OMICS A Journal of Integrative Biology
Volume 7, Number 1, 2003
© Mary Ann Liebert, Inc.

A Life Scientist's Gateway to Distributed Data Management and Computing: The PathPort/ToolBus Framework

J. DANA ECKART and BRUNO W.S. SOBRAL

ABSTRACT

The emergent needs of the *bioinformatics* community challenge current information systems. The pace of biological data generation far outstrips Moore's Law. Therefore, a gap continues to widen between the capabilities to produce biological (molecular and cell) data sets and the capability to manage and analyze these data sets. As a result, Federal investments in large data set generation produces diminishing returns in terms of the community's capabilities of understanding biology and leveraging that understanding to make scientific and technological advances that improve society. We are building an open framework to address various data management issues including data and tool interoperability, nomenclature and data communication standardization, and database integration. PathPort, short for Pathogen Portal, employs a generic, web-services based framework to deal with some of the problems identified by the bioinformatics community. The motivating research goal of a scalable system to provide data management and analysis for key pathosystems, especially relating to molecular data, has resulted in a generic framework using two major components. On the server-side, we employ web-services. On the client-side, a Java application called ToolBus acts as a client-side "bus" for contacting data and tools and viewing results through a single, consistent user interface.

INTRODUCTION

THROUGHOUT THE LAST CENTURY, various parallel technologies for analyzing DNA, RNA, proteins, and metabolites were developed and applied to the analysis of diverse biological systems. The resulting data has been managed in fragmented and non-standardized ways. Poor data management and a lack of standards has caused under-utilization of the data such that extracting fundamental knowledge and applications from these data sets requires extensive reformatting, repackaging, manual integration, etc. Investments in such efforts detract from advancement of knowledge, technologies, and applications and increase the cost of data utilization. Worse, individual scientists without the necessary IT infrastructure or staff simply cannot take appropriate advantage of many of the data and tools in the field, even when they are moderately aware of them. Finally, the need to support geographically distributed teams working jointly to bring

Virginia Bioinformatics Institute, Virginia Polytechnic Institute and State University, Blacksburg, Virginia.

diverse domains of data together is absolutely essential and cannot be supported by current data management practices in Life Sciences.

To improve management of molecular and cellular biological data, a variety of areas need to be addressed, ranging from interoperability of data and tools, standardization of nomenclature and data communication, to database integration (Sobral et al., 2002). In addition, data and tools need to evolve to meet the needs of traditional human users as well as becoming machine-readable so that software-agent technologies are capable of identifying and retrieving resources.

One way to immediately impact the field is to leverage existing technologies for interoperation from the e-business community. Extensive development of web services has occurred to support the business community's needs. Much of these developments have come from major technology providers, such as IBM, Sun, and Microsoft.

In addition to utilizing a web-centric model for managing distributed data and tools, another area of need is the development of standards for molecular and cellular data, especially data communication standards. A number of efforts exist in this regard. The Interoperable Informatics Infrastructure Consortium (I3C) represents one notable effort in which public and private members work together to develop XML-based standards for life sciences data (I3C, 2003). The adoption of standards in the active and creative bioinformatics community, however, is not widely accepted as strict standards are sometimes viewed as a hindrance to creativity.

The development of annotation standards and knowledge representation ontologies for molecular and cell data offers another important area of activity to catalyze community standards. Current efforts include the Gene Ontology Consortium (2001), the COG database (Tatusov et al., 2001), and the InterPro database (Apweiler et al., 2001).

We report here on the development of an open framework to support a web-services model of data and tool interoperability and its application to molecular and cellular data sets for key host-pathogen-environment interactions (pathosystems). The framework is built primarily using two major components. On the server-side, we employ web-services. On the client-side, a Java application called ToolBus acts as a client-side "bus" for contacting data and tools and viewing results through a single, consistent user interface. Web-services provide the opportunity for data and tool interoperability without the imposition of strict standards. The generic framework being built comes under the auspices of a project known as Pathogen Portal (PathPort, www.vbi.vt.edu/~pathport/).

