Herr, Bruce W., Huang, Weixia, Penumarthy, Shashikant, and Börner, Katy. (2007).
Designing Highly Flexible and Usable Cyberinfrastructures for Convergence.
In Progress in Convergence – Technologies for Human Wellbeing, vol. 1093,
ed. William S. Bainbridge and Mihail C. Roco, 161-179. Boston: Annals of the New York Academy of Sciences.

Designing Highly Flexible and Usable Cyberinfrastructures for Convergence

Bruce Herr, Weixia Huang, Shashikant Penumarthy, Katy Börner
School of Library and Information Science, Indiana University
Wells Library, 10th Street & Jordan Avenue
Bloomington, IN 47405, USA
[bherr|huangb|sprao|katy]@indiana.edu

ABSTRACT

Innovation and progress in most areas of science requires access 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, Drogemeier, Feldman, Garcia-Molina, Klein, Messerschmitt, Messian, Ostriker and Wright 2003). Today, some CIs provide access to thousands of interlinked experts, services, federated datasets, 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 desperate interest to save lives, secure important technical infrastructures, discover universal laws, etc. 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, datasets, algorithms, and technologies across multiple disciplines and business sectors leading to a more integrative science. This chapter presents the results of a seven-year long quest into the development of a 'dream tool' for our research in scientometrics (Börner, Chen and Boyack 2003) and more recently, network science (Börner, Sanyal and Vespignani in press). The results are two CIs: 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 datasets 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 like 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 datasets, algorithms, and compute resources across national and disciplinary boundaries.

Keywords

Cyberinfrastructure, OSGi, Plug-in, Data Models, Analysis, Network Science, Scientometrics, Usability, Flexibility, Extensibility.

1. Introduction

Fifty years back in time, scientists did not use any computer to conduct their research. Today, science without computation is unthinkable in almost all areas of science. Scientists 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 datasets and to visualize their 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 is 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. Any code has to match with the internal data format and plug-in system. It is also 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 provide access to 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, Fisher, White and Boey 2003), and Prefuse (Heer, Card and Landay 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 datasets by developers or provides a menu driven, easy to use interface for the application user.

Grid computing (Berman, Hey and Fox. 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 over a network using open standards. It also supports the composition of applications and services, workflow design, scheduling, and execution management. There are several challenges to overcome when using grid computing. To fully take advantage of the grid, one's code must be (re)written to take advantage of running in parallel on a cluster of potentially very different operating systems and environments, which appears to be a major challenge for most non-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 and application developers and application users. Many algorithm developers are searching for possible utilities 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 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 datasets that are implemented as services. Data, algorithms and additional CIShell services such as graphical user interfaces, schedulers, logging services, etc. 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 datasets and services needed in a classroom setting. Hence, CIShell creates a bridge between algorithm developers, application developers, and their users as shown in Figure 1.

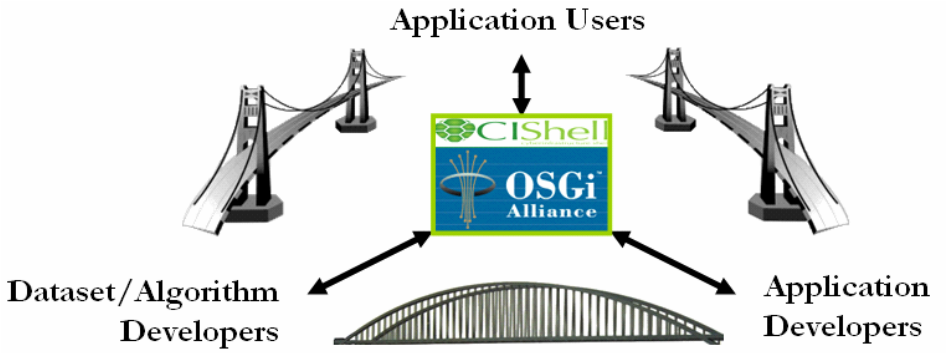

Figure 1. CIShell as a bridge among users, dataset and algorithm developers, and application developers.

The remainder of this chapter is organized as follows: Section 2 details the needs of the user groups CIShell aims to serve. Section 3 describes the inner workings of CIShell on an abstract level. Section 4

explains how CIShell is used by the three different user groups. Section 5 and 6 introduce diverse reference implementations. The chapter concludes with a discussion and an outlook of future work.

