Connected in a Small World: Rapid Integration of Heterogeneous Biology Resources

Umut Topkara   Carol X. Song   Jungha Woo   Sang P. Park
Rosen Center for Advanced Computing
Purdue University
West Lafayette, IN, 47907, USA
utopkara,carolxsong,wooj,sspark@purdue.edu

Abstract— **Timely access to the most up to date versions of resources, such as data and software, is of paramount importance for researchers in an active field like Biology. We introduce a grid enabled biological data and software collection portal architecture, SALSA (a Scalable Simple Architecture), that is tailored towards fast integration of new computational resources made available by ever faster advancing and diversifying research in this area.**

We identify two models that guide the design of SALSA: heterogeneous database model and network growth model with preferential attachment.

SALSA recognizes the challenges that are noted by the previous research on heterogeneous database model inherent in biological database resources; these resources are autonomously managed and lack a common database schema.

SALSA is also guided by a model for the growth of the portal's collection (of data and associated software to process this data) from previous research on related collections (e.g. citation networks and software package dependencies). This model suggests that in the presence of components that have a higher likelihood of gaining new connections (e.g., popular resources such as BLAST or FASTA sequences), the relationships between components tend to organize in a *small-world scale-free network*.

The growth model helps the portal developers identify important *hub* components that emerge by taking part in increasing number of tasks as the portal grows. In order to effectively improve the overall user experience, developers can direct expensive development efforts (e.g., query optimization, user interface, documentation, etc.) to hub components, rather than to specialized components that have a lesser likelihood of developing to become hubs.

In this paper we discuss a grid enabled web portal implementation that is built to contain a growing collection of biological data and software to process this data. The implementation that we present is a realization of *Scalable Simple Architecture* (SALSA) that strives to rapidly integrate newly published components into the existing collection in a sustainable fashion. Notably, this implementation uses flexibility of XML for component management, XSL for web user interface, SRB and MCAT for large data storage.

I. Introduction

Timely access to the newest resources is of paramount importance for researchers in an active field like Biology. Biological databases and software have grown to number and complexity that overwhelm the researchers in the area. The diversification of research areas as well as the availability of new technology to create large amounts of experimental data have contributed to this trend. Despite the fact that most of these data and software are publicly available through the internet, the heterogeneous nature of these resources may hinder their ease of use and immediate accessibility by researchers, hence may inhibit the pace of scientific discoveries.

In this paper we discuss a grid enabled web portal implementation that is built to contain a growing collection of biological data and software to process this data. The implementation that we present is a realization of *Scalable Simple Architecture* (SALSA) that strives to rapidly integrate newly published components into the existing collection in a sustainable fashion.

SALSA recognizes the challenges that are noted by the previous research on heterogeneous database model inherent in computational resources in Biology; these resources are autonomously developed and they lack a coordination of design (e.g., common database schema). The heterogeneity of the components in the collection denies presumptions about data format, data schemas, input parameters, and user interfaces. For this reason, SALSA does not use one canonical component model to capture all possible components. Even though this design decision brings flexibility to precisely define the components of the collection, it also comes with an added cost of complexity due to the multiplicity of incompatible component definitions. We show that the growth of the collection can be sustained despite this potential for enormous complexity (more on this later).

One of the important issues with biological resources is the intermittent availability of resources. SALSA mandates the implementation of a cache for query results when they are obtained from volatile remote resources, so that they can be re-used by processes that are fired by later jobs. SALSA also incorporates a powerful documentation facility which associates all data and software components with their sources and versions; this mitigates conflicts due to uncoordinated data and schema updates to data sources.

SALSA is also guided by a model for the growth of the portal's collection based on previous research about similar collections. This model suggests that, in a growing network, if the new components have a slight preference to establish relationships with some of the existing components, the relationships between components tend to organize in a *small-*

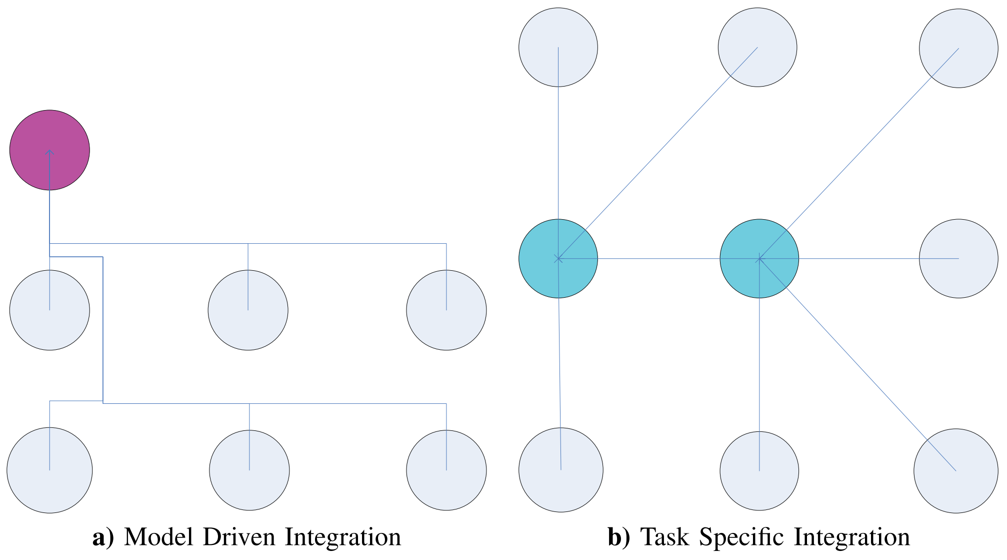

a) Model Driven Integration
b) Task Specific Integration

Fig. 1. Approaches to integration. Edges indicate a compatibility among incident components. **a)** The darker node is the canonical component model that captures all instance components, thereby making all components compatible with each other. **b)** The darker nodes are popular application or data components -hubs- which take part in many task specific workflows; other components take part in only a limited number of workflows.

world scale-free network. In a scale-free network, most of the nodes have a small number of connections with other nodes, whereas a small number of nodes are connected to a large number of components. The latter type of nodes are called *hub*s.

This growth model helps the portal developers identify important hub components (e.g., BLAST, FASTA data, etc.) that emerge by taking part in an increasing number of tasks while the portal grows. In order to effectively improve the overall user experience, developers can direct expensive development efforts (e.g., query optimization, user interface, documentation, etc.) to hub components, rather than to specialized components that have a lesser likelihood of developing to become hubs. Workflows that span many components are limited in SALSA by design, whereas tasks oriented workflows that involve a small number of hub components are abundant (more on this later).