THE PROBLEM

Life Science researchers, including those at Virginia Bioinformatics Institute (VBI), are already dealing with large amounts of a wide variety of data, from sequencing trace files to mass spectra. The growth in size of these data sets is proceeding at rates greater than the growth in the number of transistors that can be placed on a chip (Moore's Law). In addition, scientists and other Life Sciences data consumers have a need to convert these data into information about genomes, proteins, protein expression patterns, metabolites, and the interaction pathways that enable biological systems and processes to be investigated and understood. Thus, the development of a framework for data and tool management and interoperability to support the various types of Life Sciences data consumers at large is paramount.

In the first year, our efforts have focused on genomic data (e.g., DNA) and related information (e.g., DNA sequences, genome annotations, genome comparisons, phylogenetic relationships). Currently, the majority of the molecular data reside here. Over the next 3–5 years, however, data about the state of an organism and biological processes will be increasingly available. The community is shifting from solely "blueprints of life" (DNA/genomes) data acquisition to acquiring data about the dynamics of living organisms as they respond to specific perturbations (temporal mRNA profiles, protein expression profiles, protein-protein interactions, and metabolite expression profiles, for example). Thus, microarray gene expression profiles (developed on diverse technology platforms), proteomics data, metabolomics data, and geospatial and other environmental data (providing the context for the dynamic responses of living organisms), even not yet identified, will need to be incorporated and made available through the system's framework. Our chal-

lenge was therefore twofold: to build a system to deal with different kinds of data, and to allow relationships between these data to be discovered and shared by the user.

With these challenges in mind, our primary concern was to build a highly integrated system that would not become brittle as new data types and user features were added. We also understand that Life Sciences data, people, tools, and resources will remain distributed and reside partially in the public sector. Life Sciences data are also in various formats, under diverse forms of access, and with problematic ontologies. Tools to operate on Life Sciences data expect different data formats, have OS restrictions, display diverse interfaces, and tend to be very poorly integrated. Finally, visualization of Life Sciences data provides additional challenges. The data are highly complex, as are the analysis results. Part of the discovery process is finding relationships among data of various types. For these and other reasons, we believe that the framework must be open, extensible, and built on industry standards wherever possible.

OUR APPROACH

The need to solve the Life Sciences data management problem exists within our organization, for our collaborators, and for the Life Sciences community at large. The requirements include a flexible, extensible, and scalable working system to meet the needs of existing and potentially new data types. We have started from molecular data types (DNA, mRNA, proteins, and metabolites), with a long-term goal of working "upwards" to higher levels of organization (where in some cases better data management has been accomplished, e.g., environmental data). Furthermore, we needed to be able to construct a production quality base system within twelve months because of the requirements of our sponsors for development and deployment of the system to support infectious disease research, which we call "PathoSystems Biology." As a result at project inception, we were less concerned about taking on interesting research problems and more concerned about software engineering. We also recognize that web services are not the solution to all data management problems. Many areas of research, such as the development of reference ontologies for biological systems, the standardization of biological nomenclature, and data model generation, may be made available as web services that enrich the capabilities of our framework as such efforts yield results (Apweiler et al., 2001; The Gene Ontology Consortium, 2001; Tatusov et al., 2001).

To create a flexible and extensible system, we focused on a "bus" architecture (*ToolBus*) to connect "plug-in" data sources, analysis tools, and visualization components. We designed *ToolBus* to be domain independent to avoid inadvertent brittleness when new data types needed to be supported later. In addition, by keeping the plug-ins separate from the bus, we were forced to create an application-programming interface (API) to enable work on the different plug-ins to proceed in parallel with the development of *ToolBus* with a minimal amount of required communication between developers. In practice, the bus API was not frozen before development of the plug-ins began, but was sufficiently well defined to allow their initial development. The parallel development of both *ToolBus* (four software engineers) and plug-ins (12 software engineers) provided excellent feedback on the API enabling problems to be corrected early in the development process. We used a modified spiral software development model to enable changes early in the process and minimize costs (Boehm, 1988).

