Designing Highly Flexible and Usable Cyberinfrastructures for Convergence

BRUCE W. HERR, WEIXIA HUANG, SHASHIKANT PENUMARTHY, AND KATY BÖRNER

School of Library and Information Science, Indiana University, Bloomington, Indiana 47405, USA[1]

ABSTRACT: This article presents the results of a 7-year-long quest into the development of a "dream tool" for our research in information science and scientometrics and more recently, network science. The results are two cyberinfrastructures (CI): The *Cyberinfrastructure for Information Visualization* and the *Network Workbench* that enjoy a growing national and interdisciplinary user community. Both CIs use the cyberinfrastructure shell (CIShell) software specification, which defines interfaces between data sets and algorithms/services and provides a means to bundle them into powerful tools and (Web) services. In fact, CIShell might be our major contribution to progress in convergence. Just as Wikipedia is an "empty shell" that empowers lay persons to share text, a CIShell implementation is an "empty shell" that empowers user communities to plug-and-play, share, compare and combine data sets, algorithms, and compute resources across national and disciplinary boundaries. It is argued here that CIs will not only transform the way science is conducted but also will play a major role in the diffusion of expertise, data sets, algorithms, and technologies across multiple disciplines and business sectors leading to a more integrative science.

KEYWORDS: cyberinfrastructure; OSGi; plug-in; data models; analysis; network science; scientometrics; usability; flexibility; extensibility

INTRODUCTION

Fifty years back in time, few scientists used computers to conduct their research. Today, science without computation is unthinkable in almost all areas of science. Innovation and progress in most areas of science require access

Address for correspondence: Katy Börner, Indiana University, SLIS, 1320 East Tenth Street, Bloomington, IN 47405. Voice: 812-855-3256; fax: 812-855-6166.
e-mail: katy@indiana.edu

[1] This material is based upon work supported in part by the 21st Century Fund and the National Science Foundation under Grant No. IIS-0238261 and IIS-0513650. Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

Ann. N.Y. Acad. Sci. 1093: 161–179 (2006).
doi: 10.1196/annals.1382.013

to advanced computational, collaborative, data acquisition, and management services available to researchers through high-performance networks. These environments have been termed *cyberinfrastructures* (CI) by a National Science Foundation (NSF) blue-ribbon committee lead by Daniel Atkins (Atkins *et al.* 2003). Today, some CIs provide access to thousands of interlinked experts, services, federated data sets, and compute resources. They have reached a complexity that is hard if not impossible to specify, implement, and manage in a top-down fashion. Also, most content providers and users of these CIs are not computer scientists. They are biologists, physicists, social scientists, information scientists, and others with deep knowledge about data and algorithms and a deep interest to save lives, secure important technical infrastructures, and discover universal laws.

Scientists today use commercial packages, such as Microsoft Excel, SPSS (http://www.spss.com), Matlab (http://www.mathworks.com), Adobe Photoshop, etc., to analyze and model their diverse data sets and to visualize research results. These tools come with easy to use, menu-driven interfaces through which a set of standard features can be used. Data and file sharing are easy if collaborators use the very same tools and versions. However, it is not trivial to add your own or any other algorithm without major modifications. It is impossible to get a "customized filling" of the tools with exactly those algorithms that are needed by a user or user group.

In response to this need, a growing number of grass roots efforts aim to create data and software libraries, application programming interfaces (APIs), and repositories that help disseminate the best algorithms. In many cases, algorithms are made available as open source so that the concrete implementation can be examined and improved if necessary. Sample efforts are R (Ihaka and Gentleman 1996), StOCNet (Huisman and Duijn 2003), Jung (O'Madadhain *et al.* 2003), and Prefuse (Heer *et al.* 2005). However, the mentioned packages come as APIs or require scripting of code. None of them supports the easy, wizard-based integration of new algorithms and data sets by developers or provides a menu driven, easy to use interface for the application user.

Grid computing (Berman *et al.* 2003) aims to address the computational needs of "big science" problems, such as protein folding, financial modeling, earthquake simulation, or climate/weather modeling. It follows a service-oriented architecture and provides hardware and software services and infrastructure for secure and uniform access to heterogeneous resources (e.g., different platforms, hardware/software architectures, and programming languages), located in different places belonging to different administrative domains linked on a network using open standards. It also supports the composition of applications and services, workflow design, scheduling, and execution management. Using grid computing today is challenging. To fully take advantage of the grid, one's code must be (re)written to run in parallel on a cluster of potentially very different operating systems and environments, which appears to be a major challenge for most people who are not computer scientists. Further, access to

the grid is usually through command line interfaces or customized portals optimized for the needs of specific communities. Finally, while grid computing is a powerful approach to "big science" computations, many applications can be served well without parallel computing and distributed databases. The time and effort required to make an application grid-able is considerable and only justifiable if grid resources and functionality are truly needed.

In sum, there is a gap between algorithm developers and application designers and application users. Many algorithm developers are searching for easy ways to quickly disseminate their work. Many researchers and educators are in need of good algorithms but are not equipped with the mathematical sophistication and programming knowledge required to benefit from code descriptions in research papers, implemented APIs, or advanced CIs. In many cases, users are not only interested in a single algorithm but they want tools similar to MS Excel, SPSS, Matlab, or Photoshop but with the option to customize and extend them according to their needs.

The cyber infrastructure shell (CIShell) specification aims to serve the needs of all three user groups: algorithm developers, application designers, and application users. Building on the open services gateway initiative (OSGi) specification, it supports the easy, wizard-driven plug-and-play of existing and new algorithms and data sets that are implemented as services. Data, algorithms, and additional CIShell services, such as graphical user interfaces (GUIs), schedulers, and logging services can be combined and deployed as a stand-alone tool, (Web) service, or peer-to-peer service, among others. The core CIShell implementation can be "filled" with high performance services or the data sets and services needed in a classroom setting. Hence, CIShell creates a bridge between algorithm developers, application developers, and their users.

