

# GARUDA: A Case Study of Service Oriented Application Integration Framework

Raghu Anantharangachar\*

International Institute of Information  
Technology  
Bangalore, India  
[araghu@hp.com](mailto:araghu@hp.com)

Chevuru Nagarjuna Reddy

Hewlett Packard  
Bangalore, India  
[Chevuru.nagarjuna-reddy@hp.com](mailto:Chevuru.nagarjuna-reddy@hp.com)

Doraiswamy Ranganathan

Hewlett Packard  
Bangalore, India  
[Doraiswamy.ranganathan@hp.com](mailto:Doraiswamy.ranganathan@hp.com)

## Abstract

Every Organization wants to increase the level of automation, collaboration and reuse within its departments. Each department has its own set of workflows/processes, and applications. As the organization grows, the numbers of applications increase, and this impacts the operational aspects. If the applications are operating in silos, then the users are forced to use the applications individually. This approach is error prone, and lacks performance, and hinders reuse. At the same time, it may not be possible to integrate every application. When an organization is using many applications then introducing newer applications results in huge training and support costs. In the current paper, we examine some of the problems involved in such a scenario, and propose a solution that solves many of these problems.

**Categories and Subject Descriptors** D.2.2 [Software Engineering]: Design Tools and Techniques D.2.11 [Software Engineering]: Software Architectures

**General Terms** Algorithms, Management, Design, Standardization, Performance

**Keywords** Service Oriented Architecture, Repository, Integration, Data Model, Meta data

## 1. Background

Within our organization, we are using applications for various purposes like SPRINT (a home grown tool for collecting project effort related data), SAP for collecting billing information, (G)HRMS for collecting information about the people (Human Resources) and so on. These applications are working in silos today, and there is no leveraging of information across these applications. For example, when a project manager needs to prepare a report requiring data from SAP and SPRINT, he manually collects that information and prepares the report. In addition, there are multiple applications catering to the same domain – for example, both SPRINT effort application and OMEGA are used for collecting effort data. These applications are individually deployed across the company, and the end-users are trained on each of these applications. We have a large user base of more than 10,000 people who are using these applications.

## 2. Introduction

EAI (Enterprise Application Integration) deals with approaches and mechanisms to integrate multiple applications, and deliver

productivity improvements, increase reuse, and as well to build collaborative framework for sharing information across various departments within the enterprise. Service Oriented Architecture (SOA) is a paradigm in which service consumers and service providers co-exist, without any dependency on each other. In addition, the service consumers consume the services provided by the service providers, and they typically do this over an integration media.

There are some important pain-points that are observed in most of the EAI scenarios (which are as well common to the case study that we are describing), and they are described in the rest of this section.

**Multiple Functionally Overlapping Applications:** Most enterprises today use multiple applications, and as each is coming from a different vendor and is catering to a different domain, they are inherently different. They support multiple user interfaces and they have their own way of storing data. Some applications use web based interface, whereas some use application level interfaces. The support for different interfaces poses a serious challenge to the system integrator. In our case, we have multiple (possibly overlapping) applications (example – SAP, Global Human Resource Management System (GHRMS), Billing Information Portal (BIP), SPRINT effort tool and many more) and each has its own user interface and storage mechanism – thus resulting in scope for data inconsistency as well as multiple options for performing the same operations.

**Duplicate and partitioned Data:** The data is partitioned, duplicated and there is no logical ownership for data. Each application works with data, and this data is meant to include configurations, customizations, log files and other related data files that are related to the application. As the applications are usually used over multiple sites, they also define and support a partitioned data storage mechanism. However, when there are multiple applications which support distributed storage patterns of data, it is difficult to arrive at a suitable data model that can identify the stewardship for the data owned by various applications uniquely. This results in data inconsistency.

**Lack of Separation of Concerns:** The business logic and activities are together, and hence changing business processes is not easy. Typically, the enterprises implement various business

\* The author is working with HP India

processes, and each business process supports a specific business use case. And, each business process includes multiple sub-processes which also perform some related tasks for achieving the overall goal of the business process. In addition, as each business process works across multiple functional areas (which are typically realized using multiple applications), there is also a need for performing specific business activities that would support the overall business process in the context of the functional area.