The small world model enables SALSA to achieve increased robustness against the high fault rate of heterogeneous components. SALSA achieves this by deploying multiple components that achieve the same tasks with emphasis on providing a redundancy of hub components (which also include service middleware components such as Storage, and Execute). Since the robustness of these components have a disproportional effect on the overall robustness of the system – due to their central role among components– we seek to fortify these components with their alternatives, such as mirror sites, local copies, version repositories and competing applications.

In the next section we discuss the main concepts of SALSA along with its comparison to model-driven architectures. Then, in Section III, we will give a brief overview of a biology resource collection implementation using SALSA concepts. We conclude our paper with review of related literature and a discussion of future work.

II. Thinking Rapid Integration

SALSA's foremost design criterion is the ability to easily integrate new research-grade resources into a collection. This criterion is coherent with the imperative for the access of most up to date data and software in Biology research.

Before moving on to discussion of SALSA, we would like to give a brief overview of the popular alternative to integrating research grade resources. To the best of our knowledge, model-driven middleware architectures [1], [2], [3] struggle to seamlessly integrate data services and workflow enactment services by wrapping their components in uniform component models which interact through interfaces defined by these models (Figure 1.**a**). These architectures are suitable for Model Driven Development processes, which start with a blueprint of the final product and then proceed to realize the final product using formal methods and Computer Aided Software Engineering tools [4]. The effectiveness of the overall Model Driven Development process depends heavily on whether the requirements can be precisely obtained during the design phase, otherwise revisions in requirements may stifle the development [5], [6], [7]. Hence, in grid enabled portals Model Driven Development can be expected to be more effective when the implementation of portal components are coordinated, and the canonical component models are powerful enough to capture all possible components without sacrificing from precision. These architectures have a significant advantage if all components have the potential to be used by every other component: if there are N different data or software components, the integration of the overall system requires the development of only N component model instances from which wrappers or mediators can be automatically generated. This number is vastly smaller than $O(N^2)$ mediators that would need to be developed if each components pairing would need to be supported (forming a complete graph) without a uniform interface and automation.

Unfortunately the heterogeneous nature of research resources introduces some important problems for this approach.

A major drawback of the model-driven approach, which arises at design time, is that they have to capture all requirements as well as anticipated behavior of all the components in their canonical component models or schemas. Therefore, the development of the schemas for such systems requires in-depth knowledge of the underlying research fields by the developers.

Another more important drawback emerges following the deployment, while the model-driven systems grow. The initial schemas which capture the canonical components that have been solidified at the design time may become insufficient to capture a precise description of the new components as the collection acquires components from new fields or as the components in the existing research areas diversify with the natural development within research fields. As a result, the schemas need to be revised so that description of the canonical components capture the specific requirements of all these new components. Unfortunately, these revised schemas are usually more lengthy and more complex, thereby make the addition of new components to the collection a more tedious process [5], [8], [9], [10]. Eventually, it requires more time for each new component to be integrated into the system, which may add up to cripple the growth and expansion of the collection portal.

SALSA's development process anticipates the design requirements may frequently change in a grid portal that serves access to a growing number of workflows which consist of third party software and data. Hence, in contrast to Model Driven Development, SALSA employs an Agile Development [7], [10], [9], [5], [11], [12] process which provide the design flexibility to enable continuous maintenance and improvement of the portal.

In the next two sections we discuss two models that support SALSA's approach to scalable integration of biology resources into a software and database web portal: the heterogeneous database model for the nature of data, and the complex network model for the growth of the collection. Note that these models are different from generative models in Model Driven Architecture; since they do not describe *what* the final product is, but *how* the system will evolve. Then, we summarize SALSA design concepts in Section II-C.

A. Heterogeneous Data Model

Resources for computational biology research is accumulated in a rather uncoordinated process that involves a large body of researchers who are distributed to all over the World. Even though the internet is making the communication among these researchers easier, group collaborations are far from creating a coordinated research effort. Individual laboratories or researchers take design decisions while developing their data and software without any outside control. The resulting multiplicity of data resources, formats, applications, implementations, etc. are modeled in *heterogeneous data model* [13], [14], [15].

Heterogeneous data has the following properties:

- Locally controlled updates: The resources are managed by autonomous researchers, and they make syntactic and semantic changes to the resources they provide without prior notice. Following such remote updates, applications that depend on these resources may fail to operate correctly.
- Sporadic down times: Since these portals are independently maintained, there can be computer failures or scheduled maintenances that make the resources temporarily unavailable.
- Heterogeneity in Size: Resources that are provided in remote repositories can vary in size and quality. In order to effectively use several data sources, researchers will need to combine large data with small data, while paying attention to contamination of high-quality data with low quality data.
- Structural Heterogeneity: The data resources vary in their structure from highly structured relational databases to unstructured text. An effective system can process data that is represented in several types of structures.
- Heterogeneity in querying abilities: Some remote resources, or the applications in the portal (which can be viewed as queries on the data) might allow only a limited set of queries.

The aforementioned properties of heterogeneous resources account as drawbacks for model-driven approaches to integrate them. This is because model-driven approach assumes that all programs and files in the portal's collection can be represented with a canonical description that is written using their arsenal of descriptors. This assumption does not hold for biology resources, which are created by independent researchers from diverse fields to fit their individual research needs and practices - hence are inherently heterogeneous. A canonical description would either be too general to let individual programs and files to be precisely documented, or it would be too complex to the point that it will require the understanding of all the programs and data in the collection to be able to understand the descriptors and add a new component to the collection.

B. Portal Growth Model

In this paper we conjecture that, the architecture of a growing collection of biological resources does not manifest $O(N^2)$ interactions (complete graph) among its components, but only a small fraction of these interactions among components will be realized. We base our argument on the recently recognized small-world scale-free network characteristic in software components [16], [17], [18], [19], [20], [21] and citations [22], [23].