2. What Users Want

2.1 What Algorithm and Application Developers Want

Today, it is not only computer scientists which develop and create novel datasets and algorithms but also biologists, physicist, social scientists, etc. In many cases, the datasets 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 re-implemented uncountable times – a true waste of life time 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 datasets and algorithms. To make things worse, datasets come in diverse formats and algorithms are implemented in very different languages.

The diffusion of high quality datasets and novel algorithms would greatly benefit from a means to easily integrate and utilize existing datasets and algorithms.

There is also a need for the design of tools that provide access to exactly those datasets 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 custom designed tools.

2.2 What Application Users Want

Analyzing and making sense of datasets frequently involves multiple steps, such as sampling, cleaning, analyzing, and sometimes visualizing for means of communication. For means of illustration, Figure 2 shows the diverse datasets (given in italics and underlined) and processing steps involved in a scientometric study (Shiffrin and Börner 2004) aiming at the identification of the topic coverage of a document dataset. 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 (in the same line of the original list of all documents) the 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. Given this, almost all terms co-occur with each other in at least one of the many documents. Layout optimization is needed. In a first attempt, we might apply a threshold 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. 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.

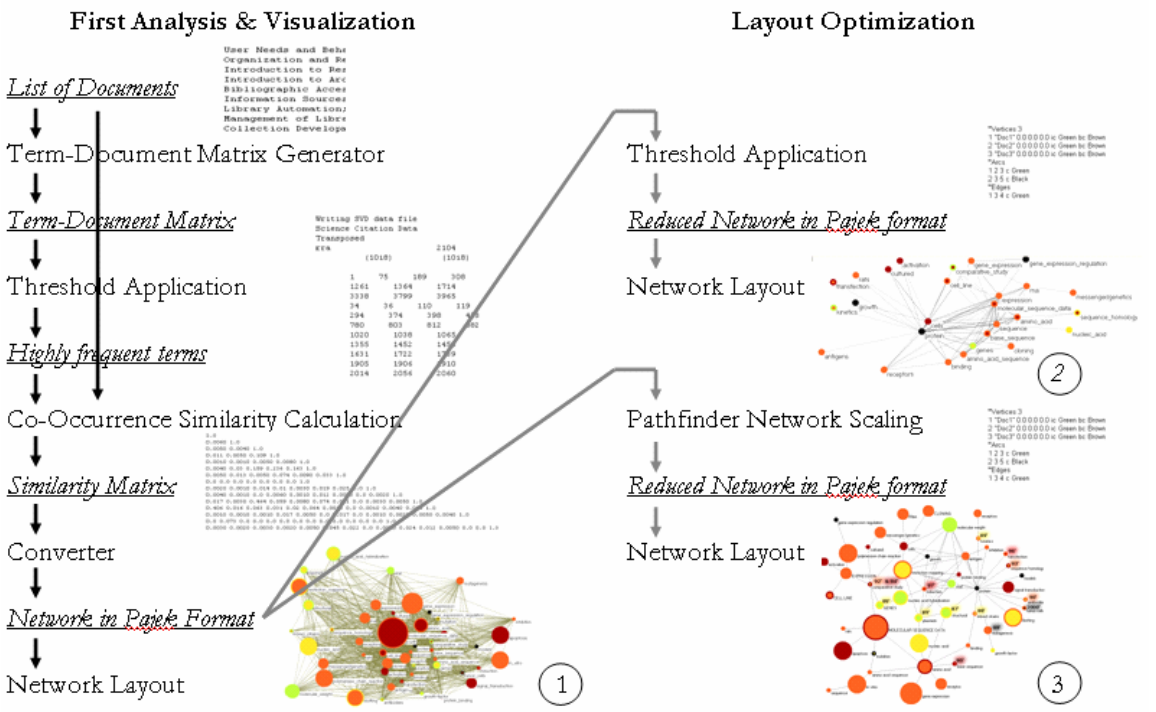

Figure 2. Sample acquisition, analysis, and visualization of a topic network.

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 datasets, algorithms, and existing tools via a graphical user interface (GUI) that also provides logging, scheduling, and other services.

3. CIShell Design

3.1 Overview

The Cyberinfrastructure Shell (CIShell) is an open source, community-driven specification for the integration and utilization of datasets, 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 Open Services Gateway Initiative (OSGi) specification, see also section 3.2. By leveraging OSGi, we gain access to a large amount of industry standard code and know-how that would take years to re-invent/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, see Figure 4 and sections 3.3, 4, and 6.

