Plug-and-Play Macroscopes

Katy Börner
Cyberinfrastructure for Network Science Center
School of Library and Information Science, Indiana University
Bloomington, IN 47405
812-855-5120
Email: katy@indiana.edu

1. MACROSCOPE DESIGN AND UTILITY

Decision making in science, industry, and politics, as well as in daily life requires us to make sense of datasets that represent the structure and dynamics of complex natural and man-made systems. The analysis, navigation, and management of these large-scale, complex, and dynamically changing datasets require a new kind of tool, a macroscope (from the Gk. *makros* "great," and *skopein* "to observe") as proposed by Joël de Rosnay [8].

Just as the microscope empowered our naked eyes to see cells, microbes, and viruses, thereby advancing the progress of biology and medicine; and just as the telescope opened our minds to the immensity of the cosmos and prepared mankind for the conquest of space; macroscopes promise us the power to view yet another dimension—the infinitely complex. Macroscopes give us a "vision of the whole" and help us "to synthesize" the elements therein. They enable us to detect patterns, trends, and outliers while granting us access to details [17, 20]. Instead of making things larger or smaller, macroscopes let us observe what is at once too great, too slow, or too complex for our eyes.

Many of the best micro-, tele-, and macroscopes are designed by scientists keen to observe and comprehend what no one has seen before. Galileo Galilei understood the potential value of spyglasses for the study of the heavens, started to grind and polish his own lenses, and used the improved optical instrument to make discoveries like the moons of Jupiter, thus providing quantitative evidence for the Copernican theory. Today, scientists repurpose, extend, and invent new hardware and software to create macroscopes that may solve both local and global challenges [19].

This paper aims to inspire computer scientists to implement software frameworks that empower domain scientists to design their own custom macroscopes. The software would be as easy to use as Flickr, YouTube, or Lego blocks, but instead of sharing images and videos or building Lego houses, scientists would use it to share scientific datasets and algorithms and to assemble them into workflows. Scientists would customize their continuously evolving macroscopes by adding and upgrading existing and removing obsolete plug-ins to arrive at a set that is truly relevant for their work—all with little or no help from computer scientists. Some macroscopes will resemble cyberinfrastructures [1] providing easy access to massive amounts of data, services, computing resources, and expert communities. Others will come as Web services or stand-alone tools. While typical microscopes and telescopes are static physical instruments, macroscopes provide access to continuously changing bundles of software.

This paper opens with a discussion of major changes in the landscape of science (see sidebar 1) that pose challenges and opportunities for the design of effective macroscopes. We then present a set of desirable features for designing plug-and-play macroscopes and review related work. Next, we exemplarily present the design of a software architecture that extends the Open Services Gateway Initiative Framework (OSGi) and uses the Cyberinfrastructure Shell (CIShell) to support the easy integration of new and existing algorithms and their combination into simple yet powerful macroscopes. The OSGi/CIShell software framework is currently used in tools that support five different research communities. Two tools—the Network Workbench (NWB) Tool and the Science of Science (Sci2) Tool—are presented here in detail. The paper concludes with a discussion of related efforts and an outlook into the not-too-distant future.

Sidebar 1: The Changing Scientific Landscape

With many recent developments in the structure and dynamics of science comes the increased need for effective macroscopes that are easy to extend, share, and customize. For example:

Star Scientist -> Research Teams: In former times, science was driven by key scientists. Today, science is driven by effectively collaborating co-author teams, often comprising experts from multiple disciplines as well as geospatial locations [4, 18].

Users -> Contributors: Web 2.0 technologies empower any user to contribute to Wikipedia and to exchange images, videos, or code via Fickr, YouTube, and SourceForge.net respectively. Wikispecies, WikiProfessionals, and WikiProteins combine wiki and semantic technology in support of real-time community annotation of scientific datasets [14].

Cross-disciplinary: The best tools frequently borrow and synergistically combine methods and techniques from different disciplines of science and empower interdisciplinary and/or international teams of researchers, practitioners, or educators to fine-tune and interpret results collectively.

One Specimen -> Data Streams: Microscopes and telescopes were originally used to study one specimen at a time. Today, many researchers must make sense of massive streams of multiple types of data having different formats, dynamics, and origins.