In scale-free networks [24], [25], the connectivity of nodes does not follow an even distribution, which would be expected from a random graph. Some nodes have significantly higher number of connections (hub nodes) and most of the nodes are connected to a few hub nodes with a small number of edges. The probability that a random node will connect to k other nodes, $P(k)$, is roughly $k^{-\alpha}$, where α is a constant that characterizes the network. This peculiar behavior of

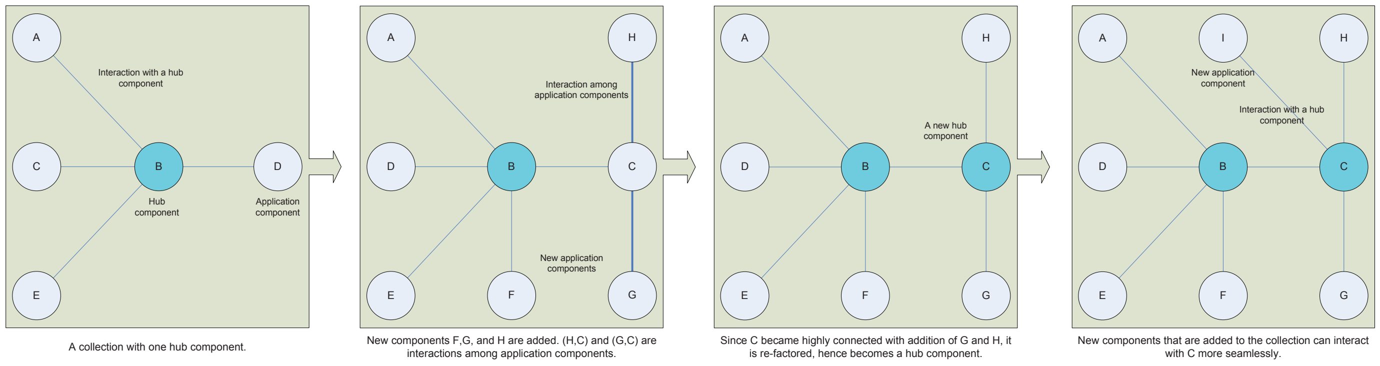

A
Interaction with a hub component
C
B
D
Hub component
Application component
E
A collection with one hub component.
A
H
Interaction among application components
D
B
C
New application components
E
F
G
New components F,G, and H are added. (H,C) and (G,C) are interactions among application components.
A
H
A new hub component
D
B
C
E
F
G
Since C became highly connected with addition of G and H, it is re-factored, hence becomes a hub component.
A
I
H
New application component
Interaction with a hub component
D
B
C
E
F
G
New components that are added to the collection can interact with C more seamlessly.

Fig. 2. The maintenance of SALSA during its growth.

networks have been explained using a generative mechanism, called "preferential attachment". According to this generative model, if some of the nodes have a higher likelihood of gaining connections from newly introduced nodes, the network becomes scale-free as it grows.

In the context of a grid portal, this model means that, most of the components will need to be compatible with only a few hub components, such as the service components (e.g., the generic storage, computation, visualization components) or common data formats (e.g., FASTA sequences, molecular interaction network graphs, etc.) and common software (e.g., BLAST, graph clustering, etc.). The interactions among non-generic components can be achieved by using only a small number of interaction-specific mediators (Figure 1.**b**).

The Debian package dependency network is an example of scale free networks that manifest themselves in software collections [18]: more than half of the packages are not referenced by other packages (skewed distribution of connectivity), and three quarter of the packages need other packages. The dependency graph is sparse, but in the graph of about 20 thousand components the average distance among components is just over 3 dependency links. The distribution of the number of incoming dependencies is $P(k) = k^{-0.90}$.

C. SALSA Framework

In a grid enabled software and data portal it is very important that the portal development process is sustainable to embrace changes to requirements of existing components and additions of components with new requirements. Large software systems usually need to undergo small design changes during their product lifecycles, however such changes are more pronounced in the life of a portal. This is mostly because the development of individual components that are part of the portal are results of an uncoordinated effort of third-party developers, and there is a continuing pressure to add more of these components to the portal. It is imperative to use development methods which can support software that evolve through continuous improvement in the design and code structure.

In recent years, Agile Development Methods have been proposed and successfully used in development of large software systems [7], [10], [9], [5], [11], [12]. These methods recognize the fact that adding features under a previously planned framework is costlier than adding without any previous irreversible design decisions. For this reason, they emphasize a continuous process of specification, design and development. The specifications are usually based on the use scenarios, and the design are least specific. The existing code is first improved to accommodate the new design using *refactoring* [11] and then new features are added. During refactoring components that perform specific tasks are converted into instantiations of more generic components; however such changes to the code are not made until they are required by the latest specification changes.

In a grid enabled portal, management of such continuous effort can be a daunting task unless it is aided by an architecture that successfully captures the evolution of the portal and helps predict the future changes to the system. SALSA architecture redirects focus from the many components that are specific to a single application (e.g., configuration input file of a program) to the few generic components (e.g., storage, execute, history and user interface components) as well as commonly used application components, which we will refer as hub applications (e.g., FASTA databases, molecular interaction databases, BLAST, etc.). The generic components and the hub applications provide common services to a large number of components in the system. The stability and maintainability of these few generic components and hub applications are critical for the evolution and growth of the collection, since the new components that are added to the system tend to depend on them.

In SALSA, new hubs that emerge in the interaction graph are converted into generic component models through refactoring (Figure 2). Refactoring process of a portal component involves optimization for improved performance, as well as the creation of a canonical model specific to the hub node that also describes the components similar to it (e.g., BLAST is refactored to be integrated into a more general pattern matching component). Since the new hubs will attract new interactions, this rare maintenance process will reduce the number of mediators required for new interactions added to the system as well as the complexity of adding these new mediators. This approach differs from the model-driven approach, in several ways. Firstly, the canonical descriptions for hub resources evolve over time, instead of being predicted at design time. Also, the canonical descriptions pertain to a very specific type of applications characterized by a hub application, rather than striving to capture the characteristics of all applications. This latter difference translates into a simpler integration process for new applications in SALSA; because every application can be defined independently from the constraints of conforming to a canonical descriptor set, which could otherwise grow to sizes that can impede this process. After enough mediators are added to the existing network, the interaction among the components will be stable, i.e., new workflows that involve existing components will not require new mediators and new mediators will only be needed for newly added components.

Focusing of the development effort (including documentation, maintenance, improvement of interface and performance) to a component translates to the flexibility of this component so that it is easier for new task workflows to use this component. Moreover, the improvement of these components directly increase the quality of the overall user experience, hence it is an effective practice for the use of the expensive programming effort.