We chose to develop *ToolBus* using an object-oriented programming language—Java (Flanagan, 2002)—permitting the API to be designed as a set of extensible classes. This approach has allowed easy development of the various plug-ins. In addition, the use of abstract classes rather than interfaces as the basis for the API enforced a similar "look and feel" (e.g., menus and options). The use of abstract classes also makes the addition of new API features easy to incorporate by giving them default, usually null, functionality that can be overridden. The adoption of Java also enabled support of multiple client platforms with minimal additional effort. As Life Scientists vary in their operation system adoption, including Unix, Windows and MacOS, the support for multiple-client operating systems was crucial. While other languages would have sufficed, some additional features of Java (e.g., dynamic class loading) and third-party support libraries have proven to be extremely helpful.

The ability to relate the information represented by the different visualization components to one another was one initial design concern, without which the system would be little better than a collection of inde-

pendent programs. As a result, the design incorporates information groups that are able to point to data represented by different visualization components, thus forming a bridge between the different kinds of information being visualized.

OVERALL DESIGN

Figure 1 depicts the overall structure of the ToolBus design. The bus uses a mediator pattern (Gamma et al., 1995), **ToolBus**, to organize the various parts of the system. Items in bold are single class instances: **Toolbus**, **ToolManager**, **Associator**, **GroupCompare**, and **EventManager**; normal indicates multiple instances (Group, DataItem); and *italics* denotes the abstract classes that plug-ins must extend (*Tool*, *Model*, *View*, and *ViewManager*). Arrows indicate object references, so *Model* and *View Manager* instances know about one another, while the *View Manager* knows about its *Views* and not vice versa. Finally, the **EventManager** underlies all of the client-side except Groups and Data Items, serving as an additional mediator specific to events.

The **ToolBus** class is the mediator of all data coming into *ToolBus*. Normally, a user will invoke a *Tool*, which typically contacts web-services; the returned results are passed to it via **ToolManager**. **ToolBus** then determines which of its installed *Model*s understands the information returned by the *Tool* and prompts the user as to which new *Model*s to create and which existing *Model*s instances to extend. *ToolBus* allows the development of extensible *Model*s that can accept additional data after they have been created. This approach supports, for example, the inclusion of additional gene predictions (from other algorithms) into an existing genomic annotation model instance. The *Model*s then create their own *ViewManager* that manages the different *View*s for this model. *Model*s in this context should be thought of as data models, and the ability to have multiple different kinds of views of that data can be vitally important in better understanding its meaning.

Web-services are the prime example of the independent *Tool*s we use, though local programs and custom tools can also be created and used. Web-services provide a platform-independent means of accessing both data and analysis, while adhering to XML output allows *ToolBus* and its visualization components to make certain assumptions in dealing with the information. While a web-service is a particular type of *Tool*, **ToolManager** can manage many web-service *Tool*s. Different instances of the web-service *Tool* are created, each storing the URL pointing to its own WSDL document. Different kinds of tools can be added to *ToolBus* by extending the *Tool* abstract class (e.g., FtpTool, PhyloTool). *Tool* usage is synchronized, thus web-services have RPC-like behavior, which makes their development easier since no session-management is necessary and they can almost always be implemented as stateless services.

Visualization components are built by extending the three abstract classes: *Model*, *View*, and *ViewManager*. *Model* is the data model and is created by **ToolBus** based on the XML results of invoked tools. While