**Non-standard Application Customization:** Application customizations are not standardized. As the users are working with applications directly, and as each has a different need to customize the application, the customization on the applications are becoming non-standard. It is required to support the customizations in a standard way, and ensure that there is consistency across users when customizations are performed. Otherwise, the customizations become non-maintainable. The users customize the applications to suit their own needs – by writing some scripts and this soon becomes difficult to maintain. This results in wasted efforts, as the reuse potential is reduced.

**Silo'ed Applications:** The applications are operating in silos. As the enterprises are acquiring applications over a period of time, and on a need-basis, they are diverse. In addition, typically the applications also work in silos, and do not share information with other applications in a systematic manner. This results in loss of productivity as a human intervention is required to ensure consistent data transfer across applications. SPRINT collects all the effort related details, and this data needs to be pulled out by a user before it can be fed into another application (say SAP) for generating the billing details. This needs to be done by a user manually.

**Difficulty of introducing newer applications:** Deployment of new application requires training and support for all users. When an enterprise needs to introduce a newer application or replace an existing application, each user needs to be trained on the new application. This case is especially redundant when a newer application is replacing an existing application. In other words, introducing a newer application is a very time consuming and expensive task.

**Need for using custom applications:** The chosen applications many times do not support all required functionality, and hence people prefer to use their own custom applications (like Microsoft Excel for keeping track of effort data). Users resort to creating their own applications or customize applications in their own way, if the functionality they are looking for is not found in existing applications. Many of these custom applications reside on the laptops of individual program/project managers, and hence it becomes much more difficult to do away these custom components.

**Lack of integration support in Applications:** Many applications only support off-line integration. Some applications do not support API based integration, and hence need file based integrations with other applications.

### 3. Current Approaches and State of the Art

Data integration [1] and Application integration is a well researched area. Integrating applications using point-to-point integrations or hub-and-spoke architectures is one common practice. This approach is not really scalable, as the maintenance issues start cropping up as the number of applications increase. In addition, this approach also has large number of failure points due to the increased number of point to point links. In other words, the chance of occurrence of major failures is higher, when the product acting as the hub (usually the process management product) fails.

Another approach that is followed is the “vanilla” EAI [2][3] integration in which the atomic APIs exposed by the applications is directly used in integration, and results in thick adapters which tend to be overloaded with functionality. This approach suffers the problem of lack of separation of concerns, and does not address the same.

Using Resource Oriented Architecture to integrate the various applications is another approach in which the applications are modeled as resources. This approach is commonly used in inventory-centric systems, in which inventory plays a major role, and thus also becomes a bottleneck. Similarly, the Interface/Component Oriented Architectures seem to give enhanced focus to interfaces, and this is common in distributed architectures like CORBA. These systems lack to convey the functionality thru the interfaces, and hence they seem to fall short.

The latest trend is SOA, and this seems to be promising, though there are many research issues [4][5][6] to be worked on and resolved. Our approach is primarily based on SOA.

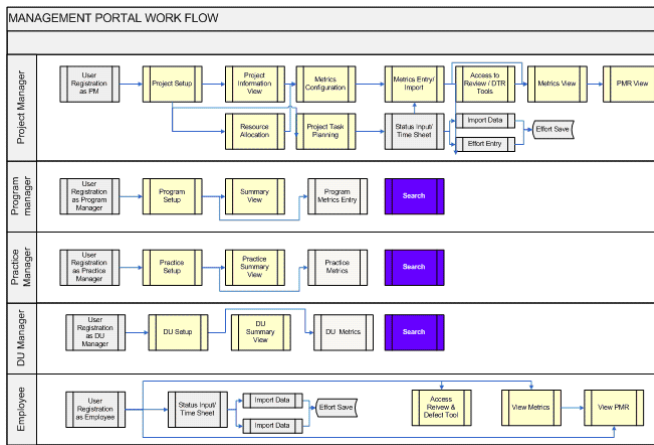
### 3. Our Approach

Our approach is an attempt to solve many of the problems stated earlier, and it is based on a Service Oriented Architecture. The highlight of our approach is given below:

1. **Achieving Service Oriented Application integration** – We defined and built a framework based on Service Oriented Architecture (SOA) conforming to Organization for Advancement of Structured Information Standards (OASIS) reference model [7]. We have identified and defined a set of services which are exposed to end-users and these services (realized using web services [8]) are supported by applications in the back-end. We are building adapters to interface the applications with the framework. In our framework, we have service consumers (which are typically end-users) which interact with the service providers (which are typically applications) over an integration media (which is realized using HTTP transport). The service consumers discover the services published by the service providers using a service repository (service catalogue). The services are realized as web services which are invoked using SOAP [9] (Simple Object Access Protocol) messages over HTTP transport. We also have support for other transport mechanisms for invoking the web services (specifically FTP and RSS, in addition to HTTP and email), which enable the consumers to use alternate invocation mechanisms to invoke the services.

**Process Control** – We have provided support to use a Process Manager (either Business Process Execution Language (BPEL) [10] based or any other generic process management like TIBCO BusinessWorks) to define business logic and as well to orchestrate the interactions within the specified use cases. These processes are exposed as services in the user interface, so that the user can select any one of these services and invoke them directly from the user interface. These processes perform specific tasks like logging the effort details for a specific employee and so on. They consist of different types of nodes – start/stop nodes which initiate/terminate the workflow, compute node which performs a specific processing task, and so on. These business processes are meant for use by the business managers.

The workflow for the various roles is given below:

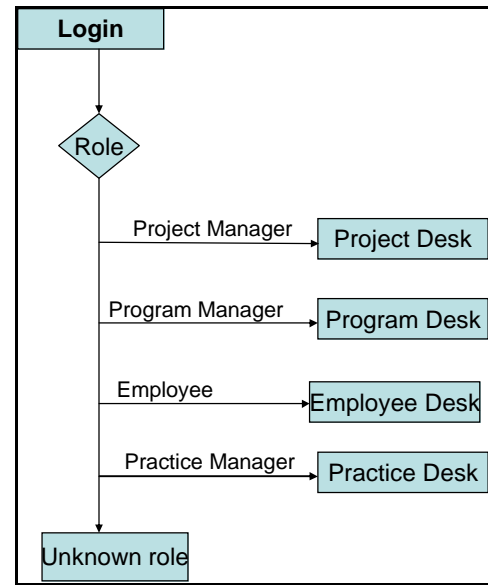


2. **Open Application Interface** – We have extended and implemented XML [11] RSS (RDF Style Sheet) [12] interface within our adapters to simplify the application interfaces. We have used ROME 0.9 [13] in our implementation. This helps in enabling interactions with other applications as well as other external frameworks and enables import/export of data as per agreed upon ontology (or vocabulary).
3. **Multiple Interfaces for each Adapter** – Our adapters support multiple interfaces to invoke the application interfaces (which could be commands, shell scripts, Application Programming Interfaces or web services). This helped us in using the same adapter framework in integrating multiple applications and in a uniform way. In addition, the adapters transform the data from common data model format to application specific format, and support notifications.
4. **Single Window User Interface System with Role based access:** This supports the following:
  - a. **User Access Interface:** Supports user login and invocation of available services. The user can customize the user interface to suit his/her needs in terms of look, feel and behavior. We have support for role based authorization of the

users. Based on the specific role (which can be configured by the administrator), a specific set of relevant workflows is made available to the user.

- b. **Self-service:** Supports self-service in which the user can configure and deploy additional user-specific configurations.

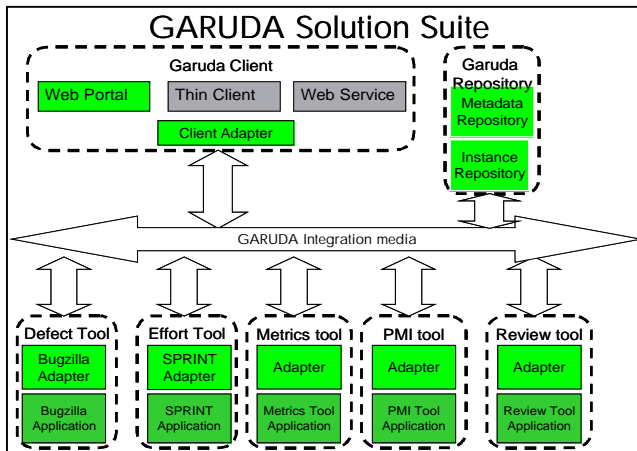
The high level view of the single window system is given below:



- c. **Service Registration:** A user can register new services using this window. The user needs to fill in certain details regarding the service and then the service is registered with meta data service (or Universal Description Discovery and Integration (UDDI) [14], and made available.
- d. **Rich Meta data Repository** – We have defined a rich vocabulary of meta data elements which have associated semantics with them. These objects are included in the ontology. The meta data repository contains information about the services and the service contracts. In addition, we have a separate instance repository of all the reports to process the requests for generating reports quickly.