SALSA enables the system to achieve more stability in a way that is hard to achieve without recognizing the growth model that underlies it: The generic components can provide redundancy to increase the overall error-tolerance of the system. Redundancy is achieved by providing a specific service through different implementations. For example, the storage service component provides operations of data store and retrieval using a local file system as well as SRB (Storage Service Broker) for distributed storage. If one component fails, an alternative component is still available. Since the failure of the overall system in a scale free network topology mostly depends on the failure of hubs, the error-tolerance at hubs has a disproportional contribution to the overall robustness of the system. The identification of hub applications, which are also central to the collection, also makes it possible to extend redundancy to these hub applications. As added benefits, the redundant alternatives for hub applications provide a better user experience (by providing alternative tools which can be used to control results obtained from one tool) and help possible problems that have been discussed in the context of heterogeneous data model (e.g., unavailable remote data, incompatibility of new formats, etc.).

We propose a web service-oriented architecture that recognizes these results to provide rapid prototyping by requiring minimal overhead to integrate new data and processing components to the system.

III. Implementation

We have used the SALSA concepts to implement a grid-enabled web portal for collection of research grade computational biology resources. In this section we discuss the overall architecture of this portal, and details that involve the component descriptions and data management.

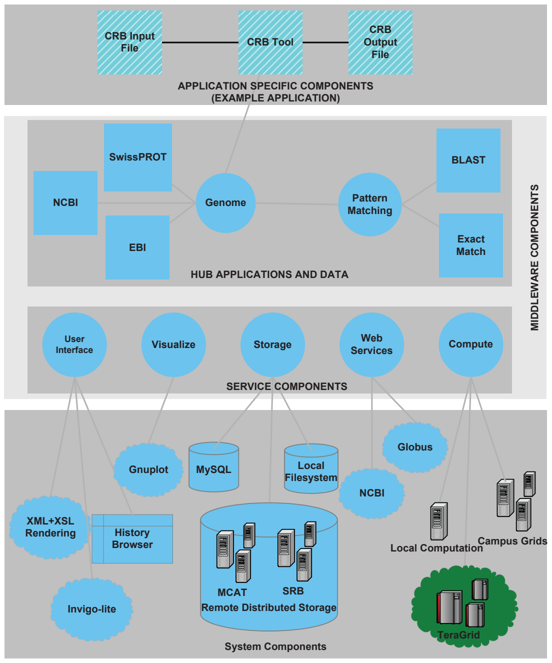


Fig. 3. The component hierarchy of SALSA.

SALSA portal is a web based collection, and its aim is to grow into a large collection of up to date biology data and software. SALSA has a small number of service middleware components, which are used by the rest of the system (Figure 3): User Interface, Storage, Compute, Web Services Interface, Visualize. In addition to these service components, there are application based middleware components, these components are services that have been derived from individual applications or data that have proved to be central to the collection. Examples of such application based middleware are pattern matching component derived from BLAST, and sequence data component derived from FASTA and SwissProt.

As we have discussed earlier, redundancy has tremendous benefits for fault tolerance of the system. Since our components are from 3rd party sources, we need to be prepared for failure of these sources, redundancy is one of the ways we use (replication of data by caching, alternative data sources from other mirrors, also for software and central components). Redundancy at Execute component is achieved by using local computing resources as well as campus grids and TeraGrid through Condor.

Storage is a significant concern when the remote data resources are not persistent, as they can be unavailable, have limited accessibility or can be frequently updated. We used several types of storage targets to meet these needs that range from local flat file storage to mySQL and to SRB.

In our implementation, web applications that interact with the user are based on the GridSphere portlet framework. GridSphere provides ease for common web development tasks, such as user profiles, login management and customized layouts. In order to meet the flexibility that is required to provide user interface for a variety of applications we augmented GridSphere portlets with an XML and XSL based interface.

In the following subsections we discuss issues relating to user interface, storage, and compute services in a grid enabled portal. We first discuss how XML can be used to manage component descriptors, submitted jobs and user interaction at the same time. Note that these are service middleware components, which will be used by almost all components in the portal.

A. Multi-purpose XML

In a grid enabled portal, it is necessary to have a specification of each component for the dual purpose of human-readable documentation and machine-readable book-keeping. Human-readable documentation include annotations by developers during the design and development of these components, as well as annotations of user generated data or submitted jobs by end users. Machine-readable specifications include all information that are automatically processed to direct the execution of the portal, such as input-output types, CPU requirements, and remote data locations. Even though developers may directly alter this information, end users usually need more user-friendly interfaces while they dispatch new jobs or add annotations. Implementations of user interfaces for web applications need to replicate and be compatible with the descriptions of their respective components. The alias relationship between the component specifications and their user interfaces may hinder the maintenance of the portal, which is anticipated to evolve as it grows. In order to facilitate a more streamlined maintenance and development process we have coupled the user interface specification with the component specification, such that a change in a component's specification are automatically reflected in the user interface. In this section we discuss the details of the component specification and user interface aspects of our portal implementation.

XML is a simple but powerful markup language for text data. It also enables the use of XSL to easily generate user interfaces to present and edit the underlying data [26]. The widely used web browsers such as Mozilla and Internet Explorer have built-in support for rendering XML to HTML through XSLT.

In our implementation, we used XML to document data and software components, as well as to keep track of the execution trail of workflows. We used XSL to build user interface for these components and workflows. This feature of XML allows us to couple component data and their user interface while efficiently separating the user interface development effort from that of portal development.

We describe each data and software component in our portal with a specific XML. The XML description for computational components include information such as the input data requirements, the specifications of output data, the hardware requirements as well as annotations by the user. Likewise the description of data components include identification information such as source, format, and date, as well as user annotations. The XML for these components are tailored to precisely describe their respective component, without subscribing to a common vocabulary of XML elements.

We used XSL to implement the behavior of the web based user interface for a component. Besides the separating the user interface development from the rest of the portal development effort, this choice has several other advantages. Firstly, the user interfaces are generated in the client side and there is no need for the use of any computational power at the server side to generate HTML-based user interfaces. Secondly, the user interface development is relieved from the burden of supporting variety of user platforms (e.g. mobile, text based, etc.), since the browser will take care of parsing XML and generating the HTML result; the only compatibility requirement is the use of a browser that can render XML files. XSL had been our choice as a user interface markup language because of its cross-platform compatibility advantage over alternatives such as XUL, MXML, and XAML [27]. Even though these alternative technologies offer advanced user interface libraries that support menu bars, tree views and tab boxes [28], XSL still has a good standing since such functionalities are attainable with the help of Javascript.