The remainder of this article is organized as follows: First, the needs of the user groups CIShell aims to serve are detailed. Second, the inner workings of CIShell are described on an abstract level. Third, how CIShell is used by the three different user groups is explained. Fourth, diverse reference implementations are introduced. Finally, the article concludes with a discussion and an outlook of future work.

WHAT USERS WANT

Today, it is not only computer scientists who develop and create novel data sets and algorithms but also biologists, physicists, and social scientists among others. In many cases, the data sets and algorithms are used almost exclusively by the person, lab, or project that created them. Some are distributed via private web pages. Consequently, many algorithms are implemented and reimplemented countless times—a true waste of lifetime and resources. Given the effort it takes to implement a set of algorithms, comparisons of novel with existing algorithms are rare. Some algorithms are made available in compiled

form or as source code. These efforts are truly appreciated by the community and lead to higher citation counts of related papers. However, there is no way to get an overview of all existing data sets and algorithms. To make things worse, data sets come in diverse formats and algorithms are implemented in very different languages and with diverse input and output formats. The diffusion of high quality data sets and novel algorithms would greatly benefit from a means to easily integrate and use existing data sets and algorithms.

There is also a need for the design of tools that provide access to exactly those data sets and algorithms that are needed by a researcher, group, or community. Commercially available tools often provide menu-driven interfaces, remote services, or scripting engines. They have workflow support, scheduling support, etc. Users will expect this from customized tools.

Analyzing and making sense of data sets frequently involves multiple steps, such as sampling, cleaning, analyzing, and sometimes visualizing for means of communication. For means of illustration, FIGURE 1 shows the diverse data sets (given in italics and underlined) and processing steps involved in a scientometric study (Mane and Börner 2004) aiming at the identification of the topic coverage of a document data set. The analysis starts with a list of documents—one document per line. This list is parsed and a term-document matrix is generated in which each cell entry states how often a certain term occurred in a certain document. The resulting term-document matrix is used to identify the top 50 most frequent terms. Next, the co-occurrence similarity of those top 50 terms is calculated. The more often two terms occur together in the same document, that is, in the same line of the original list of all documents, the

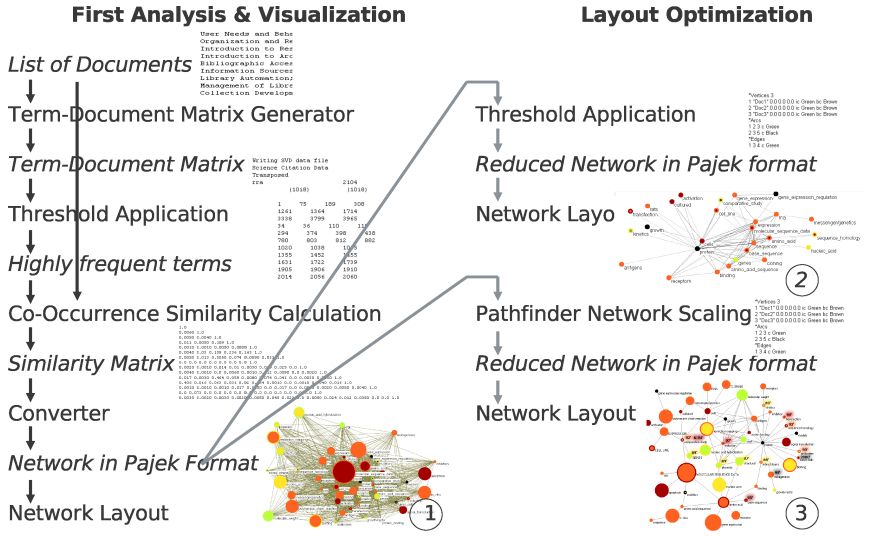

FIGURE 1. Sample acquisition, analysis, and visualization of a topic network.

higher their similarity. The similarity matrix is then converted into a list of nodes and edges that can be read by the Pajek network layout tool (Batagelj and Mrvar 1998). Unfortunately, the generated layout labeled with (1) is not very readable as almost all terms co-occur with each other in at least one of the many documents. Layout optimization is needed. In a first attempt, a threshold was applied to eliminate links below a certain similarity. However, this strategy disintegrates the network into one larger subgraph labeled (2) and many unconnected nodes. In a second attempt, Pathfinder Network Scaling (Schvaneveldt 1990) is applied to ensure that all 50 nodes stay connected yet a more readable layout is achieved, see visualization labeled with (3).

Most analyses in scientometrics and other fields of science require many more processing steps (Börner *et al.* 2003, 2007). Commonly, the output of one processing step is not compatible with the input of the next step. Tools and algorithms applied in one and the same study might be implemented in different programming languages and might only run on certain operating systems making data transfer across and the switch between platforms necessary. Many analyses are highly iterative. It is only after the entire sequence of processing steps is completed that data errors or layout optimization needs become visible. Some algorithms have a high computational complexity and have to be scheduled.

Taken together, researchers involved in data analysis, modeling, and visualization would highly benefit from a specification/application that supports the plug-and-play of algorithms written in different languages. Algorithm interface standards or converters are needed to accommodate different input and output formats. Ideally, users can select existing and generated data sets, algorithms, and existing tools via a GUI that also provides logging, scheduling, and other services.

CISHELL DESIGN

The CIShell is an open source, community-driven specification for the integration and utilization of data sets, algorithms, tools, and computing resources. The CIShell specification, API, and related documentation are available at http://cishell.org. The specification and all its reference implementations are open sourced under the Apache 2.0 license.

