method so that it produces an instance of a *SystemDriver* subclass appropriate for the particular system-resource combination.

The Framework at Work

The framework structure has been designed to allow for various development strategies depending on the type of a required Web service. There are three basic development paths that are enabled by the framework:

- Rapid development of CRUD Web services for inclusion of new resources to the integrated environment by adding new drivers
- Development of system specific operations not belonging to the CRUD set by extending the *SystemInteractingWebService*
- Development of freeform REST Web services by extending the *RESTWebService*

Figure 12.8 illustrates the rapid development of CRUD Web services for accessing equipment and document information in an EDB, and document in a DMS. Once the first Web service has been completed, subsequent Web services require only minor and well-encapsulated development efforts (Szepielak 2007).

The figure shows that each Web service comprises 6–7 code blocks from the framework, only 1–2 of which need to be newly provided when the pool of Web services is extended. Assuming the code blocks to be of equal size and complexity, this would correspond to 14–33% of code needing to be provided, or an expected average code reuse of at least 70%. This number can get much higher, if the components which have to be developed are small compared to the others.

The different development paths offer developers great flexibility and allow using only partial framework functionality, if required. This way, the potential framework application extends beyond the described integration scenario and allows it to be used for general software development purposes.

Application Examples

The integration approach and the REST Web-service framework have been developed and applied in the engineering data management domain at Deutsches Elektronen-Synchrotron DESY in Hamburg, Germany. DESY is one of the world's leading centers for research at particle accelerators. DESY develops, builds and operates particle accelerators, which are large scientific instruments, and conducts basic research in a great variety of scientific fields, ranging from particle physics to materials science and molecular biology.

This section describes three application examples of the presented REST WS framework: Integrated information access across several information systems, synchronization of information between existing systems, and building new applications on top of an existing environment. The examples involve some of DESY's key information systems, namely:

- The DESY Engineering Data Management System (EDMS), a customized product lifecycle management (PLM) solution
- A combined Geographic Information System and Facility Management System (GISFMS), built with various commercial components
- An Inventory Management System (IMS) based on a commercial IT Asset Management System

Integrated Information Access

DESY has developed a powerful portal which allows users to jointly and intuitively search and navigate the GISFMS and EDMS. The portal provides information about the DESY facilities (buildings and accelerators) through means of metadata querying, hierarchy browsing or visual navigation using maps. The information provided through the portal includes maps and building information from the GISFMS, related with documents and 3D CAD models from the EDMS. REST Web services are used for connecting to the GISFMS and EDMS, querying the systems, and retrieving (lists of) objects.

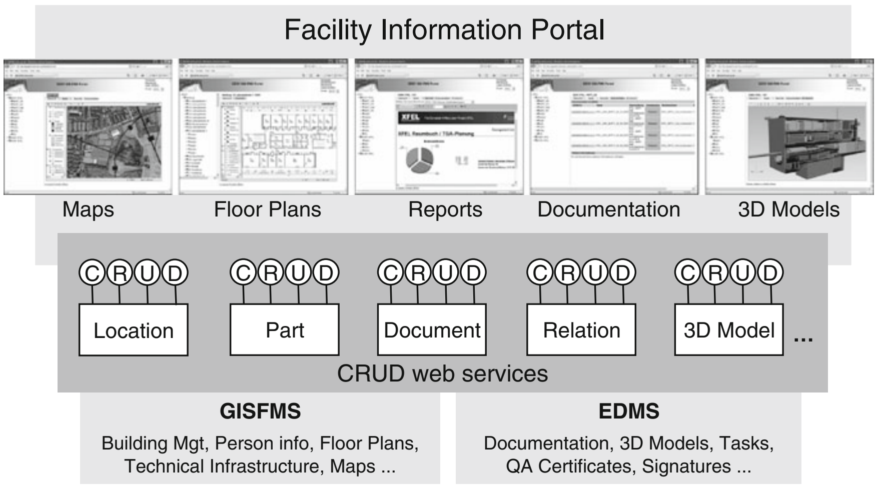

Facility Information Portal
Maps
Floor Plans
Reports
Documentation
3D Models
C R U D
Location
Part
Document
Relation
3D Model
...
CRUD web services
GISFMS
Building Mgt, Person info, Floor Plans, Technical Infrastructure, Maps ...
EDMS
Documentation, 3D Models, Tasks, QA Certificates, Signatures ...