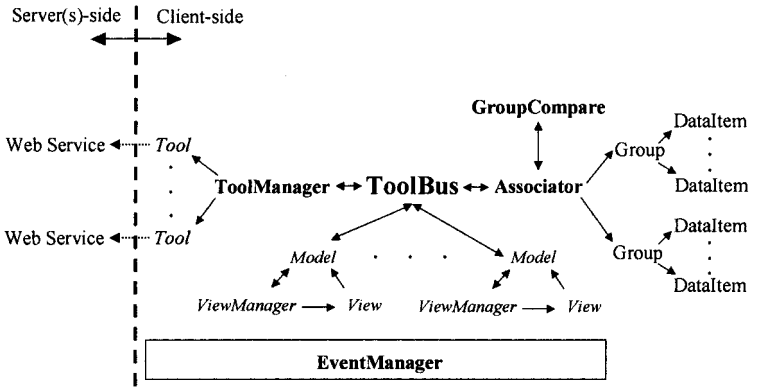

FIG. 1. Structure of the *ToolBus* design.

not an essential requirement, *Model*s normally take their data in XML format, and allowing developers to use Castor (ExoLab Group, 1999) to parse the data, extracting the information their *Model* needs. Rather than using the Model-View-Controller pattern (Gamma et al., 1995), we use the simplified Model-Delegate pattern where the *ViewManager* is responsible for updating all of the views whenever the model state is altered.

A feedback loop from *Model*s to *Tool*s is essential for supporting the kind of interactive discovery based work paradigm essential to information mining. The **EventManager** underlies most of *ToolBus* and enables the drag-'n'-drop of information from *Model* (*View*s from the user's perspective) to *Tool*s. For example, dragging a gene from a genome annotation *Model* to a PCR/Hybridization probe design *Tool.* This also allows information to be dragged from one *Model* to another *Model* (again, via their corresponding *View*s from the user's perspective). Because *Tool*s often need information of a certain type or information presented within a certain form, drag-'n'-drop data are really a collection of data in a Vector. *Tool*s, web-service tools in particular, know the form of the data they expect.[1] In our example, the *Model* from which we dragged our gene of interest could build a vector supplying the raw DNA sequence, a DAS[2] (Stein et al., 2002) sequence, the gene's textual annotation, and the translated protein sequence corresponding to the gene's DNA sequence. The probe design *Tool* would then test each of these values against what it expects and use the first acceptable data form match. This puts the task of knowing about data form transformation in a single place in the design. Furthermore, placing this task in the *Model* should require less work, as the number of *Tool*s is expected to be far greater than the number of *Model*s that would work with such data.[3] This ability to drag-'n'-drop information from visualizations to tools (or each other as in the case of the File System *Model*, causing the dragged information to be saved in a file) provides a degree of inter-connectedness not seen in other decoupled systems.

Another important *ToolBus* ability is supporting the formation of relationships between data in different *Model*s. This is accomplished by Groups and the **Associator**. Data from *Model*s can be selected (via their corresponding *View*s) and placed within a user defined Group. The **GroupCompare** class allows users to compare multiple Groups via user generated Venn Diagrams, choosing to highlight or hide the membership of entire groups, intersections, unions, complements and any combination thereof.[4] Group creation based on Data Items from one or more *Model*s allows arbitrary associations to be created (e.g. these "up-regulated" genes, located at this position on the chromosome, have homologs and orthologs in these other organisms) so that ToolBus remains domain-independent. It is this domain-independence that gives us confidence that we can meet the challenge of incorporating any kind of future data type into the system.

Saving and loading of work sessions is also an important *ToolBus* requirement—allowing the user to resume previous work sessions as well as to share them with colleagues and collaborators. The ultimate goal is for loaded sessions to look *exactly* as they did when saved. This has proven to be difficult because of the non-tree-like structure of the data.[5] Thus, Java serialization of work session data was necessary. This required a change from JAXB (Sun Microsystems, 1995) to Castor (ExoLab, 1999) for XML parsing by *Model*s since Castor objects are serializable and JAXB objects were not at that time. Saving and loading of work sessions also necessitates careful design and implementation of *Model*s so as to ensure their serializability. So far this has not proven to be a stumbling block, though it is the main reason why *ToolBus* does not currently restore *View*s exactly as saved.[6] While there are work-arounds to saving *View* information, we currently feel the effort required to be greater than the perceived benefit at this time.