CIShell builds upon the OSGi specification, as described later. By leveraging OSGi, we gain access to a large amount of industry standard code and know-how that would take years to reinvent/implement. Each application—be it a stand-alone application, a scripting engine, a remote server-client, or a peer-to-peer architecture—resembles an ecology of services. FIGURE 2 depicts how CIShell applications can be deployed as distributed data and algorithm repositories, stand-alone applications, peer-to-peer architectures, and server-client architectures.

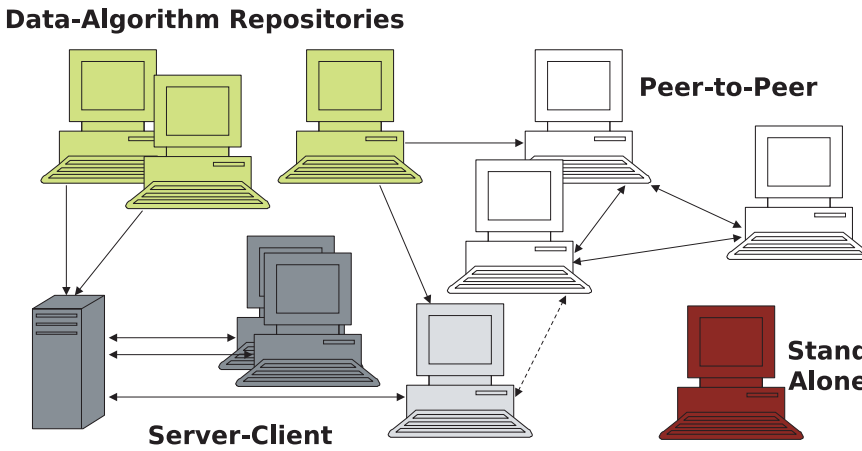

FIGURE 2. Deployment options for CIShell applications.

OSGi Specification

The OSGi specification (http://www.osgi.org) defines a standardized, component-oriented computing environment for networked services. It has been successfully used in industry from high-end servers to embedded mobile devices for 7 years. OSGi alliance members include IBM, Sun, Intel, Oracle, Motorola, NEC, and many others. OSGi is widely adopted in the open source realm, especially since Eclipse 3.0 has adopted OSGi R4 as its plug-in model. Adopting the OSGi R4 specification and technology as the underlying foundation of CIShell has many advantages.

First, the OSGi specifications define and implement an elegant, complete, and dynamic component model, which fully supports the basic functionalities of the CIShell plug-in architecture. Its class-loading model is based on top of Java but adds modularization. While Java typically uses a single class path for all the classes and resources, the OSGi-loading module layer adds private classes for a module as well as controlled linking between modules. It is also responsible for handling multiple versions of the same classes—old and new applications can execute within the same virtual machine. The dynamic installing, starting, stopping, updating, and uninstalling of bundles is well defined in the OSGi specification and adopted by all CIShell bundles. OSGi also specifies and implements a service registry that takes care of the communication and collaboration among bundles. The service registry also supports the sharing of services between bundles. Note that services can appear and disappear or be updated at any moment in time.

Second, the CIShell specification and its applications can take advantage of a large number of OSGi services that have been defined and implemented on top

of the OSGi specification. These services include logging, preferences, http services (for running servlets), XML parsing, framework layering, declarative services, and many more. These services have been developed and can be obtained from several different vendors with different optimizations.

Third, given that the OSGi specification has component-oriented architecture and each service is defined abstractly and is independently implemented by different vendors, any CIShell application can choose a subset of services as needed and has no restriction to depend on the implementations of any particular vendor. Finally, by using OSGi, all CIShell algorithms become services that can be used in any OSGi-based system.

CIShell Specification

CIShell is an open source specification for the integration and utilization of datasets, algorithms, and computing resources. FIGURE 3 shows its highly modular and decentralized system architecture. It comprises a set of OSGi bundles (left) that upon start-up instantiate a set of OSGI services (right). An OSGi bundle is a plug-in, a collection of code that can be plugged and played as needed. The CIShell specification API itself is a bundle. It does not register any OSGi services but provides interfaces for data set/algorithm services, basic services, and application services.

The bundles are prioritized upon start of the application. The bundles with the highest priority are started first followed by bundles of second, third, etc. priority. Each bundle has a manifest file with a dependency list that states what packages, package versions, and other bundles it needs to run.

Each bundle can register zero or more services. The resulting set of OSGi services can be divided into data/algorithm services, basic services, application

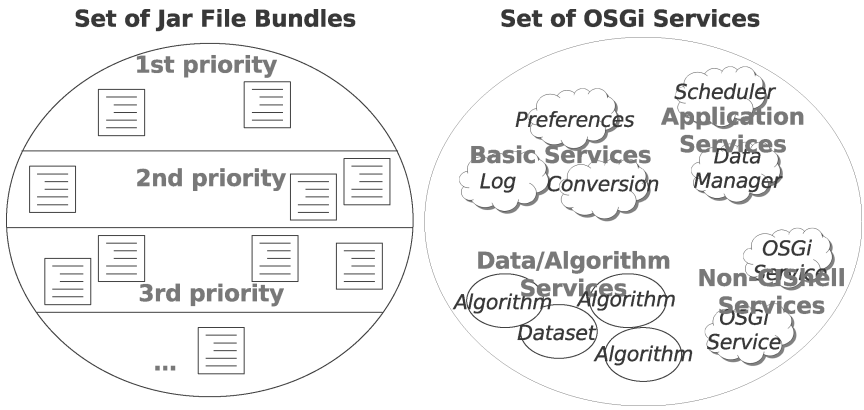

FIGURE 3. CIShell implementation.