## 4. Detailed Solution Architecture

The solution architecture is given in the diagram:



As we can see from the diagram, the approach consists of separating the business logic from the application logic, and providing an application independent view of data to the users.

Some of the important components of the architecture include the following:

### 1. Portal/User interface

The portal interface serves as the single window login interface for the system, and uses LDAP (Local Directory Access Protocol) protocol to validate the users. Once the users are authorized, they are given access to all the services available in the system, for which they have been authorized. For example, the project managers are given access to the project manager's desk which contains only the services that are relevant to them.

The portal interface provides the users the ability to invoke the available services. The users can view the data objects in the way that they are comfortable with (user data model), and select various services to perform various operations.

The portal interface also provides the users with an option to view and select the operations that are built over these services, which can also be invoked directly. For example, when a user requests for the generation of project management report, it results in the execution of a business process which actually invokes the various services to generate the report, and provides the report to the user.

When ever a user selects an operation to be performed, the client software chooses an appropriate transport mechanism for him, based on the preference specified in the configuration file. It is also required that when a particular service is registered with the repository, the details of the supported transport be provided. This set forms the superset of supported transport media for that particular service.

### 2. Business Process

The business process layer provides support for business processes to be developed that can control and sequence the various business operations. These business processes are triggered by the publication of a web service invocation from the portal interface. The WSDL (web service description language) specification of each service contract is available in the repository. These business processes are triggered by publishing a SOAP (Simple Object Access Protocol) [10] request over a specific transport (for example, http). The business processes are realized using Jboss Business Process Management (jbPM), as a set of workflows performing a sequence of steps.

### 3. Integration/Transport Media

The integration media provides support for the invocation of the various web service calls.

At this time, the supported transport mechanisms include http and email. The integration media is basically Hypertext Transfer Protocol (HTTP).

### 4. Adaptation

The adaptation layer provides support for transformation and mapping to support the data model transformations. The data in transit on the integration media is mapped to an application specific format for consumption by the applications.

In addition, the adaptation layer also provides support for implementing business activities. So, when ever a business activity changes, the only change required is in the adaptation layer, and the overall business process remains unchanged.

### 5. Applications

The initial set of applications consists of SPRINT, metrics tool, review tool, defect management tool and the PMI (Project Management Interface) tool. Each of these applications supports API interface which is made available as a web service. These web services are invoked by the service consumers.

Some of these applications are home grown, whereas some are COTS applications like Microsoft Excel. These applications run on diverse platforms – some run on Windows, and some run on Linux.

### 6. Data Model

A data model is a specification of the data elements of a specific application or a domain. An Application Specific Data model contains a list of all the elements of a particular application along with their attributes and the various operations that are possible on those data items.

Extensive data models are developed for the purpose of facilitating sharing of data across various subsystems in the overall solution. The shared data model is also known as the Common/Shared Data Model or Common Data Format (CDF). The shared data model represents the application neutral way of representing the data items. This enables easy integration with other applications.

The data items are transformed from the shared representation to the application specific representation and vice-versa using transformation scripts. These transformation scripts perform class to

class transformation, and they are developed using XSLT (XML Style Language Template) scripts.