Static Instrument -> Evolving Cyberinfrastructure (CI): The importance of hardware instruments that are static and expensive tends to decrease relative to software infrastructures that are highly flexible and continuously evolving to meet the needs of different sciences. Some of the most successful services and tools are decentralized, thereby increasing scalability and fault tolerance.

In addition, there are software development practices that make it possible for "a million minds" to design flexible and scalable software, available to many to improve and freely use. These practices include:
Modularity: The design of software modules with well-defined functionality that can be combined allows for contribution from multiple users, reduced costs, and increased flexibility in tool development, augmentation, and customization.
Standardization: The adoption of standards can accelerate development as existing code is leveraged. It helps to pool resources, support interoperability, and ease the migration from research code to production code, and hence the transfer of research results into industry applications and products.
Open data and open code: This practice allows any user to check, improve, or repurpose code and eases the replication of scientific studies.

2. PLUG AND PLAY SOFTWARE ARCHITECTURES

When discussing the proposed plug-and-play software architectures, it is beneficial to make a distinction between the "core architecture" that can be used to establish the mechanism or operating system for the plug-and-play datasets and algorithms, the dynamic "filling" of this core containing the actual datasets and algorithms, and the bundling of all components into "custom tools." Desirable features for plug-and-play macroscopes and major decisions to make are given in sidebar 2.

Sidebar 2: Desirable Features and Decisions to Make

When designing plug-and-play software architecture, a number of major decisions must be made based on domain requirements in order to arrive at powerful yet easy-to-use tools and services.

Division of Labor: The "core architecture" has to be implemented by computer scientists in close collaboration with domain experts. Dataset and algorithm plug-ins, i.e. the "filling," are provided by those most invested in the data and most knowledgeable about the utility of different algorithms—the domain experts. Domain experts also have the expertise needed to bundle different plug-ins into "custom tools" that support certain types of research. Computer scientists, on the other hand, are expert in implementing algorithms or tools in generic and efficient ways that can be best used by others. Hence, the ideal implementations of algorithms and tools will generally result from collaborations between computer scientists and domain experts. Technical manuals on how to extend the "core" can thus be compiled by computer scientists, while descriptions and tool tutorials on algorithms and datasets are provided by domain experts.
Ease of Use: As most plug-in contributions and usage will come from non-computer scientists, it must be possible to contribute, share, and use new plug-ins without having to write any new code. Wizard-driven integration of algorithms and data sets by domain experts, sharing plug-ins via email or online sites, deploying plug-ins by simply adding them to the "plug-in directory," and running them via a menu-driven user interface (as used in Word processing systems or Web browsers) seems to work well.
Core vs. Plug-ins: As will be shown, the "core architecture" and the "plug-in filling" can be implemented as sets of plug-in bundles. Answers to questions such as: "Should the graphical user interface, interface menu, scheduler, or data manager be part of the core or its filling?" will depend on the types of tools and services to be delivered.
Plug-in Content and Interfaces: Should a plug-in represent one algorithm or an entire tool? What about data converters needed to make the output of one algorithm compatible with the input of the next? Should those be part of the algorithm plug-in, or should they be packaged separately? Which general interfaces are needed to communicate parameter settings, input and output data? Answers to these questions will be domain-specific as they depend on existing tools and practices and the problems that domain experts attempt to solve.
Supported (Central) Data Models: Some tools use a central data model to which all algorithms conform, e.g., Cytoscape (see section 3). Other tools support many internal data models and provide an extensive set of data converters, e.g., the Network Workbench Tool (see section 2.4). The former often speeds up execution and visual rendering while the latter eases the integration of new algorithms. In addition, most tools support an extensive set of input and output formats, since a tool that cannot read or write a desired data format tends to be useless for domain experts.
Supported Platforms: If the software is to be used via Web interfaces, then Web services need to be implemented. If a majority of domain experts prefers a stand-alone tool running on a specific operating system, then a different deployment is necessary.