Fig. 12.9 Portal for integrated information access

Figure 12.9 illustrates the architecture of the portal application. The portal provides location-centric information access, i.e. locations are the primary key to information access. For this purpose, the portal provides a tree browser which enables navigating from sites through buildings and floors to rooms, and a map and plan viewer are provided. The Web components retrieve their data from the GISFMS database using CRUD Web services.

The location information of the GISFMS is mirrored and synchronized in the EDMS, where documentation, technical drawings and 3D models are processed and related with their locations. CRUD Web services enable accessing locations, documents, models etc. and traversing relations in the EDMS.

At the time of writing, the portal is already in operation for three years. It serves information for a large-scale accelerator construction project and needs to adapt to growing and changing requirements as the project progresses. So far, it has been both very robust and flexible against changes: Additional information types, such as e.g. 3D model viewing, have been added to the portal without impact on available functionalities, and major software upgrades of the underlying information systems have been successfully carried out without affecting the portal functionality.

Synchronizing Information

A Web-based information system had been developed based on EDMS and IMS for coordinating the installation process in an accelerator project. It registered all the components of the accelerators, provided work lists for the various technical groups, tracked the installation progress, and provided a central information access

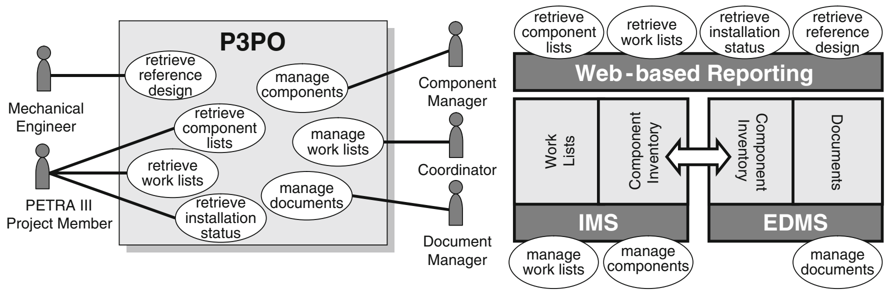


Fig. 12.10 Using IMS and EDMS to support the installation process of a large facility

point for the installation status. The IMS was used for component and infrastructure management and handling work lists, while the EDMS managed the technical documentation of components. An integration component ensured consistency of the information in both systems by propagating changes in one system to the other. The integration component used the REST Web services framework to connect to the systems, access and update objects, and trigger workflows.

Figure 12.10 summarizes the scenario. The different actors are working directly with the ISs, as their roles are mapping 1:1 to one of the systems. Coordinators and process managers use the rich native IMS or EDMS interfaces. The other project workers, who are carrying out installation works in the accelerator facility, are able to retrieve work lists and instructions through a Web-based reporting interface. An integration application in the background ensures that information changes from one system are propagated to the other: If a crucial information change is retrieved from one system, an update Web service is called which propagates the change to the other system.

The application has been realized in very short development time. It was built on top of two information systems, which were in production and starting to show an information overlap. According to the approach, the application has been non-invasive, i.e. did not affect other projects that were also using the EDMS and/or IMS for their activities.

Building New Tools and Applications

The presented framework can be used to build new, specialized clients on top of existing systems. As the DESY EDMS is a very large and complex system, users often request lightweight and easy to use clients for special purposes. The Web services are efficient building blocks for such applications by providing the necessary basic functionalities for connecting, accessing and updating information.