### 1. Use cases

A sample of the use cases covered in our initial implementation is given in Figure 1: Use Cases:

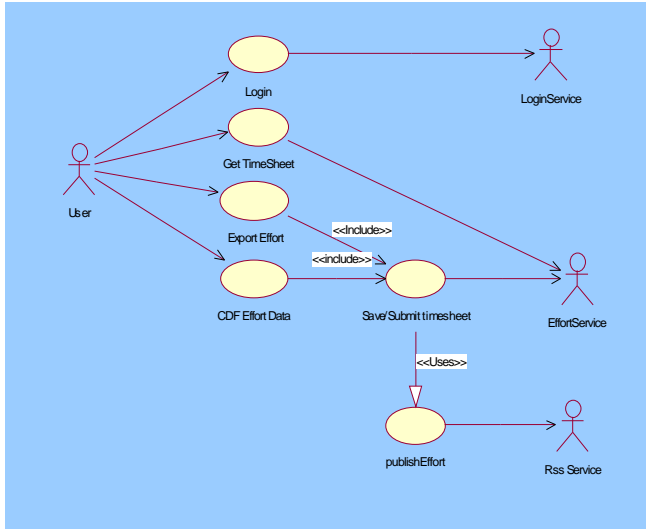


Figure 1: Use Cases

These include a login, get time sheet, save time sheet, generate PMR report, generate defect distribution and so on. Each of these use cases demonstrates a specific service within our context. For example, the get time sheet use case demonstrated invocation of the get time sheet service, which ultimately returns the time sheet for the requested time period.

### 2. Services

Some of the important services that are available include Effort service, login service, PMR service and the RSS service. The login service provides login and authentication mechanism to identify the user, and provides access to rest of the services. The effort service provides get time sheet and save time sheet operations. The RSS service provides support for RSS for integration. Each of these services is independent of the underlying application that is providing that particular feature. For example, the effort service upon invocation might actually interact with SPRINT, however, it is not tied to SPRINT. It can as well work with other products like OMEGA (which is also an effort tracking application). This is made possible by the various data models that are defined and the amount of loose coupling that is provided by the implementation in general, and SOA in particular.

### 3. RSS

RSS stands for Really Simple Syndication. In this protocol, the information required to be shared is published in a URL (Universal Resource Locator), and the application requiring this information polls this URL, and gets this information. The application that provides or generates this information is known as generator, and the application or entity that consumes this information is known as reader.

### 4. Generator

The generator creates the information and publishes onto an URL. The integration framework consolidates the requests for publication from various applications/users and gives this information to the generator. Each field of the request is mapped to a corresponding field within the RSS XML document. The link field in the RSS XML document represents the WSDL specification for the web service in our case. It is possible to configure the RSS component to store each user's data separately. The data published in the RSS file are organized based on the context, which is configurable. Also, it is possible to configure an expiry date to help identify the items to be archived.

### 5. Reader

The reader entity is responsible for fetching the new/updated data from the published RSS data file. The reader downloads the new/updated data. Once the data is retrieved, the reader invokes an appropriate web service/API to perform the action with the data. At the completion of the operation, an email message is sent to the concerned user to indicate the success/failure of the submission.

### 6. Mapping of RSS fields

RSS XML document specifies a set of fields that are usually used for giving details of the information to be shared. We have used the link field in the RSS XML to point to the SOAP payload of the request formed. The receiving end (SPRINT adapter) extracts this field, and obtains the request to process. Once the request is processed, the response is encoded as an XML file, and the pointer to the XML file is provided in the link field.

As we have adhered to the RSS XML format and mapped the link field (within item) to the payload, our implementation is compatible with any other standard RSS implementation. In other words, it is possible to re-use any existing RSS reader software as a client.

### 7. Usage of RSS for Integration

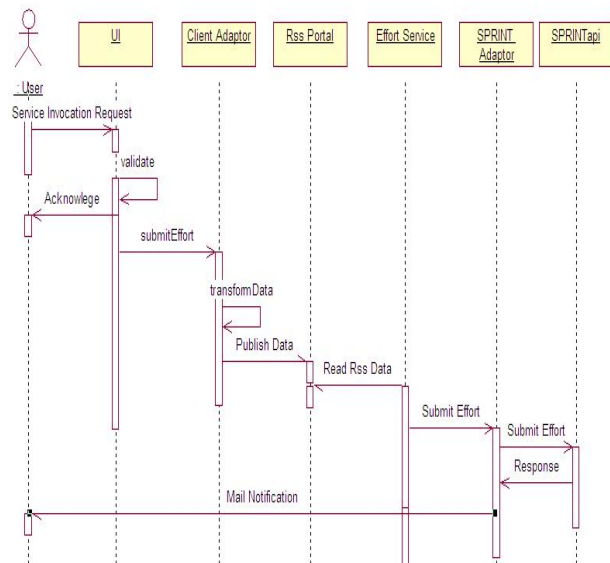
In our implementation, we have used RSS protocol to integrate the service consumer (in our case, this is the portal interface) and the service provider (in our case this is the application/adapters).