2.1 Related Work

There are diverse efforts that aim to empower multiple users to share code. Google Code and SourceForge.net provide unique means to share software. In summer 2009, SourceForge.net hosted more than 230,000 software projects by two million registered users; ProgrammableWeb.com hosted 1,366 application programming interfaces (API) and 4,092 mashups that combine data or functionality from two or more sources to create a new service. Web services convert the Web browser into a universal canvas for information and service delivery. Few of the many software projects, APIs, and mashups are useful for the design of macroscopes.
In addition, there are infrastructures that were specifically designed for researchers interested in composing and running analysis and/or visualization pipelines or

workflows. Among them are cyberinfrastructures (CIs) such as the cancer Biomedical Informatics Grid (caBIG) (https://cabig.nci.nih.gov), the Biomedical Informatics Research Network (BIRN) (http://www.nbirn.net), and the Informatics for Integrating Biology and the Bedside (i2b2) (https://www.i2b2.org), all of which serve large biomedical communities. The HUBzero (http://hubzero.org) platform for scientific collaboration uses the Rapture toolkit to serve Java applets. It utilizes the TeraGrid, the Open Science Grid, and other national grid computing resources for extra cycles. The collaborative environment of myExperiment (see section 4) supports the sharing of scientific workflows and other research objects. However, there exists no common standard on the "workflow," "pipeline," or "composition" of different algorithm plug-ins (also called modules or components) and datasets into scientifically relevant workflows. In one project alone, more than five incompatible "module composition" approaches were developed, each with its own unique implementation, e.g., of image registration. Researchers interested in adding a new algorithm plug-in to any of these cyberinfrastructures or in bundling and deploying a subset of plug-ins as a new tool/service generally must hire a computer scientist to achieve this. Consequently, common plug-ins often get duplicated, and many innovative new algorithms are never integrated into common CIs and tools due to resource limitations.

Websites such as Many Eyes by IBM (http://manyeyes.alphaworks.ibm.com/manyeyes/visualizations) and Swivel (http://www.swivel.com) resemble plug-and-play data architectures and show the power of community data sharing and visualization. In May 2009, Many Eyes had more than 66,429 datasets and 35,842 visualizations, while Swivel offered 14,622 data sets and 1,949,355 graphs contributed and designed by 12,144 users. Both sites let users share data (but not algorithms), generate and save different visualization types, and provide community support.

In addition, there are commercial tools such as Tableau (http://www.tableausoftware.com) and Spotfire (http://spotfire.tibco.com), 20 free tools and APIs reviewed in [6] and many other tools that support data analysis and visualization. While all of these offer highly valuable functionality, none of them makes it easy for domain experts to share their algorithms and to bundle them into custom macroscopes.

2.2 Core Architecture

Which core architecture is best to empower domain experts to plug, play, and share their algorithms and to design custom macroscopes and other tools? Here we describe one possible solution that is based on OSGi/CIShell.

The ***Open Services Gateway Initiative (OSGi)*** service platform developed by the OSGi Alliance (http://www.osgi.org) has been used in the industry since 1999. It is a dynamic module system for Java, supporting interoperability of applications and services in a mature and comprehensive way with a very effective (and small) API. The OSGi platform eases the plug-and-play integration of independent components by managing class path and dependency issues when combining components from different sources. As service-oriented architecture (SOA), it provides an easy way to bundle and pipeline algorithms into "algorithm clouds." A detailed description of the OSGi specification and existing reference implementations is beyond the scope of this paper but can be found at http://www.osgi.org/Specifications.

Leveraging the OSGi platform provides access to a large amount of industry-standard code—prebuilt, pretested, and continuously updated modules—and know-how that would otherwise take years to reinvent/re-implement, thereby enabling a reduction in time to market, development costs, and maintenance costs.

OSGi bundles can be developed and run using different frameworks—such as the Equinox project from Eclipse (http://www.eclipse.org/equinox), the reference implementation of the OSGi R4 core framework specification.