Figure 12.11 illustrates a number of tools and applications, which have been built on top of the DESY EDMS using the REST Web-services framework. They include e.g. direct document searches and accesses from public project Web pages, bulk

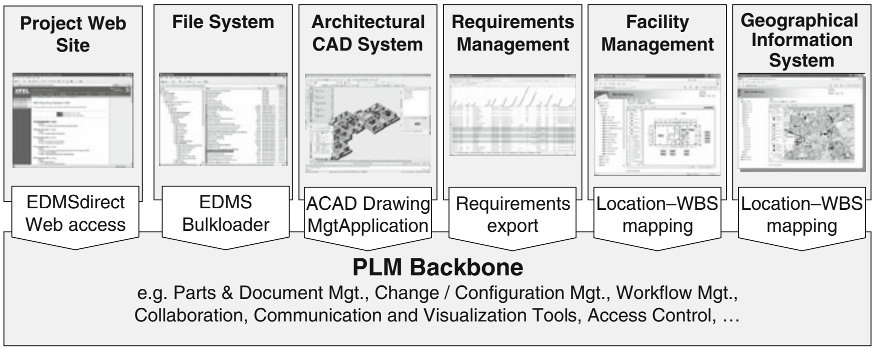

Project Web Site
File System
Architectural CAD System
Requirements Management
Facility Management
Geographical Information System
EDMSdirect Web access
EDMS Bulkloader
ACAD Drawing MgtApplication
Requirements export
Location–WBS mapping
Location–WBS mapping
PLM Backbone
e.g. Parts & Document Mgt., Change / Configuration Mgt., Workflow Mgt., Collaboration, Communication and Visualization Tools, Access Control, ...

Fig. 12.11 Special-purpose tools and applications on top of the DESY EDMS

loaders for batch upload of large amounts of files, and connectors for exchanging and synchronizing data with other external databases and applications. Many of these tools are requested at extremely short notice. With the framework in place, such requests can usually be handled.