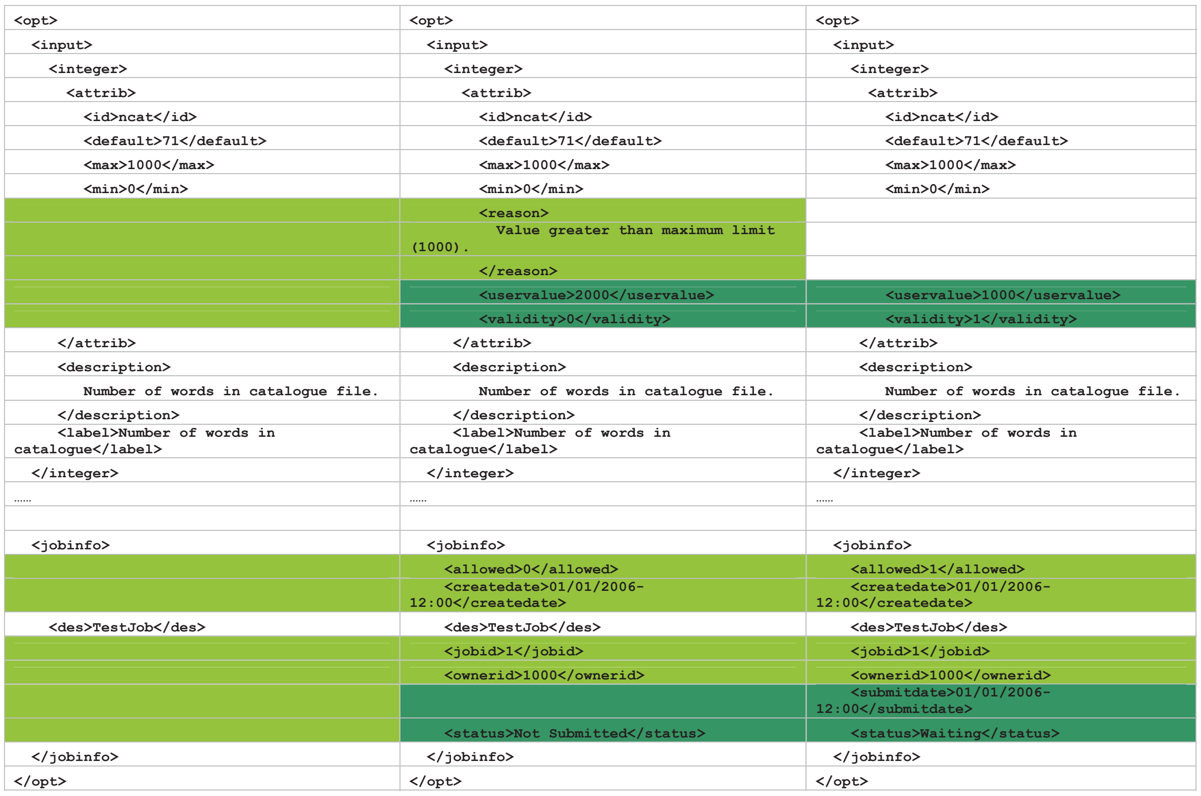
The use of XSLT enables the deployment of Javascript for a more useable and interactive user interface that goes beyond simple HTML forms. Currently, we use Javascript to verify the inputs at the entry time so that we can keep semantic integrity of the data and avoid execution failure due to invalid inputs. When a user tries to launch an application through job submission portlet, the portlet shows an HTML page containing all required attributes to be filled in. At this stage, Javascript checks whether the entered values are valid for the expected data type. Javascript is also useful for interactive browsing of user initiated jobs and their output files through interactive graphs.

After a user runs an application, the XML description that is used to describe the application type is updated with the input data that has been entered by the user. This updated XML becomes an instance of a job of the corresponding application type (see middle column of Figure 4). The same XML is then updated by the Execute service component to record its trail until it finishes its execution (see right column in Figure 4). The history data for each execution can be easily rendered to be shown in the browser with another XSL which is very similar to the XSL which was used for user input interface, (see Figure 5).

XML and XSL help rapid integration of a new application into the portal and evolution of existing components that are already in the portal. We would like to emphasize that one of the biggest advantages of using XML is its ability to provide powerful and customized web based user interfaces, while requiring little development and maintenance effort.

B. SRB, MCAT and SQL

SRB is a middleware that uses a hierarchical logical namespace to manage distributed and heterogeneous data collections. Among data grid middleware, SRB is commonly accepted



<opt>	<opt>	<opt>
<input>	<input>	<input>
<integer>	<integer>	<integer>
<attrib>	<attrib>	<attrib>
<id>ncat</id>	<id>ncat</id>	<id>ncat</id>
<default>71</default>	<default>71</default>	<default>71</default>
<max>1000</max>	<max>1000</max>	<max>1000</max>
<min>0</min>	<min>0</min>	<min>0</min>
	<reason>	
	Value greater than maximum limit (1000).	
	</reason>	
	<uservalue>2000</uservalue>	<uservalue>1000</uservalue>
	<validity>0</validity>	<validity>1</validity>
</attrib>	</attrib>	</attrib>
<description>	<description>	<description>
Number of words in catalogue file.	Number of words in catalogue file.	Number of words in catalogue file.
</description>	</description>	</description>
<label>Number of words in catalogue</label>	<label>Number of words in catalogue</label>	<label>Number of words in catalogue</label>
</integer>	</integer>	</integer>
……	……	……
<jobinfo>	<jobinfo>	<jobinfo>
	<allowed>0</allowed>	<allowed>1</allowed>
	<createdate>01/01/2006-12:00</createdate>	<createdate>01/01/2006-12:00</createdate>
<des>TestJob</des>	<des>TestJob</des>	<des>TestJob</des>
	<jobid>1</jobid>	<jobid>1</jobid>
	<ownerid>1000</ownerid>	<ownerid>1000</ownerid>
		<submitdate>01/01/2006-12:00</submitdate>
	<status>Not Submitted</status>	<status>Waiting</status>
</jobinfo>	</jobinfo>	</jobinfo>
</opt>	</opt>	</opt>