The ***Cyberinfrastructure Shell (CIShell)*** (http://cishell.org) is an open source software specification for the integration and utilization of datasets, algorithms, and tools [11]. It extends OSGi by providing "sockets" into which existing and new datasets, algorithms, and tools can be plugged using a wizard-driven process. CIShell serves as a central controller to manage datasets and seamlessly exchange the data and parameters among various implementations of algorithms. It defines a set of *generic data model APIs* and *persistence APIs*. By extending the data model APIs, various data model plug-ins can be implemented and integrated. Each data model requires a persister plug-in to load, view, and save a dataset from/to a data file in a certain format. Some data models have no "persister" plug-in, but instead convert data to or from some other format that does have one. CIShell also defines a set of *algorithm APIs* that allows developers to easily develop and integrate diverse new or existing algorithms as plug-ins. While CIShell itself is written in Java, it supports the integration of algorithms written in other programming languages, e.g., C, C++ or Fortran. In practice, a pre-compiled algorithm needs to be wrapped as a plug-in that implements basic interfaces defined in the CIShell Core APIs. Pre-compiled algorithms can be integrated with CIShell by providing metadata to indicate the types of their input and output. Different templates are available to facilitate the integration of algorithms into CIShell. A plug-in developer simply needs to fill out a sequence of forms for creating a plug-in and export the plug-in to the installation directory; the new algorithm then appears in the CIShell graphical user interface (GUI) menu. In this way, any algorithm or tool that can be run from a command line can be made into a CIShell compatible plug-in.

CIShell's reference implementation also has basic services such as a work log tracking module that records user actions, displays them in the console, and saves them in a log file, while also saving error logs in a separate file. Plus, there is a default menu-driven interface and an algorithm scheduler that shows all currently scheduled or running processes together with their progress. It can be deployed as a stand-alone tool or made available as either a Web or peer-to-peer service.

The CIShell Algorithm Developer's Guide [7] details how to develop and integrate Java and non-Java algorithms or third-party libraries into OSGi/CIShell-based systems.

OSGi/CIShell Combined: Software designed using OSGi/CIShell mainly consists of a set of Java Archive (JAR) bundles, also called plug-ins. There are OSGi services, CIShell services, and dataset/algorithm services all running in the OSGi container. The CIShell framework API is itself an OSGi bundle that does not register any OSGi services. Instead, it provides interfaces for dataset and algorithm services, basic services (e.g., logging and conversion), and application services (scheduler and data manager). Each bundle has a manifest file with a dependency list that states which packages and other bundles it needs to run. All bundles are prioritized. Upon application start-up, bundles with the highest priority are started first, followed by bundles of second, third, fourth, etc. priority. Bundles can also be started at run time if needed.
A bundle can create an object and register it with the OSGi service registry under one or more interfaces. The services layer connects bundles in a dynamic way by offering a publish-find-bind model for Java objects. Each service registration has a set of standard and custom properties. An expressive filter language is available to select relevant services. Services are dynamic, i.e., bundles can be installed and uninstalled on the fly while other bundles adapt. The service registry accepts any object as a service. However, registering objects under (standard) interfaces—e.g., OSGi interfaces or the CIShell dataset and algorithm interfaces—helps to ensure reuse. Because of the declarative specification of bundle metadata, a distributed version of CIShell could be built without changing most of the algorithms.

2.3 Algorithm Filling

In January 2010, the OSGi/CIShell plug-in pool has more than 200 plug-ins. These include approximately 60 "core" OSGi/CIShell plug-ins and a "filling" of more than 140 algorithm plug-ins plus 40 sample datasets. In addition, there are configuration files, log files, and sample data files. Nearly 60 percent of the algorithm plug-ins are implemented in Java, 20 percent in Fortran, and the remaining ones in C, C++, Jython, and OCaml.

2.4 Custom Tools

The OSGi/CIShell framework is at the core of five plug-and-play tools that resemble simple macroscopes and serve different scientific communities: the IVC was developed for research end education in information visualization; the Network Workbench (NWB) Tool was designed for large-scale network analysis, modeling, and visualization; the Sci^2 tool is mostly used by science of science (policy) researchers; the EpiC Tool is under development for use by epidemiologists; and TEXTrend supports the analysis of text. We present two of these tools in detail: the NWB tool and the Sci^2 tool.