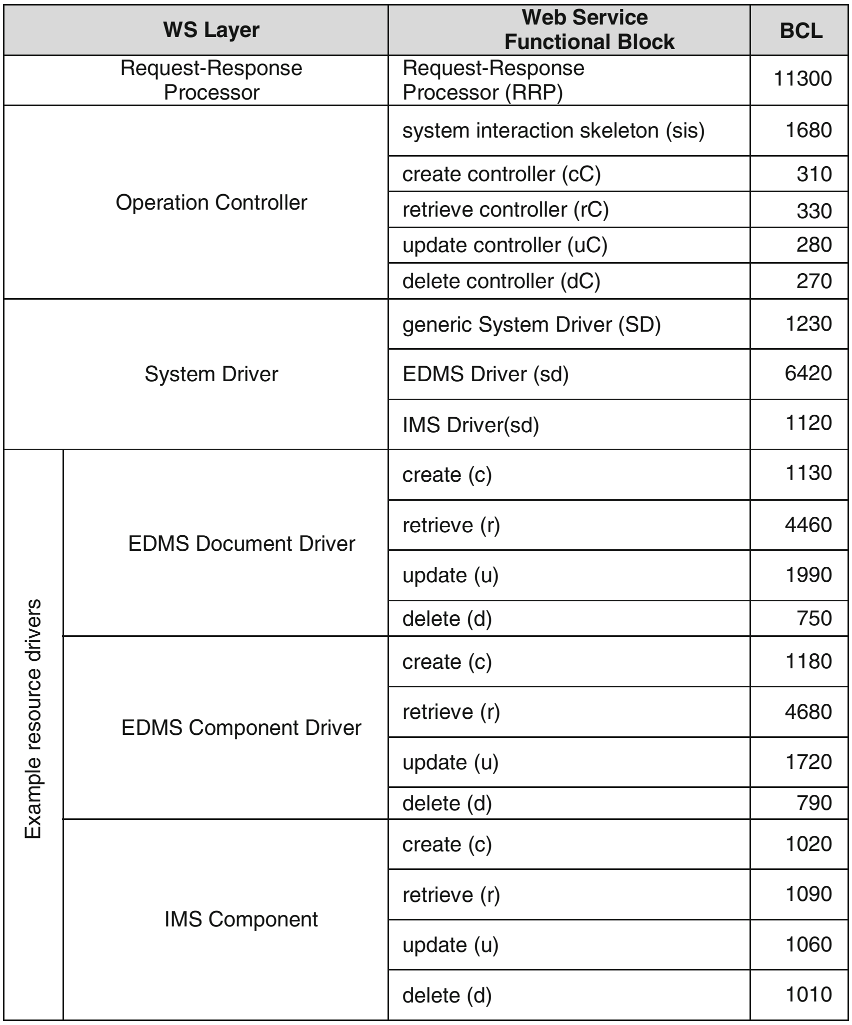
Summary

This chapter summarizes results and experience from implementing and operating the described framework, and provides an outlook on a strategy for extending the framework architecture for automating the integration of information systems.

Results

The first components of the presented framework are in stable in operation since their initial deployment at DESY in 2005. Numerous extensions and applications have been developed since then, increasing both the scope of operations and the number of accessible information systems.

Figure 12.12 shows the byte code length for the different code blocks of the framework as they have been measured for the initial set of Web services. Figure 12.13 shows the increasing level of code reuse that has been observed as more and more Web services have been developed (Szepielak 2007). The observed level of reuse for all Web services operating in the DESY environment ranges between 83% (for the most complex Web services) and 98% (for the simplest Web services) with an average of 93%. The calculations concern only the internal level of reuse of the framework code itself. Taking into account external libraries used to build the framework, as well as the fact that the Web services are designed to be used in multiple applications, the average level of reuse exceeds 95%.



WS Layer		Web Service Functional Block	BCL
Request-Response Processor		Request-Response Processor (RRP)	11300
Operation Controller		system interaction skeleton (sis)	1680
		create controller (cC)	310
		retrieve controller (rC)	330
		update controller (uC)	280
		delete controller (dC)	270
System Driver		generic System Driver (SD)	1230
		EDMS Driver (sd)	6420
		IMS Driver(sd)	1120
Example resource drivers	EDMS Document Driver	create (c)	1130
		retrieve (r)	4460
		update (u)	1990
		delete (d)	750
	EDMS Component Driver	create (c)	1180
		retrieve (r)	4680
		update (u)	1720
		delete (d)	790
	IMS Component	create (c)	1020
		retrieve (r)	1090
		update (u)	1060
		delete (d)	1010

Fig. 12.12 Byte code length of code blocks in the initial set of Web services

Experience

Setting up the framework was experienced as a time consuming process, but the initial time spent on building the framework resulted in faster and more efficient development of the necessary Web services. The framework allows developing new Web services for accessing further objects from underlying information systems within a few days of work, thus assuring scalability for dynamic environments and increasing integration. The framework also greatly eases the maintenance of existing code.

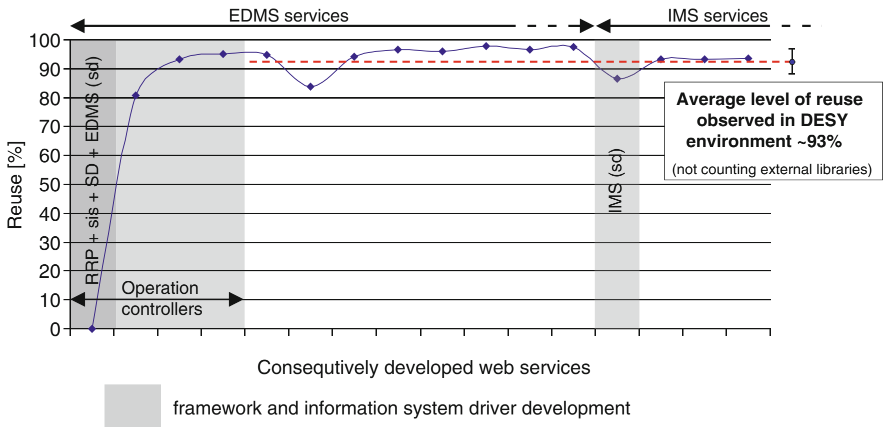


Fig. 12.13 Increasing level of code reuse observed during Web service development