The following sequence of activities happens during this interaction:

- a. The user updates the transport protocol field to refer to RSS in the initial configuration file.
- b. Then, the user initiates an effort entry operation, and fills in the appropriate fields.
- c. Then, he submits the request for processing. The client software builds a SOAP request.
- d. The client adapter publishes the request (payload) in the RSS portal.
- e. Once this done, the application adapters (eg. SPRINT adapter) is notified that the RSS portal has changed.

- f. Immediately, the SPRINT adapter reads the data in the RSS portal and processes the request that is encapsulated in the request (which is the link field in the RSS portal).
- g. Then the application adapter invokes the appropriate web service to perform the specific task on the application (for example, to update the effort data into the SPRINT product).
- h. Once this operation is completed, the application returns the status of the operation.
- i. The adapter verifies this status, and invokes the email client to notify the client application asynchronously about the status of the requested operation.

**Sequence diagram for RSS integration**



**Figure 2: RSS Interaction Sequence Diagram**

**Solution Deployment**

The above solution has been already deployed within our company and being used extensively. We are able to extract the relevant data seamlessly across multiple products using our approach.

**Evaluation and Results**

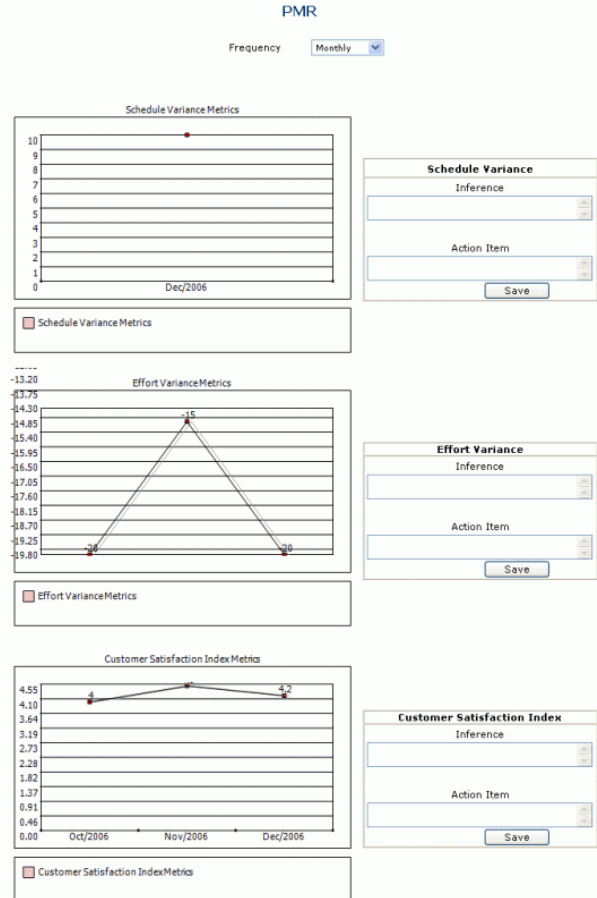
We have performed a high level evaluation of the solution and have found that it is able to address many of the challenges.

Firstly, we are able to expose more meaningful services to the end users, and thereby enable them to choose the one relevant to them. The users need not worry about the underlying applications, and they only view the services exposed by those applications. This helps the users to be goal oriented.

Secondly, we are able to link the data more easily and hence able to generate reports spanning multiple systems. In this context, consider the project management reviews which are conducted by the project managers every month, and this requires a variety of data – some of which are directly available in the tools (like the

effort, schedule, etc) and some of which need to be inferred. The current integration framework has helped us in generating these reports with ease, and without any errors.