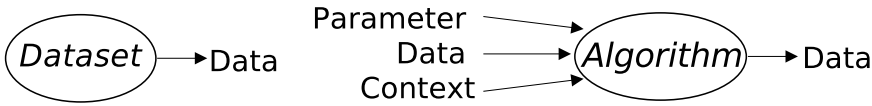

FIGURE 4. Input and output of data set and algorithm services.

services, and non-CIShell-specific services. A data set or algorithm that is written to adhere to the CIShell specification can be used in a wide variety of applications.

Data set services do not take any input data but serve data. *Algorithm services* typically take in user-provided parameters, data, and a context for getting basic services. They commonly return data after being executed. FIGURE 4 compares both service types. However, there are exceptions; modeling algorithms do not read in data, visualization algorithms do not write out data, and depending on how integrated a toolkit is, entire toolkits integrated as algorithms may not read in or write out data.

There are several *basic services* that an algorithm service can use. These services allow an algorithm to log information, get and set configuration options, create simple user interfaces for data input, and convert data to different formats. Access to these services is made available through the context passed to the algorithm. An algorithm that uses basic services exclusively and does not create its own GUI can be run remotely.

To shorten development time by application writers, several additional services have been specified in the CIShell specification. These *application services* help to manage data that algorithms create or to schedule algorithm execution time. More services will be available in later revisions of the specification.

USING CISHELL

CIShell creates a division of labor that allows algorithm writers to concentrate on writing algorithms, data set providers on providing data, application developers on creating applications, and researchers and practitioners on using the resulting applications to advance science and technology. CIShell aims to make each of these user groups as productive as possible by providing data set and algorithm integration templates and application solutions to developers and easy to use interfaces to users, as in FIGURE 5.

Data Set/Algorithm Providers: How to Integrate Data Sets and Algorithms

Researchers and practitioners interested to widely distribute their data sets and algorithms using CIShell need to produce OSGi bundles that can be run

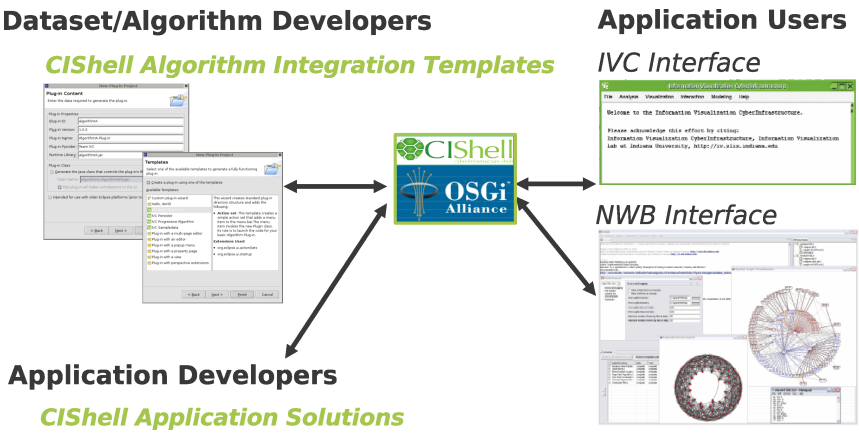

FIGURE 5. CIShell support for different kinds of users.

in any OSGi or CIShell application. To be a service in the OSGi framework, a bundle must provide three things: properties of the service (a set of key/value pairs), one or more interfaces, and an instantiated implementation of the interface(s). The service properties are very important as they are used to query the OSGi service registry for relevant algorithms, to learn where to place them in a menu, and to capture what meta-data should be shown to the user. Diverse templates are available for integrating data sets, non-Java-based algorithms (usually in the form of executables), and Java-based algorithms. Using these templates, developers can quickly produce OSGi bundles from existing data sets and code. A wizard-driven process guides the developer though a series of forms that lets them upload their data set/code and enter properties, such as its name, type, any input parameters, type of input and output data for algorithms, reference to any scholarly publications, etc. The information is then used to generate the code and documentation for a project that is used to build the jar bundle for distribution. Using this jar file, the data set–algorithm can now be run within any CIShell application.

Data sets and algorithms can also be integrated manually providing more control over how they interrelate to other services.

Application Tool Developers: How to Develop Applications

Application developers can leverage the OSGi and CIShell specification to compose a rich variety of custom applications. There are no specific templates set up for application development since the range of applications is huge. Instead, we provide well-documented CIShell reference implementations to teach

application development in an exemplar-driven way. Existing OSGi implementations are another valuable source of inspiration and technical guidance.

Application developers will need to have a good understanding of OSGi in addition to the algorithm specification defined by CIShell. However, they greatly benefit from the modularity and flexibility that OSGi provides. They will not have to worry about exactly how an algorithm is implemented nor how to integrate diverse data sets and algorithms into one application. Instead, they can focus on how to design novel applications that best serve the needs of their users.

To create an application from scratch, one usually starts with a base solution that provides the OSGi framework implementation, the CIShell specification, and an implementation of the defined CIShell services. From there, an application developer can use the OSGi and CIShell specification to create an application that uses the pool of algorithm services made available in a novel way. A forthcoming technical paper on the CIShell specification and existing implementations provides details on how this is done (Herr 2007).

Alternatively, a user can take a CIShell Base Solution (OSGi plus CIShell specification and services), a set of algorithms and data sets that were either developed in house or downloaded, and a rebranded version of the CIShell reference GUI to create an application that best fits a community. This approach was taken to implement the *Information Visualization Cyberinfrastructure* (IVC) and the *Network Workbench* (NWB). Both offer a menu-driven, branded interface that provides access to different sets of algorithms and data sets, a community web page, and individual documentation on how to use them. The IVC provides access to information visualization data sets and algorithms, whereas the NWB serves a considerably larger network science community. Using one base solution to implement two very different CIs has saved us enormous amounts of time and effort in terms of design, implementation, and maintenance of the CIs.