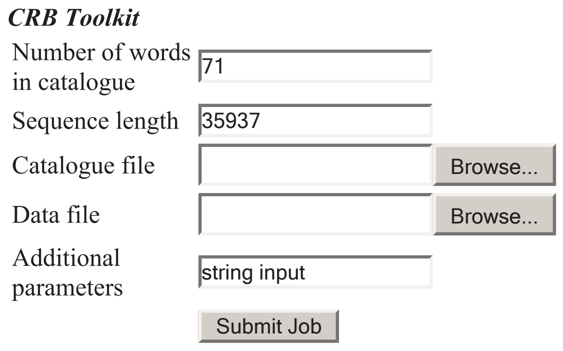
Fig. 4. The lifetime of XML file in SALSA.

to be the most popular and mature of its kind. It supports file management through logical data space, scalable capacity, security support, replication, federation, and APIs for development. SRB supports bulk-operations and data grid federations, both of which are essential for the scalability. SRB can be used with Globus Toolkit and Condor, the commonly used grid tools today. SRB has its weaknesses, nonetheless. For example, it is heavily dependent on Metadata Catalog (MCAT). If the MCAT server becomes unavailable, it is not possible to retrieve data even though the data server is operational. We have found that SRB can readily be integrated into our implementation and serve as our main file storage method. Its Java API (JARGON) is easy to use, making up for the shortcoming of not being based on standards.

SRB has proved to be a good choice for our data grid component since the biology research applications we support are all data-intensive computations and need to access large amount of data. The choice of SRB and MCAT simplified the development process as our system takes advantage of SRB's capability of handling unlimited storage space and automated data grid management. We use SRB to store not only the output data produced by the computations, but also the execution history and user input. Execution history data is also stored as metadata in the MCAT. This enables the user to examine completed jobs, modify the input parameters if he/she wishes, and re-execute the application with new inputs. Storing job history metadata also allows for flexibly searches and speedy retrieval of job information.

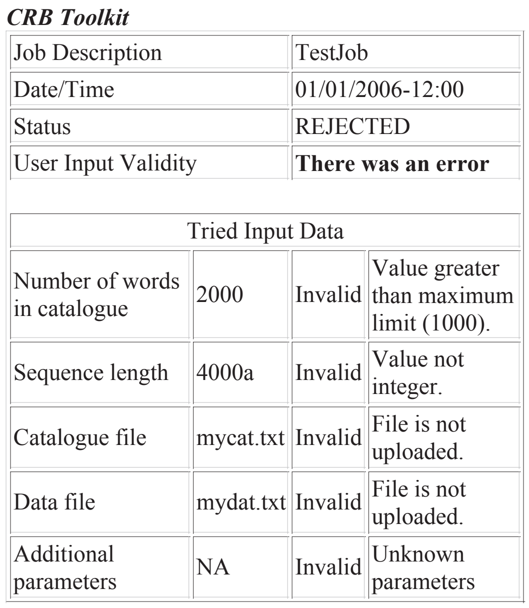
In our data experiments, we use MCAT to store metadata that describe large data collections. SRB retrieval latency remains the same regardless of the size of the data collections since unique identifiers are used to access the files stored in SRB. However, the MCAT access latency increases proportionally when size of the data collection (number of files) expands. This is because query operations are used to access MCAT data and they are sensitive to the size of data, which in our experiment was in the order of hundreds of thousands.

In order to alleviate the slow response time from the MCAT, we use a method of information replication to achieve faster response for some important types of the queries. In addition to MCAT, we use MySQL to manage a local database for storing metadata that pertain to job management, which is typically a small amount of data. With the replication, a portlet page containing the job history of a user can be rendered faster without being affected by the latency SRB data fetching or MCAT metadata query processing. This is particularly useful when MCAT is not available, because the job information database serves as a local copy of the most of the information that has been stored in MCAT. On the other hand, the XML descriptions of the jobs stored in SRB can be used to reconstruct the local MySQL server in the case of its failure. The XML descriptions stored in SRB contain a complete trace of each job launched from the portal, starting from their instantiation at the User Interface component until they are processed by


CRB Toolkit

Number of words in catalogue	71
Sequence length	35937
Catalogue file	Browse...
Data file	Browse...
Additional parameters	string input

a)Job submission interface


CRB Toolkit

Job Description	TestJob
Date/Time	01/01/2006-12:00
Status	REJECTED
User Input Validity	**There was an error**

Tried Input Data			
Number of words in catalogue	2000	Invalid	Value greater than maximum limit (1000).
Sequence length	4000a	Invalid	Value not integer.
Catalogue file	mycat.txt	Invalid	File is not uploaded.
Data file	mydat.txt	Invalid	File is not uploaded.
Additional parameters	NA	Invalid	Unknown parameters

b)Job status interface.

Fig. 5. Rendering of two XML files from Figure 4 generated by Firefox 1.5. This job has been rejected by the secondary validity check at job execution component; the user sent illegal parameters bypassing the primary validity check that uses javascript at the browser – probably by disabling scripting. Note that input validity check is required at server side for security reasons too, as the user input can be used to launch a code injection attack.

the Execute component and completed with results. Note that SRB, and MySQL are alternatives that provide redundancy for the Storage component; this redundancy alleviates reliability concerns by improving the Storage components fault tolerance. In summary, a little data redundancy provides faster query time and increase reliability of our systems.

C. Compute Service Component

In the Compute service component, we create interfaces to allow jobs to run at various computation resources, including on the local computer, campus grid, and the TeraGrid. This implementation provides the flexibility for users to compute on the resource that is most efficient for the task at hand. On the Purdue campus grid, we utilize the Purdue's high throughput computing Condor pool. It is one of the largest academic Condor sites in the U.S., with over 4200 computers distributed on the campus. Jobs that can generate a large number of independent, self-contained tasks, such as many computational biology tools, are suitable for running on a Condor pool. For example, the CRB tool [29] is a great candidate utilizing the Condor resources. The CRB tool implements a Bayesian model for locating regulatory regions in a DNA sequence. To process a long DNA sequence, we have improved the tool to process segments of the sequence simultaneously, creating multiple Condor jobs to run in parallel [30].

The Compute component allows job submission to the TeraGrid by using the Condor-G interface through TeraGrid gatekeeper computers. This enables SALSA portal users with highly parallel computations transparent access to the nation's powerful supercomputing power through a web portal.

The graph in Figure 6 shows the Condor utilization of idle CPU cycles in the Purdue Condor pool. The green area represents the Condor cycles used during a typical week.