Please see the Figure 3: High Level PMR Report for details:



**Figure 3: High Level PMR Report**

The other reports that are to be highlighted include the test defect distribution based on the actual data and the predicted defect density. A sample chart is given in Figure 4: Defect Distribution Chart

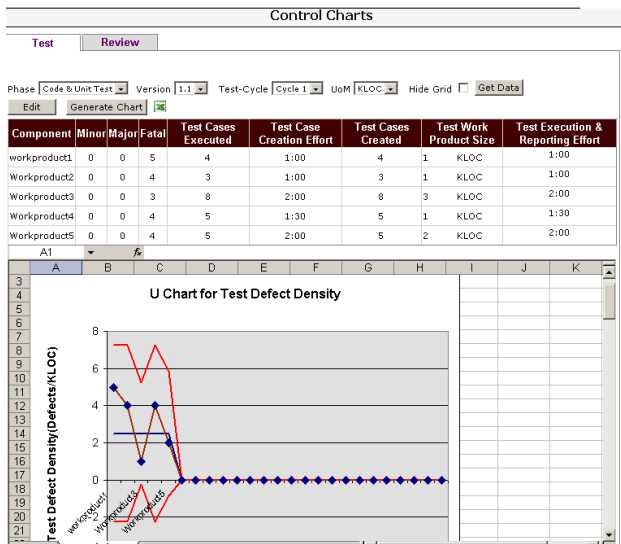


Figure 4: Defect Distribution Chart

Thirdly, we are able to reduce the number of errors that were introduced earlier, by virtue of human interactions.

Fourthly, we are able to introduce newer applications without problems, as we only need to identify and build the services that are relevant in that context, and define the mapping rules for the data elements. This has resulted in reduction in the lead time for introducing newer applications. In one case, we were able to introduce multiple versions of a defect tracking software (Bugzilla) within 2 weeks time.

Fifthly, we are able to reuse the existing custom applications along with the other applications.

Sixthly, we are able to achieve better productivity with the existing users, as they no longer need to spend huge amounts of time in collecting data and transforming data manually. This has resulted in very good productivity improvements.

Lastly, the entire system is highly reusable, and can be reused without problems. We have reused the system in other solutions as well, and this has enabled us to build newer solutions quickly.

## 5. Lessons Learnt

Based on our experience in designing and implementing the integration framework, we have learnt many useful lessons.

These are:

It is very important to involve the end users in the loop during the entire development cycle. We involved the users from the beginning, and this helped us to understand their perspectives

about the services that are planned, and thereby we were able to identify the services that were required.

One of the factors that played a critical role in the solution deployment is the usability of the solution, and as the services are planned to be used by the users directly, they need to have a good user interface.

We also found out that by pre-creating reports that were routinely being used by many users, we were able to improve the performance of the system, reduce the network load, and as well increase the user experience. This resulted in the development of the instance repository for the reports itself.

## 6. Conclusion

Application integration is a challenging task, and it is observed from our paper that following a structured approach to application integration is a key success factor. In addition, using SOA as a design pattern, and data modeling as an enabler helped us to realize a complete solution in a very short period of time. We are able to use heterogeneous technologies successfully, and integrate them seamlessly, using the principles stated in our approach.

## References

1. Data Integration Teenage Years, Alon Helavy et al, VLDB'06, Sep 12-15, 2006, Seoul, Korea
2. Enterprise Integration with ERP and EAI, Jinyoul Lee et al, Communications of the ACM, February 2003, Vol 46, No 2, pp54-60
3. Enterprise Application Integration and Complex Adaptive Systems, Jeff Sutherland et al, Communications of the ACM, October 2002, Vol 45, No 10, pp 59-64
4. Service Oriented Architectures: Approaches, technologies and research issues, Mike P Papazoglou et al, VLDB Journal, 2007, 389-415
5. The Landscape of Service Oriented Systems: A Research Perspective, Kostas Kontogiannis et al, SDSOA'07
6. The impact of SOA on Enterprise Information Architectures, Paul Patrick, SIGMOD 2005, June 14-16, 2005, Baltimore, Maryland, USA
7. <http://www.oasis-open.org>
8. <http://www.w3c.org>
9. <http://www.w3.org/TR/soap/>
10. <http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf>
11. <http://www.xml.com/pub/a/2002/12/18/dive-into-xml.html>
12. <http://www.w3.org/RDF/>
13. <http://rome.dev.java.net/>
14. <http://www.uddi.org>