Application Users: How to Use Applications Built Using CIShell

Researchers, educators, and practitioners are the main beneficiaries of the "empty shell" approach to the sharing of data sets, algorithms, and computing resources.

Users of CIShell applications can easily customize the "filling" of their applications—replacing old with new, more efficient code, or keeping all versions around for comparison. They can decide to either download the filling commonly used by their respective community or exactly the data sets and algorithms they need.

The CIShell specification makes possible the design of sociotechnical infrastructures that provide easy access to any data set, algorithm, tool, or compute resource in existence. Not knowing about an existing data set or algorithm,

the continuous reimplementation of the very same algorithm, or the frustration caused by incompatible file formats and algorithms that run on different platforms, etc. becomes history. The more people adhere to the CIShell specification the more data sets, algorithms, other services, and applications will be available for usage and supplied by different vendors.

REFERENCE IMPLEMENTATIONS

The CIShell *base solution* combines the Eclipse (http://www.eclipse.org) Equinox OSGi R4 reference implementation, several service bundles from the Eclipse, the CIShell specification bundle, and some reference CIShell service implementation bundles into an application that can be run. It is a bare-bones distribution that has no real interface but can be filled with other bundles to provide a user interface, algorithms, data sets, and whatever else is needed in the application to satisfy users' needs. An application does not have to use this solution to make an end user application, but it is provided as a basic solution on which application developers and other advanced users can build on.

The *client-server middleware* (an application that another application can use for added functionality) application was developed as a proof of concept to show how a remote client-server could be implemented. It uses web service technology so that a client application can connect to the remote server and use the algorithms and data sets available there. In praxis, services on the server show up as proxy services on the client. They can be executed (on the server) by an application running on the client without any special handling code. If new services are added to the server, the middleware detects them, and makes them usable in all applications running on the client. This technique will be further extended in the future to create a web front-end and a peer-to-peer middleware solution.

The *scripting application* was developed as a proof of concept to show how a scripting engine could be easily integrated into a CIShell-based system. When the bundle holding this application is started, it opens a console so that a user can enter scripting code to access the OSGi service registry, find algorithms, and use them. While not as simple as a GUI, there are many users who prefer this level of interaction. A scripting interface could also be used to create a middleware application that enables other applications to transfer and execute code. By keeping a log of all user actions as scripting code, any sequence of user actions can be saved, shared, and rerun as needed.

A *GUI application* was developed as an easy to use, menu-driven interface that can be used by itself or branded for custom end-user applications. The GUI is built using the eclipse-rich client platform (RCP), which is built on OSGi. Its look and feel benefits from lessons learned in our previous CI development efforts, though with new code and some new features. This GUI is used by the CIs discussed hereafter.

INFORMATION VISUALIZATION CYBERINFRASTRUCTURE

IVC started as a software repository project in 2000. Its goal was and is to provide access to a comprehensive set of data sets, algorithms, and computing resources but also educational materials that ease the utilization of data mining and information visualization algorithms. The project's web page is at http://iv.slis.indiana.edu.

Katy Börner, Yuezheng Zhou, and Jason Baumgartner implemented the very first algorithms (Börner and Zhou 2001). In summer 2003, Jason Baumgartner, Nihar Sheth, and Nathan J. Deckard led a project to design an XML toolkit that enables the serialization and parallelization of commonly used data analysis and visualization algorithms (Baumgartner *et al.* 2003). In summer 2004, Shashikant Penumarthy and Bruce W. Herr master-minded the IVC framework. Josh Bonner and Laura Northrup, Hardik Sheth, and Jeegar Maru were involved in the implementation of the IVC and the integration of algorithms. Maggie B. Swan and Caroline Courtney were of invaluable help for the design, proof reading, and validation of the diverse online documentations. In early 2005, James Ellis, Shashikant Penumarthy, and Bruce Herr revised the IVC to use eclipse-RCP as the underlying plug-in model and GUI. Later that year, the underlying core of the IVC was separated even further from the application resulting in the Information Visualization Cyberinfrastructure Software Framework (IVCSF). In early 2006, work was started on the successor to the IVCSF, CIShell, as described in this article.

Over the last 7 years, this effort has been supported by the School of Library and Information Science, Indiana University's High Performance Network Applications Program, a Pervasive Technology Lab Fellowship, an Academic Equipment Grant by SUN Microsystems, and an SBC (formerly Ameritech) Fellow Grant, as well NSF grants DUE-0333623 and IIS-0238261.

The major components of the IVC are databases, computing resources, software, and learning modules as shown in FIGURE 6. The CIShell specification is used to integrate diverse data sets (e.g., time series, documents, matrices, networks, geospatial data) and algorithms (e.g., preprocessing, analysis, modeling, visualization, interaction).

Since 2000, the repository/CI has been used to teach the *Information Visualization* class at Indiana University. Starting 2003, it was also used in Börner's *Data Mining and Modeling* class. Since its debut on Sourceforge.net in June 2004, it has been downloaded more than 5000 times—mostly by institutions, organizations, and companies in United States, Europe, and Asia. Other InfoVis tool(kits) exist, such as:

1. Jean-Daniel Fekete's InfoVis Toolkit (Fekete 2004)
   http://ivtk.sourceforge.net
2. Visualization Toolkit software by Kitware Inc.
   http://www.kitware.com/vtk.html