The ***Network Workbench (NWB)*** project (http://nwb.slis.indiana.edu) developed the NWB tool and associated community website for the biomedical, physics, and social sciences. The tool uses 39 OSGi plug-ins and 18 CIShell plug-ins for the core architecture. Two of these plug-ins define the functionality of the simple GUI shown in Fig. 1; the menu (top) lets users load data and run algorithms and tools. There is a console that logs all data and algorithm operations and lists acknowledgment information on authors, programmers, and documentation URLs for each algorithm. The data manager (right) displays all currently loaded and available datasets. A scheduler lets users keep track of the progress of running algorithms. It is worth noting that this interface can be easily branded or even replaced, e.g., by a terminal interface.
There are 21 converter plug-ins that help load data into in-memory objects or into formats the different algorithms can read behind the scenes. Most interesting for users are the algorithm plug-ins that can be divided into algorithms for preprocessing (17), analysis (39), modeling (8), and visualization (8) purposes.
Last but not least, there are two stand-alone tools, GUESS and GnuPlot, available via the menu system. GUESS is an exploratory data analysis and visualization tool for graphs and networks (http://graphexploration.cond.org) (see Fig. 1). The system contains a domain-specific embedded language called Gython (an extension of Python, or more specifically Jython) which supports the customization of graph designs. GnuPlot is a portable, command-line-driven, and interactive plotting utility for data and related functions (http://www.gnuplot.info). NWB uses 15 supporting libraries such as Colt, JUNG, Jython, and Prefuse. Detailed listings are provided in the NWB tutorial [2 http://nwb.slis.indiana.edu/Docs/NWBTool-Manual.pdf].

[Place Figure 1 here]

Figure 1: The Network Workbench Tool interface and two visualizations of the same network dataset using an algorithm plug-in and the GUESS tool plug-in.

A common network science workflow includes data loading and/or modeling, preprocessing, analysis, visualization, and export of results. Different sample workflows used by different sciences are sketched below. Details and references to peer-reviewed papers are given in [2].

- ***Data conversion (across sciences)*** uses different converter algorithms to translate between more than 20 data formats.
- ***Error and attack tolerance analysis (physics/computer science)*** requires loading or modeling a network and deleting random nodes (error) or highly connected hub nodes (attack).
- ***Peer-to-peer network analysis (computer science)*** might comprise the simulation of different networks and the analysis of their properties.
- ***Temporal text analysis (information computer science, computer science)*** might apply the burst detection algorithm to identify sudden increases in the

usage frequency of words. Text has to be prepared for this analysis. Results are often visualized.

- ***Social network analysis (information science, social science)*** might compare properties of scholarly and friendship networks for the same set of people. The former network might be derived from publications, the latter data acquired via questionnaires and loaded.
- ***Discrete network dynamics (biology)*** can be studied using the Discrete Network Dynamics (DND) tool. The tool bundles the loading/modeling of a multistate discrete network model, the generation of the state space graph of the model, analysis of the attractors of the state space, and the generation of a visualization of one of the attractor basins.

Most workflows require the serial application of algorithms developed in very different areas of science and contributed by different users. Much of this complexity is hidden. For example, users do not see how many converters are involved in the execution of one workflow. Only those algorithms that can be applied to a currently selected dataset can be selected and run (all others are grayed out). Expert workflow templates and tutorials provide guidance through the vast space of possible algorithm combinations.

The ***Towards a Macroscope for Science Policy Decision Making*** project (http://sci.slis.indiana.edu) develops tools for the study of science (scientometrics) that also help answer science policy questions. The project focuses on the mapping of science at the micro (individual), meso (institution, state), and global (all of science, international) levels using temporal, geospatial, topical, and network analyses and visualization techniques [3, 16].

The Science of Science (Sci^2) Tool provides easy access to relevant algorithms. Most of the algorithms were developed outside of scientometrics. For example, temporal analysis algorithms come from statistics and computer science; geospatial analysis algorithms from geography or cartography; semantic analysis algorithms from cognitive science, linguistics, and machine learning; network analysis from social science, physics, economics, Internet studies, and epidemiology. The different areas of research have highly contrasting preferences for data formats, programming languages, and software licenses, yet the Sci^2 tool presents them all via a unified interface. The importance and utility of algorithms and data formats depends on the specific scientometric analysis. New algorithms can be easily integrated and used. In order to use a new data format, exactly one converter has to be implemented that converts the new format into one of the existing formats. Many workflows involve more data converters than algorithms, as multiple converters might be applied to bridge formats used by different algorithms. Frequently, workflows have to be rerun several times, as input data might be imperfect and initial parameter settings may need to be optimized or different algorithms compared. Thanks to Sci^2 tool, analyses that once required weeks or months to set up and run can now be designed and replicated within a few hours. Plus, the automatically generated work logs can be shared, rerun, and improved by any user.