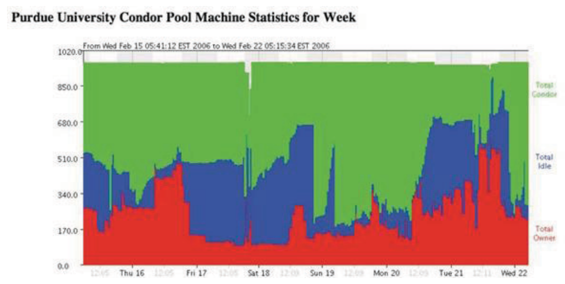


Fig. 6. Condor utilization of idle CPU cycles in the Purdue Condor pool. The green area represents the Condor cycles used during a typical week.

IV. Related Work

Integration of research grade data and software resources for Biology has been an active research area, since it will improve the research productivity and speed in this area. It has been repeatedly emphasized that the heterogeneous nature of resources is one of the biggest obstacles before such an ambitious aim.

In [13], Davidson et.al. identified the challenges in integration of biological databases: i) heterogeneous data (e.g., some relational databases, some hyper-linked documents, some text) and application programs which can be viewed as queries on data. ii) users wish to perform "bulk" queries. iii) updates to data are locally and autonomously controlled. They also studied the integration steps: 1) Data model transformation 2) Semantic schema matching 3) Schema integration 4) Data transformation 5) Semantic data matching. This early paper, demonstrates that integration of data from two independent sources is an involved process that is even harder to generalize for many sources.

Sujansky gives an introduction to database heterogeneity and its specific examples for biomedicine in [14]. Notably, the author proposes that single data model for biomedicine is not a possible goal. Sujansky also gives a list of properties that are specific to heterogeneous data integration: i) complex declarative queries should be handled; ii) heterogeneity of underlying data should be transparent to the user; iii) write access is not required from remote databases; iv)autonomous updates of databases and schemas is to be expected; v) timely access to newest data is paramount. Finally, the author provides a categorization for the types of heterogeneity: structural; naming; semantic; content. In our system, we maintain the understanding of heterogeneity as an inherent characteristic of computational biology resources, and prepare our framework for possible drawbacks this will cause.

North Carolina Bioportal (NCBioportal) [3] use PISE as a user interface middleware component for a model-driven architecture. PISE supports a comprehensive set of pre-defined data types and generates an HTML page after parsing the user specification document that has been generated using these pre-defined data types. However, it is still possible that some of the applications will require data types different from the ones provided by PISE. Our XML and XSL based user interfaces can provide customized services to such applications. This approach is good for achieving interoperability and getting fast component integration but provides limited freedom for the customization for non-standard data which is very common in heterogeneous settings.

Although Virtual Data System (VDS) was first introduced in 2002, difficulty of installing VDS has been one of the major hindrances for getting popularity, until it was made available over the web [31]. Basic idea of Virtual Data System is to re-use function definitions and real function calls. If a query procedure has been written already and saved in a database, users who want the same or similar function can take advantage of existing ones. If the users can find a function definition or real call, they only need to adjust input parameters. The re-use of functions can decrease query time of repeated or similar request. VDS also supports object history for ensuring the object semantic security. This helps users verify the safety of objects they want to re-use.

The importance of continous development in software with evidence from the complex network structure of large software systems has been reported by Myers [21]. Myers measured various characteristics of Class Collaboration Diagrams in large software and suggested a generative model of software evolution that tries to capture these characteristics. The systems that were studied by Myers' are results of a coordinated effort and design, hence some of the results do not apply to grid enabled portals in which third-party components are plentiful. Notably, Myers observes that the complexity in software systems is not responsible for fault tolerance as opposed to many complex engineering systems. However, in this respect grid enabled portals that are the focus of our paper behave closer to complex engineering systems and gain fault tolerance throgh redundancy as well as degeneracy.

Grid enabled portals are similar to Commercial Off-The-Shelf (COTS) based systems, since they rely heavily on third-party components. In this respect, COTS-based development experience can be used to improve portal development processes. On the other hand COTS-based system development is very different from grid enabled portal development, because i) the third-party components are usually of research-quality rather than commercial quality, ii) aggregation of all competing components into the system is required as opposed to selection of one. It has been repeatedly reported that the true cost of COTS-based systems lies at the maintenance [32], [33] of the system, since such systems need to update their code in order to improve their overall quality as well as to comply with new versions of the COTS components. The cost of maintenance of the COTS-based system are determined [32] by i) number of COTS packages that need to be synchronized, ii) technology refresh cycle times, iii) maintenance workload for wrapper updates, iv) reconfiguration of packages, v) product evaluation, vi) update databases, v) migrate to new standards vi) license costs It is for this reason that the managers of COTS-based systems value the functionality and reliability of COTS components more than their financial affordability at the initial deployment [33].

V. Conclusion

We have presented an architecture for a grid-enabled web-based collection of research-grade resources. We based our design decisions on two models that capture the nature of the individual resources (i.e. heterogeneous database model) and the nature of the collection (i.e. small-world scale-free network model). We diverged from the model-driven approach to adopt an application driven approach, in which individual applications are treated as separate applications from the collection, until -in rare cases- they become hub components. The flexibility in handling of individual components translate into their faster integration into the collection. We conjecture that the lazy approach to building the intermediate links between applications and data does not incur a large development complexity penalty due to the small-world scale-free model of the collection. This model states that only a small fraction of the overall components will be used by other components, hence would need to be written according to a canonical component description.

The main contributions of SALSA can be listed as follows:

- Emphasis on rapid integration of new components to the portal through a judicious management of system maintenance.
- More focus on highly connected components, without being constrained by a canonical component model. After refactorization, hub components become more compatible and can be easily added to a large number of workflows.
- Ability to grasp the trends in the requirements of a research field by observing the emergence of workflow patterns in the collection.
- Less connected components are still available in standalone workflows. This enables an early start to collect experience for future support of potentially important components.
- A simple component description method that is specific to the relevant tasks, which enables easier integration of heterogeneous resources and at the same time maintains the simplicity of this process.

- Using added redundancy for hub components to alleviate reliability issues that are inherited from heterogeneous nature of the sources of components.
- Better user experience with effective improvement of more important hub components.

There are strong indications for the small-world scale-free nature of independently developed resources. As we have demonstrated in this paper, such models have great impact on our understanding of these collections and how we design our systems. As future work we would like to quantify the exact parameters of the scale-free nature of such networks for computational biology.

Implementation of SALSA is an inherently continuing process. One of the main goals of SALSA is to be able to interchange hub components with alternatives to increase fault tolerance. In this paper we have shown an example of this for Execute and Storage services. In the future, User Interface services can be made richer using alternative approaches that could be suitable for a larger user base when combined, such as Mobile Java Applets or Ajax.