CIShell applications can be deployed as distributed data and algorithm repositories, stand alone applications, peer-to-peer architectures, and server-client architectures, see Figure 3 and section 5.

Subsequently, we describe the OSGi and CIShell specifications in detail.

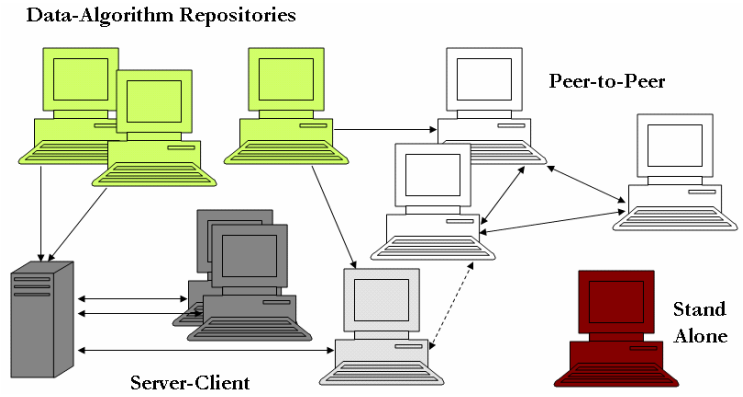

Figure 3: CIShell applications can be deployed as distributed data and algorithm repositories, stand alone applications, peer-to-peer architectures, and server-client architectures

3.2 Open Services Gateway Initiative (OSGi) Specification

The Open Services Gateway Initiative (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 seven 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 plugin model. Adopting the OSGi R4 specification and technology as the underlying runtime and the foundation of CIShell has many advantages.

Firstly, 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 classpath 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.

Secondly, 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.

Thirdly, 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.

3.3 Cyberinfrastructure Shell (CIShell) Specification

CIShell is an open source specification for the integration and utilization of datasets, algorithms, and computing resources. Figure 4 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 plugin, 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 dataset/algorithm services, basic services, and application services.

The bundles are prioritized upon start of the application. The bundles of 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 services, and non-CIShell specific services. A dataset or algorithm that is written to adhere to the CIShell specification can be used in a wide variety of applications.

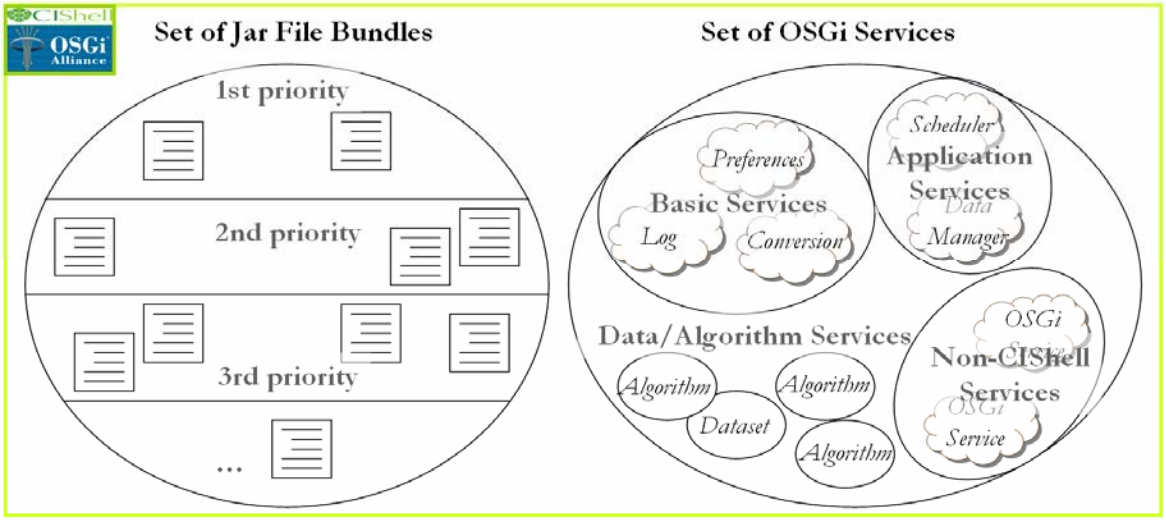

Figure 4: CIShell implementation comprising a set of OSGi bundles (left) that register a set of OSGi services (right).