The framework has been built completely from scratch, as at the time of the project no mature enough frameworks were available. With the official JSR for RESTful Web Services in place, JAX-RS (JSR311: JAX-RS 2009), a similar integration framework could be created based on one of the available JAX-RS implementations. Most of the functionality of the RESTWebService and CRUDWebService classes could be taken directly from e.g. Sun's reference implementation of JAX-RS, Jersey. The other classes would still need to be custom-developed, as they are specific to the presented integration framework and to date not available in any generic REST framework.

Several of the underlying information systems have undergone major software upgrades. As the framework successfully encapsulated those systems, no side effects were observed on applications which were built using the Web services. As newer versions of underlying ISs offer richer functionality, some of the Web services may need to be extended to make this functionality also accessible to other applications. In such cases, the REST CRUD paradigm has shown to be well-suited for maintaining backward compatibility and thus avoid impacts on productive environments.

Also the resource-centric approach has shown various advantages in the software development process. The major advantage is that it reflects the business vocabulary, which is particularly beneficial for developers, as they do not need to familiarize with specific system APIs, but can work with a high-level intuitive information access layer which is addressed in the same vocabulary as used in the business itself. This greatly reduces the time until developers get productive and at the same time improves the quality of the resulting software. For example, some of the tools and applications described in "Building New Tools and Applications" have been developed by new staff or students within the first month of their work.

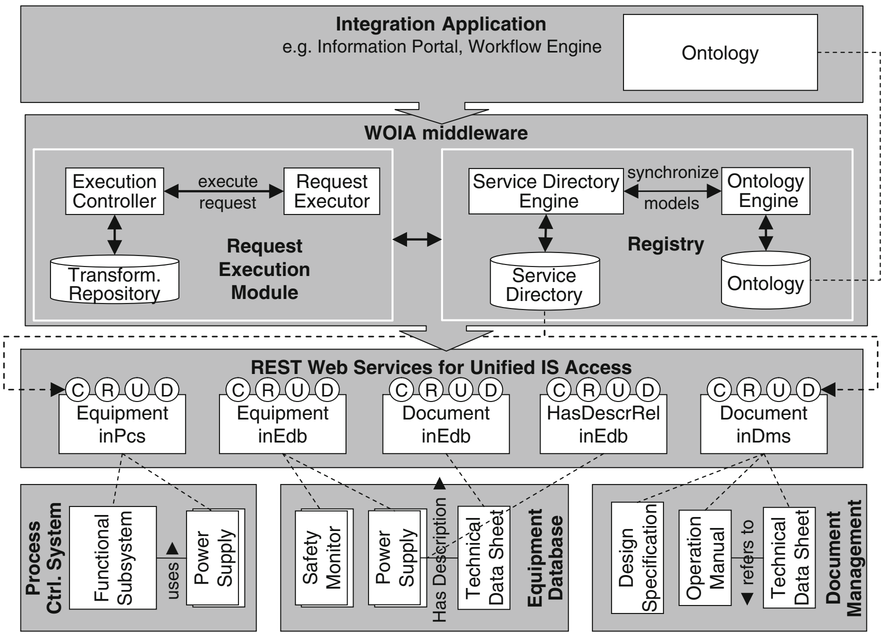


Fig. 12.14 Web-oriented integration architecture (WOIA)

Extending the Integration Framework

The analysis in "Integration Architecture" has shown that all integration applications share two core functionalities: They need to be able to establish cross-system relations, and to handle business objects which are fragmented over several ISs. DESY has developed a dedicated integration application which generalizes these capabilities. It shall act as a middleware which provides Web service registration, discovery, composition and execution capabilities. The architecture which employs this middleware is called Web-Oriented Integration Architecture (WOIA) (Szepielak 2007; Szepielak et al. 2010).