Workflows validated by scientometricians and published in peer-reviewed journals can easily be used by science policy makers. Frequently, visualization—or visual mapping—is employed to communicate analysis results to a diverse audience (see Fig. 2). The top left network of co-authors was extracted from Medline publications downloaded from the *Scholarly Database* [6]. The network consists or authors (represented as nodes) are connected by co-authors (edges). Nodes are size- and color-coded by their betweenness centrality. Top nodes are assumed to have gatekeeper roles and are labeled by the author name. The top right shows the *UCSD Base Map of Science* with overlays of references extracted from a set of publications. While the legend shows the number of journal papers cited in different scientific disciplines, the map shows which areas of research those publications are citing or drawing from as an indication of “input interdisciplinarity.” The output of an individual, institution, or country can be mapped analogously to give an indication of core competencies that might be highly focused in one area of science or highly interdisciplinary. Horizontal bar graphs are useful when aiming to understand the portfolio of a funding agency or the funding intake of an individual or institution. Funded projects might be downloaded from the NSF Award Search site (http://www.nsf.gov/awardsearch). Each project is represented by a bar that starts and ends at a certain time (time is shown to run from left to right). Width represents the duration while area size corresponds to the total awarded dollar amount. Equipment grants are represented by narrow bars of significant height, meaning they are highly funded over a short period of time. The project title is given to the left of the bar. Bars can be color-coded by award type, e.g., SBIR, Career, etc. The visualization is provided as a postscript file, and thousands of projects can be examined at once.

[Place Figure 2 here]

Figure 2: Results of three Sci^2 tool workflows.

OSGi/CIShell-compliant plug-ins can be shared among tools and projects. For example, network analysis algorithms implemented for the NWB tool can be shared (as JAR files) via email or other means, saved in the plug-in directory of another tool, and then made available for execution in the menu system of that tool. Text-mining algorithms originally developed in TEXTrend might be plugged in the Sci^2 tool to support the semantic analysis of scholarly texts. Though funding for the NWB tool formally ended in 2009, its functionality continues to steadily increase as plug-ins developed for other tools become available. Even if no project were to fund the OSGi/CIShell core for some time, it remains functional and in-demand due to being compact, lightweight, and easy to maintain. Beyond these advantages, the true value lies in the continuously evolving algorithm “filling” and the “custom tools” being designed by domain scientists.

3. DISCUSSION

Instead of making a pilgrimage to the Library of Alexandria, the Large Hadron Collider, or any of the world’s largest optical telescopes, many researchers will soon be embracing Web 2.0 technology to share not only

images and videos but also datasets, algorithms, and tools. They will learn to operate in a world where massive amounts of new data (streams), tools, services, results, and expertise become available at every moment of every day. Computer scientists have a major role to play in making this a productive and rewarding experience. Their decisions will affect the ease with which biologists, physicists, social scientists, and others reuse, combine, and extend existing algorithms and tools across disciplinary and geospatial boundaries in support of scientific discovery, product development, and education. This article serves as a "call to action" for computer scientists to assist domain experts in sharing their algorithms and datasets. Ideally they can give domain scientists a "fishing rod instead of a fish." They will have succeeded in the design of "core architecture" if they are not needed anymore for the "filling" or the bundling of components into "custom tools."
Just as the value of the first telephones increased in proportion to the number of people using them, plug-and-play macroscopes gain value relative to the increase observed in their core functionality and the number of dataset and algorithm plug-ins. Our team plans to extend the OSGi/CIShell core to support a greater variety of interfaces and Web services. In addition, there need to be more effective means by which to share datasets and algorithms via scholarly marketplaces.

Modularity at User Interface Level
The current OSGi/CIShell core supports modularity at the algorithm level but not at the visualization level. Analogous to the decomposition of workflows into algorithm plug-ins, it is possible to modularize visualization and interaction design. Future work will focus on the development of "visualization layers" that support the easy selection and combination of different reference systems, projections/distortions, graphic designs, clustering/grouping, and interactivity.