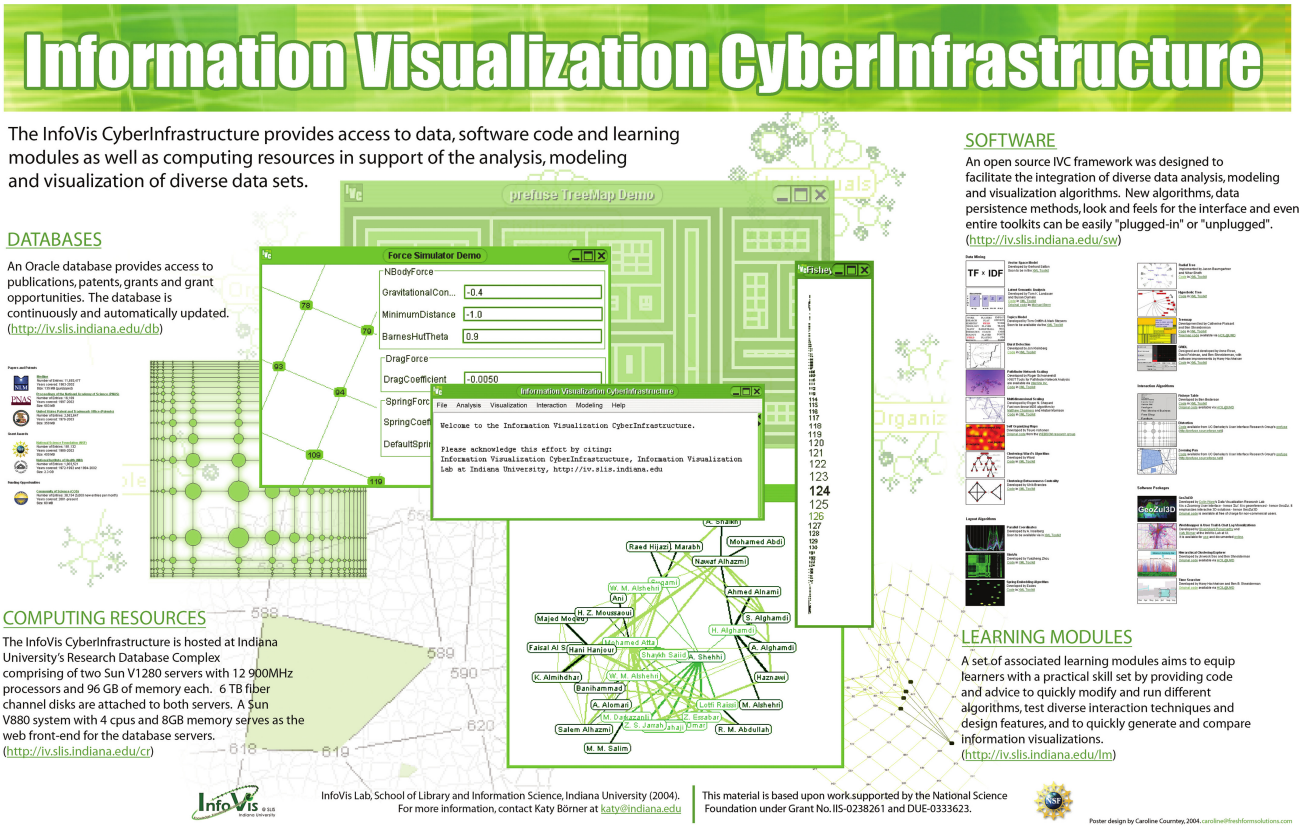

FIGURE 6. Components of the IVC.

3. the CAIDA visualization tools accessible at http://www.caida.org/tools
4. and the many others listed by http://vw.indiana.edu/ivsi2004/

CIShell differs from them in that it aims to ease the integration of new data sets and software algorithms by diverse noncomputer scientist developers. In the near future, there will also be a means to contribute algorithm descriptions and learning modules.

NETWORK WORKBENCH

The NWB is a network analysis, modeling, and visualization toolkit for biomedical, social science, and physics research. It is generously funded by a $1.1 million NSF award for a 3-year duration: September 2005–August 2008. Investigators are Katy Börner, Albert-Laszlo Barabási, Santiago Schnell, Alessandro Vespignani, Stanley Wasserman, and Eric Wernert. The software team is lead by Weixia (Bonnie) Huang and comprises CIShell developer Bruce Herr and algorithm developers Ben Markines, Santo Fortunato, and Cesar Hidalgo. The project's web page is at http://nwb.slis.indiana.edu.

The project will design three different applications: A NWB tool for use by network science researchers and practitioners, an educational web service that teaches biologists and others the basics of network science, and a mapping

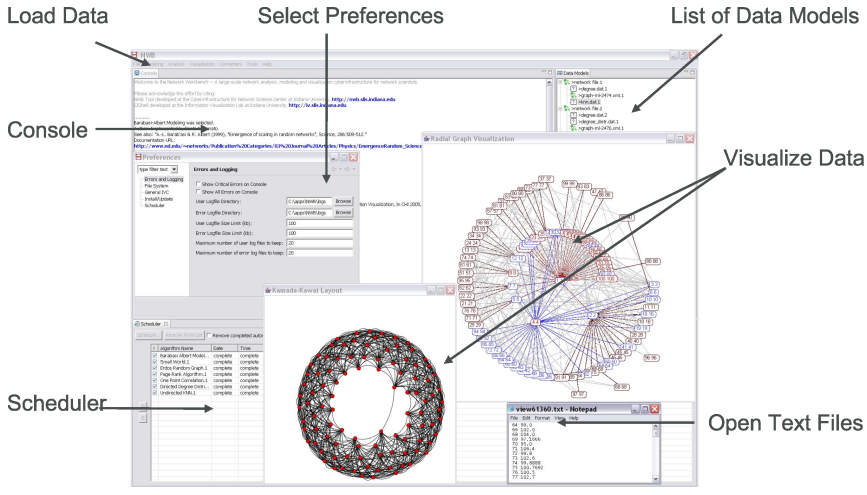

FIGURE 7. Interface of the network workbench tool.

science service that the general public can use to explore and make sense of mankind's collective scholarly knowledge.