---

[1] *ToolBus* friendly web-services make additional information available via a specially named operation that yields these specifications. Currently the data form is indicated using a mix of regular expressions (Friedl, 2002) and simple XPath expressions (Clark and DeRose, 1999).

[2] Distributed Annotation System (DAS) is based on the DAS/1 XML format developed by Lincoln Stein.

[3] *ToolBus* should not perform this function as the desire for domain independence would be compromised.

[4] The DataItems within a Group contain a *Model* reference and a Datum reference within that *Model,* thus **ToolBus** does not have any knowledge or control over the internal design of *Models.*

[5] DataItems and *Models* both reference Datum within the *Models.*

[6] *ToolBus* has been implemented using Java Swing (Flanagan, 1999). Although Swing is advertised as being serializable, we have found holes in this ability that nearly always cause our *Views* to fail serialization.

DESIGN ENHANCEMENTS

A number of additional capabilities have been identified as desirable during our development of *ToolBus*. Some of these were issues known at the outset, but whose detailed design was postponed so that their requirements could be re-evaluated in light of detailed design changes that were expected and have come about during the development of the system. These issues and capabilities include dynamic plug-in installation, automated group suggestion, tool discovery, and security and accounting.

Originally, the entire Java classpath was searched by *ToolBus* to discover tools and models. As the classpath increased in size, this caused unacceptably longer system startup times. To reduce startup time, we decided to search a smaller path that was passed as a command line argument to the Java interpreter. This required the inclusion of a dynamic class loading facility, which Java supports. Not only did this solve the immediate problem but it also allowed us to include a dynamic plug-in installation menu option to *ToolBus* and provides for the future possibility of allowing tools (e.g., web-services) to download *Model-View-ViewManager* plug-ins that understand and can display the data returned by the tool. Furthermore, this might be done automatically by *ToolBus* if it checks to see whether the tool you are about to run will give results for which you have no corresponding *Model*.[7]

Group creation is currently a high manual and slow user-driven process. Our first thought was to build a generic domain independent mechanism into *ToolBus* that would snoop all the information[8] and make possible new group suggestions to the user. Recently we have decided to use visualization components for this purpose since *Model*s are asked about all the data that comes across the **ToolBus** (i.e., do they understand it) and can thus provide group suggestions based on data snooping without having to incorporate this ability into *ToolBus* itself, thus making the system more flexible and leaving open the possibility for multiple, possibly domain-aware, group suggestors without requiring new *ToolBus* releases or updates.

Because it is unlikely that users will know about the existence of all the tools they might ever want or be able to use, we originally incorporated a Universal Description Discovery and Integration (UDDI) (Apte and Mehta, 2002) search capability into **ToolManager** to aid in web-service discovery. Unfortunately, the UDDI libraries are quite large and it wasn't clear to us that UDDI would continue to be actively used in the future or that it wouldn't be substantially supplanted by a newer technology, for example, based on LDAP (Weltman and Dahbura, 2000). We have since made the tool search facility web-service based. Thus the UDDI specifics have been removed from *ToolBus* and placed into a tool-searching web-service. Not only does this streamline *ToolBus*, but the tool-searching web-service can be easily updated to incorporate LDAP and other potential tool directory organizations without requiring changes or additions to *ToolBus*.

Security has become a greater concern in recent years. Currently, *ToolBus* uses the HTTP protocol for contacting web-services and we have recently added support for HTTPS (HTTP + SSL), for non-self-signed certificates. We plan to add additional encryption for the document in the SOAP envelope in an effort to protect logname and password safeguards, in addition to data passed to and from the web-service. While logname/password systems are common, many users are accustomed to single sign-on web-portals. Unfortunately, single sign-on won't work in this context since the tools (e.g., web-services) may not belong to the same organization. Although not yet implemented, we plan to use an Authentication, Authorization, and Accounting web-service (AAAWS) into which users would login that would return a timed ticket usable by all tools provided by that organization. Such tickets would be managed by *ToolBus* via a ticket visualization component, with users simply dragging the appropriate ticket to be used by their tools to provide the necessary information.