Streaming Data
Vast amounts of data are being generated today, which need to be understood in real time. Examples include patient surveillance data streams, which are data generated by epidemic models that predict the levels of susceptible, infected, and recovered individuals in a population over time. The EpiC Tool development is contributing algorithms that read and/or output streams of data tuples, enabling algorithms to emit their results as they run (rather than only upon completion). Data graph visualizations plot these tuple streams in real time, resizing (shrinking) the temporal axis as time passes.

Web Services Deployment
The OSGi/CIShell-based tools discussed here are stand-alone desktop applications that support off-line work, on potentially sensitive data, using a GUI that is familiar to target users. However, there exist application domains that benefit from the online deployment of macroscopes. While the OSGi specification provides basic support for Web services, CIShell will be extended to address the specific needs of macroscope designers.

Marketplaces and Incentive Design
Many domain experts are seriously challenged when trying to utilize an evolving set of thousands of possibly relevant datasets compiled for different studies, in different quality and coverage, saved in different formats, and tagged using terminology specific to the original research domains. In addition, there are thousands of algorithms with different functionality, supporting diverse input and output formats, written in different languages, packaged as algorithm or tool plug-ins, having different licenses, and developed in diverse projects by domain experts from any domain of science. Domain experts clearly need effective means by which to find those datasets and algorithms that are relevant for their work, in order to bundle them into efficient workflows and relate the results to existing works. Scholarly marketplaces that might resemble Flickr or YouTube can be employed to help ease the sharing, navigation, and utilization of scholarly datasets and algorithms. They would reinforce existing reputation mechanisms, e.g., by providing simple ways to cite and acknowledge users who share, highlighting most downloaded and highly rated contributions, and offering other means by which to make datasets and algorithms, workflows, and tutorials part of a valued scholarly record.

4. OUTLOOK

Today, CIShell/OSGi is at the core of several different tools and cyberinfrastructures (see discussion section 2.4). Most interestingly, a number of other projects have recently adopted OSGi (and in one case, CIShell). Those projects include:

Cytoscape (http://www.cytoscape.org)—led by Trey Ideker at University of California, San Diego—is an open source bioinformatics software platform for visualizing molecular interaction networks and integrating these interactions with gene expression profiles and other state data [15]. Inspired by a workshop on Software Infrastructures in July 2007 [5], Mike Smooth and Bruce W. Herr implemented a basic proof-of-concept OSGi-based Cytoscape core in October 2007. First OSGi bundles are available at http://chianti.ucsd.edu/svn/core3. Once the new Cytoscvope core is implemented, it will become much easier to share plug-ins between the NWB tool and Cytoscape, thereby extending the functionality and utility of both.

Taverna Workbench (http://taverna.sourceforge.net)—developed by the myGrid team (http://www.mygrid.org.uk) and led by Carol Goble at University of Manchester, United Kingdom—is a suite of free open source software tools for designing and executing workflows [12]. Taverna allows users to integrate many different software tools, including more than 30,000 Web services from diverse domains such as chemistry, music, and social sciences. The workflows are designed in Taverna Workbench and can then be run on a Taverna Engine, either within the Workbench, on an external server, within a portal, or on a computational grid. Taverna has an extensible and flexible architecture with approximately 20 plug-ins..Currently, Taverna uses a Taverna-specific classloader and registry mechanism (Raven), but an implementation based on Spring Dynamic

Modules (which uses OSGi) is under development. The myExperiment (http://www.myexperiment.org) social website supports finding and sharing of workflows and has special support for Taverna workflows [9].

MAEviz (https://wiki.ncsa.uiuc.edu/display/MAE/Home) —managed by Shawn Hampton at the National Center for Supercomputing Applications (NCSA)—is an open-source, extensible software platform which supports seismic risk assessment based on the Mid-America Earthquake (MAE) Center research in the Consequence-Based Risk Management (CRM) framework [10]. It uses the Eclipse Rich Client Platform (RCP) that includes Equinox, a component framework based on the OSGi standard (https://wiki.ncsa.uiuc.edu/display/MAE/OSGI+Plug-ins).