The NWB tool uses the stand-alone CIShell GUI application and resembles the IVC. However, it has a different branding and "filling." A snapshot of the NWB tool interface is shown in FIGURE 7. Major parts are labeled.

A user can load data via the menu-driven interface or generate a network data set. She or he can analyze and visualize a data set using a wide variety of different complementary and alternative algorithms. All used data sets are listed in the right window of the interface. All user actions as well as citations for selected algorithms are printed in the console.

A generic *NWB* data format was defined to support the storage and processing of million node graphs. Using the NWB converter plug-ins, the tool can load, view, and save a network from/to a *NWB* data format file. Although the *NWB* data model is the fundamental data structure, other data models, such as the *Prefuse Graph* model and *Matrix* model, have been developed and integrated into the NWB tool.

CIShell algorithms written in diverse programming languages can be integrated and used. CIShell templates are used to integrate code written in Java, C++, or FORTRAN.

Among others, JUNG and Prefuse libraries have been integrated into the NWB as plug-ins. After converting the generated *NWB* data model into *JUNG Graph* and *Prefuse Graph* data model, NWB users can run JUNG and Prefuse graph layouts to interactively explore visualizations of their networks. NWB also supplies a plug-in that invokes the XMGrace application for plotting data analysis results.

The NWB tool has a number of unique features: It is open source, highly flexible, easily extendable, and scalable to very large networks. While some of the most powerful network tools, for example, Pajek and UCINET (Borgatti *et al.* 2002) run on Windows, the NWB tool can be compiled for multiple operating systems including Windows, Linux, Unix, and Mac OS. Another set of tools, for example, Cytoscape (Shannon *et al.* 2003) and TopNet (Yu *et al.* 2004) were designed specifically for the study of biological networks. However, we believe that the true value of network science will come from a cross-fertilization of approaches, algorithms, and technologies developed in different disciplines. Both, the NWB tool and Cytoscape use open source, plug-and-play software architectures. While Cytoscape requires all plug-ins to work with their internal data model, the NWB tool defines and uses an efficient NWB data format, but also provides support for other data formats that algorithm developers might use. GEOMI (Xu *et al.* 2006) and Tulip (Auber 2003) are both tools that support the visualization of huge graphs. GEOMI uses WilmaScope, a 3D graph visualization system. Tulip uses Open GL as the rendering engine. Both have a plug-and-play architecture, but due to sparse documentation it is very difficult for other researchers to evaluate the framework or contribute plug-ins.

DISCUSSION AND FUTURE WORK

The CIShell specification is a culmination of 7 years of toolkit development. The design has gone under several names: InfoVis Repository (IVR), IVC and specifications: IVCSF and CIShell. However, the goal is still the same, to improve the diffusion of data sets, algorithms, and applications from those that develop them to those that benefit from their usage. We have made major progress from the pre-IVR days when our programs were dispersed throughout differing operating systems and file systems and had to be run manually from the command line. Today, the CIShell specification provides a means to design highly modular, highly decoupled, very flexible, and powerful applications and infrastructures.

We are in the process of creating more reference implementations, such as a web front-end, peer-to-peer sharing, and workflow engines. The reference implementations that are already available will be extended and hardened to convert them from proof of concept to robust software applications.

A key for client–servers, web front-ends, and peer-to-peer sharing will be in a formal definition of the web service model using web services definition language (WSDL). Creation of a standard by which networked CIShell instances communicate will help these applications to easily cooperate. Furthermore, this will allow any software application (CIShell compliant or not) to be able to connect to a running CIShell instance with relative ease and security using standard web service techniques.

Another addition will apply the idea of "semantic association networks" (Börner 2006). The published papers, their authors, as well as data sets and algorithms used will all be interlinked. The linkages can then be traversed in service of superior knowledge access and management. For example, all authors who used a certain data set or algorithm can be retrieved or all algorithms that have ever been used to analyze a certain data set can be identified.

We will continue to promote and foster an atmosphere of cooperation, communication, and open access. While we will use CIShell for our own application development and usage, its value will increase with the number of people using it. We will continue to give talks and tutorials and organize workshops that introduce algorithm and application developers as well as users to CIShell and its reference implementations. The feedback and the buy-in from a growing development team and user group will be essential to the design of the best possible CIShell.

Last but not least, we would like to point out that we did benefit enormously from related efforts, such as TeraGrid (http://www.teragrid.org), R, GeoVISTA Studio (Takatsuka and Gahegan 2002), and many others. Although these systems seem to have little in common, they share the vision behind CIShell—to ease the dissemination of data sets, algorithms, and computing resources to a broad user base.

TeraGrid (http://www.teragrid.org) is an NSF-sponsored grid computing initiative that provides scientists access to massively distributed computing power and data storage for research. It supports the design of highly scalable services and computing infrastructures. However, the integration of new data sets, services, and the design of online portals is typically done by computer scientists in close collaboration with the application holders. In contrast, CIShell focuses on ease of use for algorithm/application developers and users and is mostly used for applications that do not require access to highly scalable and distributed infrastructures. However, it can be used to integrate services that support sharing of distributed data sets and services running on parallel computing resources. Note that CIShell can be interlinked with grid-based infrastructures by using the Web service protocol under development. This would make grid services available via CIShell applications or provide CIShell services, for example, data analysis, modeling, or visualization services, to grid applications.

R is a language and environment for statistical computing and graphics. It allows for addition of new packages and has a vibrant developer and user community. It is very much a text based system built around the language and integrated algorithms, but there are some externally made GUIs available. What separates R from CIShell is that R's integration methods are nontrivial and become truly difficult if code is not written in C, C++, or Fortran. CIShell supports any programming language and provides templates for data set and algorithm integration. To us, R appears to have a steeper learning curve in both using the software and integrating algorithms and packages.