Figure 12.14 illustrates the WOIA middleware in the context of an integrated environment as shown in Fig. 12.2: It consists of a registry and a request execution module, which are both using the ontology to operate. Information systems register within the registry as providers of resources which are defined in the ontology. Based on the registration data, the request execution module allows consumers to directly operate on resources without any knowledge about their providers, almost as if the middleware itself would be providing all the Web services. The middleware has a REST interface which allows the consumers to interact with it in the same way as they would with any other REST service: Consumers send requests for required resources directly to the middleware (the only part of the request that changes is the host name), and the middleware will automatically identify the necessary providers, invoke the required Web services and compose the response, in case of distributed resources by combining responses from several Web services. It also enriches the response with links pointing to the related resources based on the information retrieved from the ontology before the complete response is sent to the consumer.

Using such a generic middleware has the potential to reduce the integration effort to defining an ontology and providing system and resource drivers for the available information systems, while the rest of the required integration software would be provided by the framework.

References

Boxall, M.A.S., Araban, S.: Interface Metrics for Reusability Analysis of Components. In Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04). IEEE Computer Society, Los Alamitos, CA, pp. 28–37, 2004

Brose, G., Vogel, A., Duddy, K.: JavaTM Programming with CORBATM: Advanced Techniques for Building Distributed Applications. Wiley, NY, USA, 3rd edition 2001

Chang, M., He, J., Castro-Leon, E.: Service-Orientation in the Computing Infrastructure, In Proceedings of second IEEE International Symposium on Service-Oriented System Engineering (SOSE'06), 2006

Curbera, F., Weerawarana, S., Leymann, F., Storey, T., Ferguson, D.F.: Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More. Prentice Hall PTR, Englewood, Cliffs, NJ 2005

Dietz, J.L.G.: Enterprise Ontology: Theory and Methodology. Springer, New York 2006

Erl, T.: Service-Oriented Architecture (SOA): Concepts, Technology, and Design. Prentice Hall PTR, Upper Saddle River 2005

Erradi, A., Anand, S., Kulkarni, N.: Evaluation of Strategies for Integrating Legacy Applications as Services in a Service Oriented Architecture. In Proceeding of IEEE International Conference on Services Computing (SCC'06), 2006

Guber, T.R.: A Translation Approach to Portable Ontologz Specifications. Academic Press, New York 1993

Gui, G., Scott, P.D.: Coupling and Cohesion Measures for Evaluation of Component Reusability. In Proceedings of the 2006 International Workshop on Mining Software Repositories. ACM Press, New York 2006

Howerton, J.T.: Service-Oriented Architecture and Web 2.0. IT Professional, vol. 9, no. 3, pp. 62–64, May/Jun 2007

JSR311: JAX-RS: The JavaTM API for RESTful Web Services available at: http://jcp.org/en/jsr/summary?id=311, accessed on June 08, 2011 (2009)

Kleppe, A., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Professional, Reading, MA, USA, 1st edition 2003

MDA Guide Version 1.0.1 available at: http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf, accessed on June 08, 2011 (2003)

Monson-Haefel, R., Chappell, D.: Java Message Service (O'Reilly Java Series). O'Reilly Media, 1st edition 2000

Musser, J. and O'Reilly Radar Team: Web 2.0 Principles and Best Practices. ISBN: 0–596–52769–1 O'Reilly Radar 2006

Szepielak, D.: Web Oriented Integration Architecture for Semantic Integration of Information Systems, PhD Thesis, Silesian University of Technology, Gliwice/DESY, Hamburg 2007

Szepielak, D., Tumidajewicz, P., Hagge, L.: Integrating Information Systems Using Web Oriented Integration Architecture and RESTful Web Services, pp. 598–605, 6th World Congress on Services 2010

Vinoski, S.: REST Eye for the SOA Guy, IEEE Internet Computing, vol. 11, no. 1, pp. 82–84, 2007