Dataset and Algorithm Services. Dataset services (cf. Figure 5, left) do not take any input data but serve data. Algorithm services (cf. Figure 5, right) typically take in user-provided parameters, data, and a context for getting basic services. They return data after being executed. However, there are exceptions; modeling algorithms do not read in data, visualization algorithms do not write out data, and depending on how integrated the toolkit is, entire toolkits integrated as algorithms may not read in or write out data.

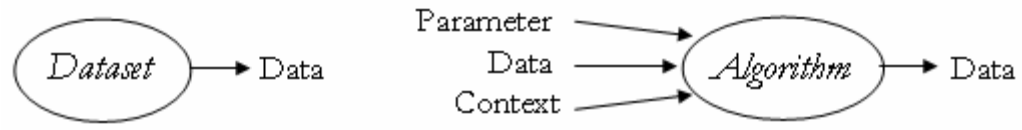

Figure 5: Input and output of dataset and algorithm services

Basic Services. 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.

Application Services. 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.

4. Using CIShell

4.1 Overview

CIShell creates a division of labor that allows algorithm writers to concentrate on writing algorithms, dataset providers on providing data, application developers on creating applications, and researchers and practitioners to use the resulting applications to advance science and technology. CIShell aims to make each of these user groups as productive as possible by providing dataset and algorithm integration templates and application solutions to developers and easy to use interfaces to users, see also Figure 6.

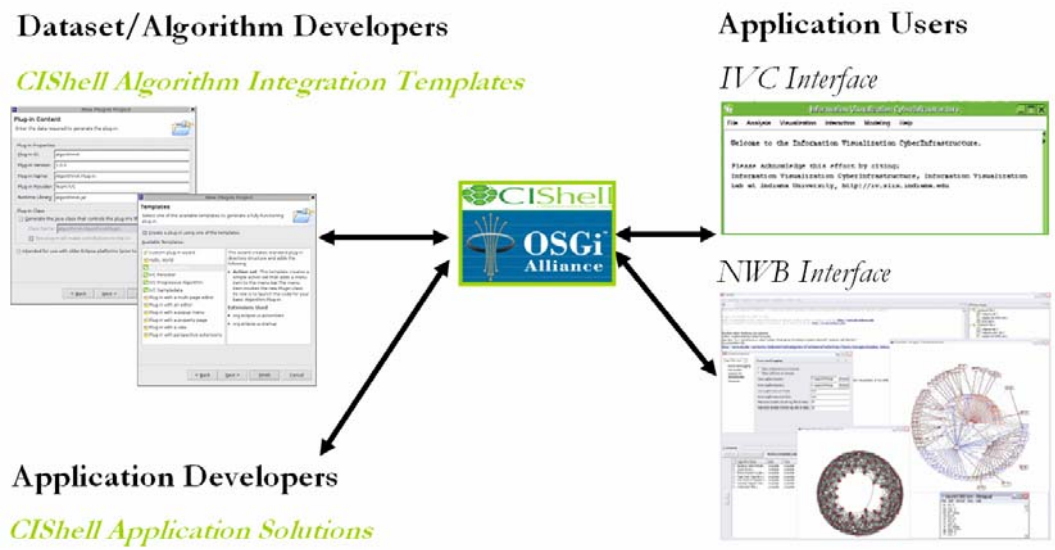

Figure 6: CIShell support for the integration of datasets and algorithms and for tool development.

Subsequently, we discuss what steps are necessary by the different users to benefit and contribute to the CIShell specification and its existing and future implementations.

4.2 Dataset/Algorithm Providers: How to Integrate Datasets and Algorithms

Researchers and practitioners interested to widely distribute their datasets and algorithms using CIShell need to produce OSGi bundles that can be run 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 datasets, 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 datasets and code. A wizard driven process guides the developer though a series of forms that lets them upload their dataset/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, see also Figure 4 (left), the dataset/algorithm can now be run within any CIShell application.

Datasets and algorithms can also be integrated manually providing more control over how they interrelate to other (basic) services.

4.3 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 datasets and algorithms into one application. Instead they can focus on how to design novel applications that best serve the needs of their users.

A CIShell application can be developed from scratch or by combining a set of algorithms (pre-made or custom built), toolkits integrated as algorithms, datasets, and CIShell applications, brand them in a certain design style, and make them available for use. Note that some applications act as middleware and as such may not be packaged by itself but are always contained within a larger application.