GeoVISTA Studio is an open source, programming-free development environment that allows users to quickly build applications for geocomputation and geographic visualization. It uses JavaBeans to support the plug-and-play of different algorithms. A visual programming interface is employed to support the integration of new code.

To our knowledge, there exists no other effort that attempts to build an "empty shell" that supports the easy integration and utilization of data sets and code; runs on all common platforms; in a stand-alone, client-(Web)server or peer-to-peer fashion; is highly decoupled; and builds on industry standards to create a powerful, simple to use, yet highly flexible algorithm environment. It is our sincere hope that the CIShell specification will be widely adopted and used to create highly flexible and usable CIs in service of international and interdisciplinary innovation and progress.

REFERENCES

ATKINS, D.E., K.K. DROGEMEIER, S.I. FELDMAN, H. GARCIA-MOLINA, M.L. KLEIN, D.G. MESSERSCHMITT, P. MESSIAN, J.P. OSTRIKER, and M.H. WRIGHT. 2003. Revolutionizing science and engineering through cyberinfrastructure: report of the National

Science Foundation blue-ribbon advisory panel on cyberinfrastructure. Arlington, VA: National Science Foundation.

AUBER, D. 2003. Tulip: A huge graph visualisation framework. Pp. 105–126 in P. Mutzel, and M. Jünger (eds.), *Graph Drawing Softwares, Mathematics and Visualization*. Heidelberg: Springer-Verlag.

BATAGELJ, V., and A. MRVAR. 1998. Pajek – Program for large network analysis. *Connections 21*(2), 47–57.

BAUMGARTNER, J., K. BÖRNER, N.J. DECKARD, and N. SHETH. 2003. An XML Toolkit for an information visualization software repository. IEEE Information Visualization Conference, Poster Compendium. 72–73.

BERMAN, F., A.J.G. HEY, and G.C. FOX. 2003. *Grid Computing: Making The Global Infrastructure a Reality*. New York: Wiley Press.

BORGATTI, S.P., M.G. EVERETT, and L.C. FREEMAN. 2002. *UCINET for Windows: Software for Social Network Analysis*. Harvard: Analytic Technologies.

BÖRNER, K. 2006. Semantic association networks: using semantic web technology to improve scholarly knowledge and expertise management. Pp. 183–198 in V. Geroimenko and C. Chen (eds.), *Visualizing the Semantic Web*. Springer Verlag.

BÖRNER, K., and Y. ZHOU. 2001. A Software Repository for Education and Research in Information Visualization. Fifth International Conference on Information Visualisation, London, England: IEEE Press. 257–262.

BÖRNER, K., C. CHEN, and K. BOYACK. 2003. Visualizing Knowledge Domains. Pp. 179–255 in B. Cronin (ed.), *Annual Review of Information Science and Technology*, *Vol. 37*. Medford, NJ: Information Today, Inc./American Society for Information Science and Technology.

BÖRNER, K., S. SANYAL, and A. VESPIGNANI. 2007. Network Science. Pp. 41 in B. Cronin (ed.), *Annual Review of Information Science & Technology*. Medford, NJ: Information Today, Inc./American Society for Information Science and Technology.

FEKETE, J.-D. 2004. The Infovis Toolkit Proceedings of the 10th IEEE Symposium on Information Visualization, IEEE Press. 167–174.

HEER, J., S.K. CARD, and J.A. LANDAY. 2005. Prefuse: a toolkit for interactive information visualization. *CHI 2005-Proceedings* 421–430.

HERR, B. 2007. CIShell: Cyberinfrastructure Shell. *A Novel Algorithm Integration Framework*, *forthcoming*.

HUISMAN, M., and M.A.J.V. DUIJN. 2003. StOCNET: software for the statistical analysis of social networks. *Connections 25*(1), 7–26.

IHAKA, R., and R. GENTLEMAN. 1996. R: a language for data analysis and graphics. *Journal of Computational and Graphical Statistics 5*(3), 299–314.

MANE, K.K., and K. BÖRNER. 2004. Mapping topics and topic bursts in PNAS. *Preceedings of the National Academy of Sciences of the United States of America 101*(Suppl 1), 5287–5290.

O'MADADHAIN, J., D. FISHER, S. WHITE, and Y. BOEY. 2003. The JUNG (Java Universal Network/Graph) framework. *Technical Report, UC Irvine*.

SCHVANEVELDT, R.W. 1990. *Pathfinder Associative Networks: Studies in Knowledge Organization*. Norwood, NJ: Ablex Publishing.

SHANNON, P., A. MARKIEL, O. OZIER, N.S. BALIGA, J.T. WANG, D. RAMAGE, N. AMIN, B. SCHWIKOWSKI, and T. IDEKER. 2003. Cytoscape: a software environment for integrated models of biomolecular interaction networks. *Genome Research 13*, 2498–2504.

TAKATSUKA, M., and M. GAHEGAN. 2002. GeoVISTA Studio: a codeless visual programming environment for geoscientific data analysis and visualization. *The Journal of Computers and Geosciences 28*(10), 1131–1144.

XU, K., S.-H. HONG, M. FORSTER, N.S. NIKOLOV, J. HO, D. KOSCHUTZKI, A. AHMED, T. DWYER, X. FU, C. MURRAY, R. TAIB, and A. TARASSOV. 2006. GEOMI: geometry for maximum insight. Pp. 468–479 in P. Healy, and N.S. Nikolov (eds.), *Proceedings Graph Drawing*. Ireland: Limerick.

YU, H., X. ZHU, D. GREENBAUM, J. KARRO, and M. GERSTEIN. 2004. TopNet: a tool for comparing biological sub-networks, correlating protein properties with topological statistics. *Nucleic Acids Res 32*, 328–337.