[7] Web-service tools provide additional information about the kind of data that the operations take as parameters. Likewise, this information is also provided for the return type; the XPath expression describing the return type could be passed to *Models* to determine which can understand this data format.

[8] This is possible since *ToolBus* uses a mediator pattern via the **ToolBus** class.

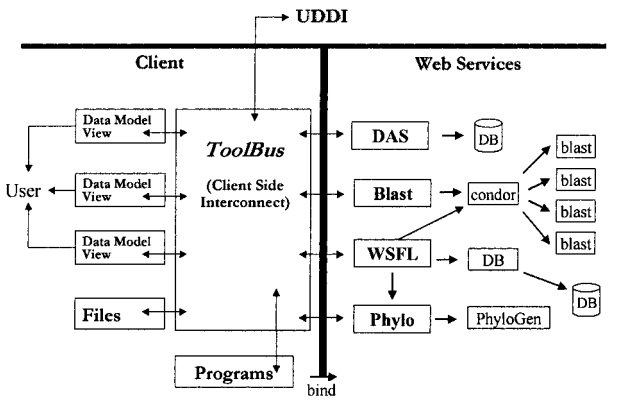

FIG. 2. PathPort architecture in which *ToolBus* connects server data sources and analysis tools with client visualization plugins. PathPort utilizes *ToolBus* plus specific data and tools associated with host-pathogen-environment interactions. The project exemplifies how technology can be leveraged to support distributed collaborations in Life Sciences, such that the data and tools selected provide the specificity of the research topic, rather than it being hard-wired into the framework. This approach should significantly enhance the re-use of the framework for different thematic areas of research. UDDI, universal description, discovery and integration.

TOOLBUS ADVANTAGES

ToolBus' unique combination of features supports a rich set of capabilities for rapid, scalable, and platform independent development of large systems, such as *PathPort*[9] (Fig. 2), which span a wide variety of information types within sub-topic areas contained within an overarching domain. The advantages of *ToolBus* include the following:

1. Scalability, because of a lack of centralization of data sources, analysis tools, or even within the data models for information within a domain.
2. Platform independence, as a consequence of using XML and the Java programming language and associated technologies.
3. Domain independence, which ensures that *ToolBus* will be able to support new data types as they are identified for inclusion within *PathPort.*
4. Rapid visualization component development, which can be accomplished by taking advantage of common elements from abstract base classes within the API. This allows developers to focus more on visualization rather than recreating the common needs of all plug-ins.
5. Information management, by virtue of being a client-side interconnect, *ToolBus* gives users greater control over information and analysis resources from a variety of off-site servers and allows them to download only that data necessary for their work rather than entire databases.
6. Collaboration, which is possible by sharing saved work sessions (e.g., via email) with colleagues and collaborators at the same or different institutions.
7. Compliments grids, with web-services serving as points of entry into Grids (i.e., Grid-services; Foster et al., 2002).
8. Information grouping, comparison, and combination, which create related groups of information and support interactive comparison of their membership by user-generated Venn diagrams.

[9] *PathPort,* a biological "pathogen portal," uses *ToolBus* to bring together molecular, cellular, and literature data sources and analysis tools, in addition to domain specific visualization components, to provide an integrated bioinformatics system for pathosystems research (www.vbi.vt.edu/~pathport/).

9. Tool reuse, that is accomplished by building web-services from collections of pre-existing web-services, or by constructing, for example, an interactive query tool on top of an existing database access web-service.

CHALLENGES

As mentioned previously, automatic group suggestion is essential as we move into the larger data sets for microarray gene expression experiments. In addition, the creation of lexicon ontologies (or dictionaries) would greatly aid in the construction of interesting groups whether created statically by the Life Sciences community or "on-the-fly". Unique identifiers, such as the GenBank accession number (NCBI, 2003), are helpful but are insufficient to discover interesting relationships for which no standard naming convention exists.