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 utilizes 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 2006).

Alternatively, a user can take a CIShell Base Solution (OSGi plus CIShell specification and services), a set of algorithms and datasets that were either developed in house or downloaded, and a re-branded version of the CIShell reference graphical user interface (GUI) to create an application that best fits a community. This approach was taken to implement the *Information Visualization Cyberinfrastructure* (IVC) (cf. section 6.1) and the *Network Workbench* (cf. section 6.2). Both offer a menu driven, branded interface that provides access to different sets of algorithms and datasets, a community web page, and individual documentation on how to use them. The IVC provides access to information visualization datasets and algorithms, whereas the Network Workbench 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.

4.4 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 datasets, 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 datasets and algorithms they need.

The CIShell specification makes possible the design of socio-technical infrastructures that provide easy access to any dataset, algorithm, tool, or compute resource in existence. Not knowing about an existing dataset or algorithm, the continuous re-implementation 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 datasets, algorithms, other services, and applications will be available for usage and supplied by different vendors.

Section 5 and 6 introduce diverse reference implementations designed for different user groups.

5. Reference Implementations

5.1 Base Solution

The CIShell Base Solution combines the Eclipse (http://www.eclipse.org) Equinox OSGi R4 reference implementation, several service bundles from the Eclipse and Knopflerfish (http://www.knopflerfish.org) projects, 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, datasets, 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.

5.2 Client–Server Middleware

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, see also Figure 3. It uses web service technology so that a client application can connect to the remote server and use the algorithms and datasets 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.

5.3 Scripting Application

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 re-run as needed.

5.4 GUI Application

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 cyberinfrastructure development efforts, though with entirely new code and some new features. This GUI is used by the CIs discussed in the next section.

6 Cyberinfrastructures Using CIShell

6.1 Information Visualization Cyberinfrastructure (IVC)

The 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 datasets, 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 lead a project to design an XML toolkit that enables the serialization and parallelization of commonly used data analysis and visualization algorithms (Baumgartner, Börner, Deckard and Sheth 2003). In summer 2004, Shashikant Penumarthy and Bruce W. Herr masterminded 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 graphical user interface. 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 paper.

Over the last six 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 National Science Foundation grants DUE-0333623 and IIS-0238261.

The major components of the IVC are databases, computing resources, software, and learning modules as shown in Figure 7.