Caching is another way to achieve fault tolerance, besides increasing overall performance. However, the sheer size of a unit of computational biology data record makes traditional caching techniques insufficient. This calls for some interesting research that explore the trade-off between disk-space and reliability, rather than speed.

VI. Acknowledgments

Authors would like to the anonymous referees for their helpful comments and suggestions.

This work was supported by the National Science Foundation under TeraGrid Resource Partners grant OCI-0503992.

References

[1] X. Zhang and G. Agrawal, "Enabling Information Integration and Workflows in a Grid Environment with Automatic Wrapper Generation," in *The 6th IEEE/ACM International Workshop on Grid Computing*, 2005, pp. 156–163.

[2] V. Fontes, B. Schulze, M. Dutra, F. Porto, and A. Barbosa, "CoDIMS-G: a data and program integration service for the grid," *Proceedings of the 2nd workshop on Middleware for grid computing*, pp. 29–34, 2004.

[3] A. Blatecky, K. Gamiel, L. Ramakrishnan, D. Reed, and M. Reed, "Building the Bioscience Gateway," *Workshop on Science Gateways: Common Community Interfaces to Grid Resources, GGF*, vol. 14, 2005.

[4] D. Schmidt, "Model-Driven Engineering," *IEEE Computer*, vol. 39, no. 2, pp. 25–31, 2006.

[5] J. Shore, "Continuous design," *Software, IEEE*, vol. 21, no. 1, pp. 20–22, 2004.

[6] S. Mellor, A. Clark, and T. Futagami, "Model-driven development-Guest editor's introduction," *Software, IEEE*, vol. 20, no. 5, pp. 14–18, 2003.

[7] S. Ambler, "Agile model driven development is good enough," *Software, IEEE*, vol. 20, no. 5, pp. 71–73, 2003.

[8] C. Ebert, "Understanding the Product Life Cycle: Four Key Requirements Engineering Techniques," *IEEE Software*, vol. 23, no. 3, pp. 19–25, 2006.

[9] R. Nord and J. Tomayko, "Software Architecture-Centric Methods and Agile Development," *Software, IEEE*, vol. 23, no. 2, pp. 47–53, 2006.

[10] D. J. Reifer, "How good are agile methods?" *IEEE Software*, vol. 19, no. 4, pp. 16–18, 2002.

[11] M. Fowler, K. Beck, *et al.*, *Refactoring: improving the design of existing code*. Addison-Wesley, 1999.

[12] D. Thomas, "Agile programming: design to accommodate change," *Software, IEEE*, vol. 22, no. 3, pp. 14–16, 2005.

[13] S. Davidson, G. Overton, and P. Buneman, "Challenges in Integrating Biological Data Sources," *Journal of Computational Biology*, vol. 2, no. 4, pp. 557–572, 1995.

[14] W. Sujansky, "Methodological Review," *Journal of Biomedical Informatics*, vol. 34, pp. 285–298, 2001.

[15] T. Hernandez and S. Kambhampati, "Integration of biological sources: current systems and challenges ahead," *ACM SIGMOD Record*, vol. 33, no. 3, pp. 51–60, 2004.

[16] S. Valverde and R. Sole, "Hierarchical Small Worlds in Software Architecture," *Arxiv preprint cond-mat/0307278*, 2003.

[17] A. Mockus, R. T. Fielding, and J. D. Herbsleb, "Two Case Studies of Open Source Software Development: Apache and Mozilla," *ACM Transactions on Software Engineering and Methodology*, vol. 11, no. 3, pp. 309–346, 2002.

[18] N. LaBelle and E. Wallingford, "Inter-Package Dependency Networks in Open-Source Software," *Arxiv preprint cs.SE/0411096*, 2004.

[19] S. Valverde and R. Sole, "Logarithmic growth dynamics in software networks," *Arxiv preprint physics/0511064*, 2005.

[20] R. Wheeldon and S. Counsell, "Power law distributions in class relationships," in *Third IEEE International Workshop on Source Code Analysis and Manipulation*, 2003, pp. 45–54.

[21] C. Myers, "Software Systems as Complex Networks: Structure, Function, and Evolvability of Software Collaboration Graphs," *Physical Review E*, vol. 68, no. 4, p. 46116, 2003.

[22] S. Bilke and C. Peterson, "Topological properties of citation and metabolic networks," *Physical Review E*, vol. 64, no. 3, p. 36106, 2001.

[23] A. Vazquez, "Statistics of citation networks," *Arxiv preprint cond-mat/0105031*, 2001.

[24] A. Barabasi and E. Bonabeau, "Scale-free networks." *Sci Am*, vol. 288, no. 5, pp. 60–9, 2003.

[25] A. Barabasi, "The physics of the Web," *Physics World*, vol. 14, no. 7, pp. 33–38, 2001.

[26] S. McGrath, *XML by example: building e-commerce applications*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1999.

[27] J. Nichols and A. Faulring, "Automatic Interface Generation and Future User Interface Tools," *ACM CHI 2005 Workshop on The Future of User Interface Design Tools*.

[28] D. Hyatt, B. Goodger, I. Hickson, and C. Waterson, "XML User Interface Language (XUL) Specification 1.0," *http://www.mozilla.org/projects/xul*, Last visited November 2006.

[29] E. Crowley, K. Roeder, and M. Bina, "A Statistical Model for Locating Regulatory Regions in Genomic DNA," *Journal of Molecular Biology*, vol. 268, no. 1, pp. 8–14, 1997.

[30] C. Song, U. Topkara, J. Woo, S. P. Park, and M. Bina, "Enabling Advanced Bioinformatics Research through SALSA: A Scalable, Simple Architecture," in *TeraGrid Conference*, Indianapolis, IN, USA, June 12–15 2006.

[31] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson, "Grid middleware services for virtual data discovery, composition, and integration," *Proceedings of the 2nd workshop on Middleware for grid computing*, pp. 57–62, 2004.

[32] D. J. Reifer, V. R. Basili, B. W. Boehm, B. Clark, and R. Consultants, "Eight lessons learned during COTS-based systems maintenance," *Software, IEEE*, vol. 20, no. 5, pp. 94–96, 2003.

[33] M. Keil and A. Tiwana, "Beyond Cost: The Drivers of COTS Application Value," *Software, IEEE*, vol. 22, no. 3, pp. 64–69, 2005.