As users gain experience with a particular type of data, they will want to create workflow scripting. This helps to avoid sending large data sets (e.g., a complete genome) back and forth between the client and servers. In this case, a work script should be sent to a server and the data transfers should occur on the presumably faster network to which the server is attached. Such workflow scripting languages are still being developed, for example, BPEL4WS (Curbero et al., 2002), but as a class they are immature, with no complete implementations as of yet. *ToolBus* should facilitate the creation of such workflows by recording user actions and then allowing users to turn these action sets into parameterized workflows, thus permitting interesting discovery processes to be codified as reusable macros.

In addition to sending a workflow script to the server, additional benefits can be realized by supporting lazy database query evaluation when the results are passed to a separate analysis tool. Our view is that while pushing the analysis into the database can make the whole process faster, there will always be additional analysis tools that researchers will want to use. Thus, pushing BLAST into the database does not help users of FASTA. Solving the general problem is harder, but ultimately the community gleans longer-term benefit as a result.

There are few appropriate XML standards, and those that do exist often need additional customization [e.g., pair scores for MSAML (NCBI, 2001), combining segments and features in the same DAS document]. As a result, we have often needed to create our own XML formats (e.g., for describing web-service result attachments, DNA digestion, molecular pathways), which may reduce the generality and resuability of our efforts. This is a community problem which the I3C (2003) is trying to solve, but the answer is probably not a few large XML formats [e.g., BSML (2003)], but a myriad of small ones [e.g., MSAML (NCBI, 2001), DAS (Stein et al., 2002), BLASTXML (Stajich, 2003)]. Development of new standardized XML formats and accepted translations between relevant formats is vitally important.

ToolBus assumes that tools are used synchronously. The web-services we have built are typically stateless and so there is no provision for asking how a certain task is progressing. The RPC model is easy to implement, but it can lead to long wait times for certain tools. We provide a pop-up status monitor that alerts the user when the task was started and the elapsed time, but if the user exits *ToolBus* then any results returned by the tool will be lost. The preferred solution would be for *ToolBus* to remember that there were outstanding tool invocations and for tools to store results temporarily until *ToolBus* was reinvoked so that it could deliver the results, though there are currently no plans to do this.

Earlier, we mentioned plans to create a AAA web-service and associated visualizer plug-in. Unfortunately, since all models can see the data that comes across *ToolBus*, this means that other models can see the tickets. While the tickets will be encrypted, this would not prevent a rogue model from unintended use of a tool utilizing the user's tickets. Such use could even be hidden from the user by performing the web-service contact actions itself rather than going through *ToolBus.* A class instance based sandbox mechanism for limiting access to resources is needed (e.g., sockets).

CONCLUSION

The Information Technology challenges in Life Sciences are many and complex. Scientists need integration and interoperation of Life Sciences data, analysis tools and visualization components. Di-

verse tools must be easy to use from the perspective of data consumers. Data storage systems are and will continue to be distributed. Users also require high performance, high-throughput computing for data processing and analysis, which must also be distributed. These goals cannot be achieved by a single organization working in isolation. Therefore, in addition to technical challenges, diverse and complex sociological barriers need to be overcome to enable data management in life sciences to advance.

ACKNOWLEDGMENTS

The PathPort project, which resulted in the creation of *ToolBus*, was supported by DoD grant (DAAD 13-02-C-0018) to Dr. Bruno Sobral. We are grateful to our corporate partners, IBM and Sun Microsystems, for their support of this work via a Sun Center of Excellence in Bioinformatics and an IBM Shared University Research Award, respectively, to Dr. Bruno Sobral. We are grateful to our collaborators at Soldier Biological and Chemical Command (SBCCOM) at Edgewood, especially Dr. Jay Valdes and his team, for feedback throughout the software development process. Thanks to Drs. Stefan Hoops and Allan Dickerman (VBI) for key suggestions and comments as members of the PathPort team. The entire PathPort team deserves thanks for their dedicated software engineering in the face of rapidly changing needs. Special thanks to Dr. Neysa Call (VBI) for editorial comments and support.