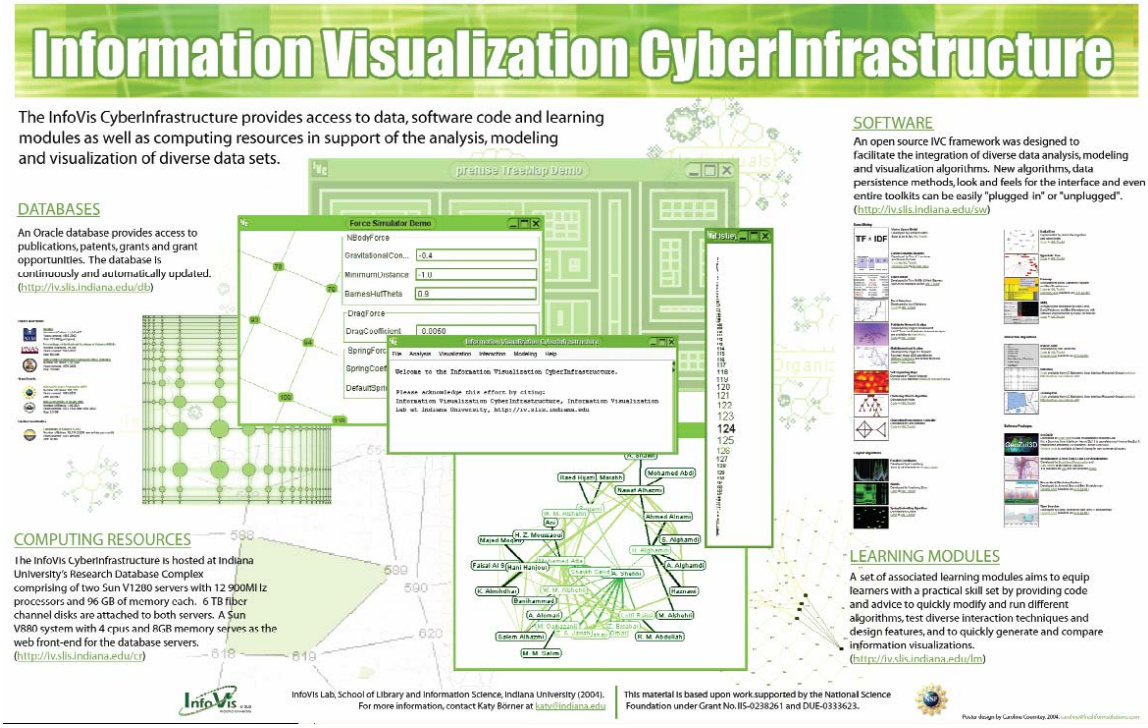

Figure 7. Components of the Information Visualization Cyberinfrastructure

The CIShell specification is used to integrate diverse datasets (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.org in June 2004, it has been downloaded more than 5,000 times – mostly by institutions, organizations, and companies in U.S., Europe, and Asia.

CIShell differs from other InfoVis tool(kits) such as

- Jean-Daniel Fekete's InfoVis Toolkit (Fekete 2004) at http://ivtk.sourceforge.net,
- Visualization Toolkit software by Kitware Inc. http://www.kitware.com/vtk.html
- the CAIDA visualization tools accessible at http://www.caida.org/tools, and
- the many others listed from http://vw.indiana.edu/ivsi2004/

in that it aims to ease the integration of new datasets and software algorithms by diverse non-computer scientist developers. In the near future, there will also be a means to contribute algorithm descriptions and learning modules.

6.2 Network Workbench

The *Network Workbench* (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

three year duration: Sept. 2005 - Aug. 2008. Investigators are Katy Börner, Albert-Laszlo Barabasi, 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 science service that the general public can use to explore and make sense of mankind's collective scholarly knowledge.

The NWB tool uses the standalone 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 8. Major parts are labeled.

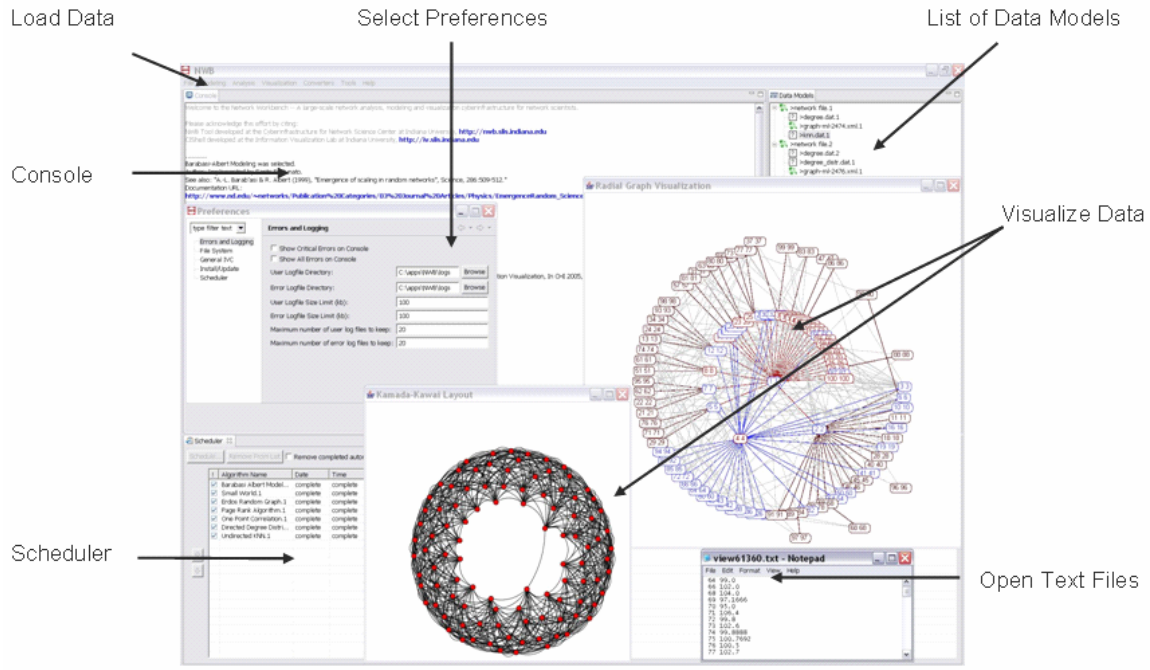

Figure 8. Interface of the Network Workbench Tool

A user can load data via the menu driven interface or generate a network dataset. S/he can analyze and visualize a dataset using a wide variety of different complementary and alternative algorithms. All used datasets 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 discussed in section 4.2 are used to integrate code written in Java, C++, or FORTRAN. Additional templates are under development

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 ther 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, e.g., Pajek and UCINET (Borgatti, Everett and Freeman 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, e.g., Cytoscape (Shannon, Markiel, Ozier, Baliga, Wang, Ramage, Amin, Schwikowski and Ideker 2003) and TopNet (Yu, Zhu, Greenbaum, Karro and Gerstein 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, Hong, Forster, Nikolov, Ho, Koschutzki, Ahmed, Dwyer, Fu, Murray, Taib and Tarassov 2006) and Tulip (Auber 2003) are both tools that support the visualization of huge graphs. GEOMI uses the 3D graph visualization system WilmaScope and Tulip uses Open GL as the rendering engine. Both have a plug-and-play architecture. Although they claim that the framework is extendable for new algorithms, the plug-in APIs, developer's guides and detail architecture documentation is not available. So it is very difficult for other researchers to evaluate the framework or contribute any plug-ins.

To our knowledge, none of the above mentioned tools utilize any industry standards. The NWB is the only one that benefits from the OSGi industry standard and CIShell specification.

7. Discussion & Future Work

The CIShell specification is a culmination of seven years of toolkit development. The design has gone under several names: InfoVis Repository (IVR), IVC and specifications: IVC Software Framework (IVCSF) and CIShell, but the goal is still the same, to improve the diffusion of datasets, 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, and very flexible 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.

The specification also mentions two key areas that are not addressed by the architecture: brushing & linking and remote visualization. Brushing and linking can be achieved with shared data models, but there is currently no data-model-agnostic way to do so. In future revisions, we hope to address both of these use cases.

Another addition will apply the idea of 'semantic association networks' (Börner 2006). The published papers, their authors, as well as datasets 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 which used a certain dataset or algorithm can be retrieved or all algorithms that have ever been used to analyze a certain dataset can be identified.

We will continue to promote and foster an atmosphere of cooperation, communication, and open access. While we will utilize 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 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 three seem to have little in common, they share the vision behind CIShell – to ease the dissemination of datasets, 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 datasets, 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 datasets 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, e.g., 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 as well. What separates R from CIShell is that R's integration methods are non-trivial and become truly difficult if code is not written in C, C++, or Fortran. CIShell supports any programming language and provides templates for dataset 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 utilizes JavaBeans to support the plug-and-play of different algorithms. A visual programming interface is employed to support the integration of new code. It is a powerful system, but it can act only as a workflow engine.

To our knowledge, there exists no other effort that attempts to build an ‘empty shell’ that supports the easy integration and utilization of datasets 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 cyberinfrastructures in service of international and interdisciplinary innovation and progress.

ACKNOWLEDGMENTS

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.

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. *Graph Drawing Softwares, Mathematics and Visualization*. P. Mutzel and M. Jünger, Springer-Verlag**:** 105-126.

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, Wiley.

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. *Visualizing the Semantic Web*. V. Geroimenko and C. Chen, Springer Verlag**:** 183-198.

Börner, K., C. Chen and K. Boyack. 2003. Visualizing Knowledge Domains. *Annual Review of Information Science & Technology*. B. Cronin. Medford, NJ, Information Today, Inc./American Society for Information Science and Technology. **37:** 179-255.

Börner, K., S. Sanyal and A. Vespignani. in press. Network Science. *Annual Review of Information Science & Technology*. B. Cronin. Medford, NJ, Information Today, Inc./American Society for Information Science and Technology. **41**.

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.

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. 2006. 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.

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.

Shiffrin, R. M. and K. Börner, Eds. 2004. Mapping Knowledge Domains, PNAS.

Takatsuka, M. and M. Gahegan. 2002. GeoVISTA Studio: A Codeless Visual Programming Environment For Geoscientific Data Analysis and Visualization. *The Journal of Computers & 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. *Proceedings Graph Drawing*. P. Healy and N. S. Nikolov. Limerick, Ireland**:** 468-479.

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**: 328-37.