REFERENCES

APTE, N., and MEHTA, T. (2002). *UDDI: Building Registry-Based Web Services Solutions.* (Prentice-Hall PTR, Indianapolis, IN).

APWEILER, R., ATTWOOD, T.K., BAIROCH, A., et al. (2001). The InterPro database, an integrated documentation resource for protein families, domains and functional sites. Nucl. Acids. Res. **29**, 37–40.

BOEHM, B. (1988). A spiral model of software development and enhancement. IEEE Computer **21,** 61–72.

BSML (Bioinformatics Sequence Markup Language). (2003). Available: http://www.bsml.org/.

CLARK, J., and DEROSE, S., eds. (1999). *XML Path Language (XPath) Version 1.0.* Available: www.w3.org/TR/xpath.

CURBERO, F., GOLAND, Y., KLEIN, J., et al. (2002). *Business Process Execution Language for Web Services, Version 1.0.* Available: http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/.

EXOLAB GROUP. (1999). *The Castor Project.* Available: http://castor.exolab.org/.

FLANAGAN, D. (1999). *Java Foundation Classes in a Nutshell.* (O'Reilly & Associates, Sebastopol, CA).

FLANAGAN, D. (2002). *Java in a Nutshell*, 4th ed. (O'Reilly & Associates, Sebastopol, CA).

FOSTER, I., KESSELMAN, C., NICK, J., et al. (2002). Grid services for distributed system integration. Computer **35.**

FREIDL, J.E.F. (2002). *Mastering Regular Expressions*, 2nd ed. (O'Reilly & Associates, Sebastopol, CA).

GAMMA, E., HELM, R., JOHNSON, R., et al. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* (Addison-Wesley, Boston, MA).

THE GENE ONTOLOGY CONSORTIUM. (2001). Creating the gene ontology resource: design and implementation. Genome Res. **11**, 1425–1433.

I3C (INTEROPERABLE INFORMATICS INFRASTRUCTURE CONSORTIUM). (2003). Available: http://www.i3c.org/. Accessed 2/7/03.

NCBI (National Center for Biotechnology Information). (2001). MSAML (Multiple Sequence Alignment Markup Language). Available: http://maggie.cbr.nrc.ca/~gordonp/xml/MSAML/.

NCBI (National Center for Biotechnology Information). (2003). GenBank overview. Available: http://www.ncbi.nlm.nih.gov/Genbank/GenbankOverview.html.

SOBRAL, B., ECKART, D., LAUBENBACHER, R., et al. (2002). The role of bioinformatics in toxicogenomics and proteomics. Presented at the NATO Advanced Workshop on Toxicogenomics and Proteomics.

STAJICH, J. (2003). NCBI's Blast XML. Available: http://doc.bioperl.org/releases/bioperl-1.0.2/Bio/SearchIO/blastx ml.html.

STEIN, L., EDDY, S., and DOWELL, R. (2002). *Distributed Sequence Annotation System (DAS): Version 1.52.* Available: http://www.biodas.org/documents/spec.html.

SUN MICROSYSTEMS. (1995). *Java Architecture for XML Binding (JAXB).* Available: http://java.sun.com/xml/jaxb/.

TATUSOV, R.L., NATALE, D.A., GARKAVTSEV, I.V., et al. (2001). The COG database: new developments in phylogenetic classification of proteins from complete genomes. Nucl. Acids. Res. **29**, 22–28.
WELTMAN, R., and DAHBURA, T. (2000). *LDAP Programming with Java.* (Addison-Wesley, Boston, MA).

Address reprint requests to:
Dr. B.W.S. Sobral
Virginia Bioinformatics Institute
Virginia Polytechnic Institute and State University
1880 Pratt Drive
Building XV (0477)
Blacksburg, VA 24061

E-mail: sobral@vt.edu