TEXTrend (http://www.textrend.org)—led by George Kampis at Eötvös University, Hungary—develops a framework for the easy and flexible integration, configuration, and extension of plug-in-based components in support of natural language processing (NLP), classification/mining, and graph algorithms for the analysis of business and governmental text corpuses with an inherently temporal component [13]. TEXTrends recently adopted OSGi/CIShell for the core architecture. The first plug-ins include the Unstructured Information Management Architecture (UIMA) (http://incubator.apache.org/uima); the data-mining, machine-learning, classification, and visualization toolset WEKA (http://www.cs.waikato.ac.nz/ml/weka); Cytoscape; Arff2xgmml converter; R (http://www.r-project.org) via iGgraph and scripts (http://igraph.sourceforge.net); yEd (http://www.yworks.com); and the Cfinder clique percolation analysis and visualization tool (http://www.cfinder.org). TEXTrend extended CIShell's workflow support and offers the first Web services.

As the functionality of OSGi/CIShell-based software frameworks improves, and as the number and diversity of dataset and algorithm plug-ins increases, so too will the capabilities of custom tools or macroscopes continue to proliferate and expand.

5. ACKNOWLEDGEMENTS

I would like to thank Micah Linnemeier and Russell J. Duhon for their stimulating discussions and extensive comments on this paper. Bruce W. Herr II, George Kampis, Gregory J. E. Rawlins, Geoffrey Fox, Shawn Hampton, Carol Goble, Mike Smoot, Yanbo Han, and a number of anonymous reviewers provided valuable input and comments to an earlier draft of this paper. I owe thanks to the members of the Cyberinfrastructure for Network Science Center and the Network Workbench and Science of Science project team for their contributions toward the work presented here. There was also a tremendous benefit to software development due to the efforts of the open source community. Full software credits are distributed with the source, but we would especially like to acknowledge Jython, JUNG, Prefuse, GUESS, GnuPlot, and OSGi, and Apache Derby used in the Sci^2 tool. This material is based upon work supported by the National Science Foundation under grants SBE-0738111, IIS-0513650, and the National Institutes of Health under grant NIH RM-07-004. 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

6. REFERENCES

[1]. Atkins, D.E., Drogemeier, K.K., Feldman, S.I., Garcia-Molina, H., Klein, M.L., Messerschmitt, D.G., Messian, P., Ostriker, J.P. and Wright, M.H. Revolutionizing Science and Engineering through Cyberinfrastructure. in *Report of the National Science Foundation Blue-Ribbon Advisory Panel on Cyberinfrastructure*, National Science Foundation, Arlington, VA, 2003.

[2] Börner, K., Barbási, A.-L., Schnell, S., Vespignani, A., Wasserman, S., Wernert, E.A., Balcan, D., Beiró, M., Biberstine, J., Duhon, R.J., Fortunato, S., Herr II, B.W., Hidalgo, C.A., Huang, W.B., Kelley, T., Linnemeier, M.W., McCranie, A., Markines, B., Phillips, P., Ramawat, M., Sabbineni, R., Tank , C., Terkhorn, F. and Thakre, V. Network Workbench Tool: User Manual, 1.0.0. (9/16). http://nwb.slis.indiana.edu/Docs/NWBTool-Manual.pdf (accessed 1/26/2010).

[3] Börner, K., Chen, C. and Boyack, K.W. Visualizing Knowledge Domains. in Cronin, B. ed. *Annual Review of Information Science & Technology (ARIST)* Information Today, Inc./American Society for Information Science and Technology, Medford, NJ, 2003, 179-255.

[4] Börner, K., Dall'Asta, L., Ke, W. and Vespignani, A. Studying the Emerging Global Brain: Analyzing and Visualizing the Impact of Co-Authorship Teams. *Complexity: Special Issue on Understanding Complex Systems*, *10* (4). 57-67.

[5] Börner, K., Herr II, B.W. and Fekete, J.-D., IV07 Software Infrastructures Workshop. in *11th International Conference on Information Visualization*, (Zurich, Switzerland, 2007).

[6] Börner, K., Huang, W.B., Linnemeier, M., Duhon, R.J., Phillips, P., Ma, N., Zoss, A., Guo, H. and Price, M.A. Rete-Netzwerk-Red: Analyzing and Visualizing Scholarly Networks Using the Network Workbench Tool. Scientometrics: Springer Online 2010. http://www.springerlink.com/openurl.asp?genre=article&id=doi:10.1007/s11192-009-0149-0 (accessed on 1/26/2010).

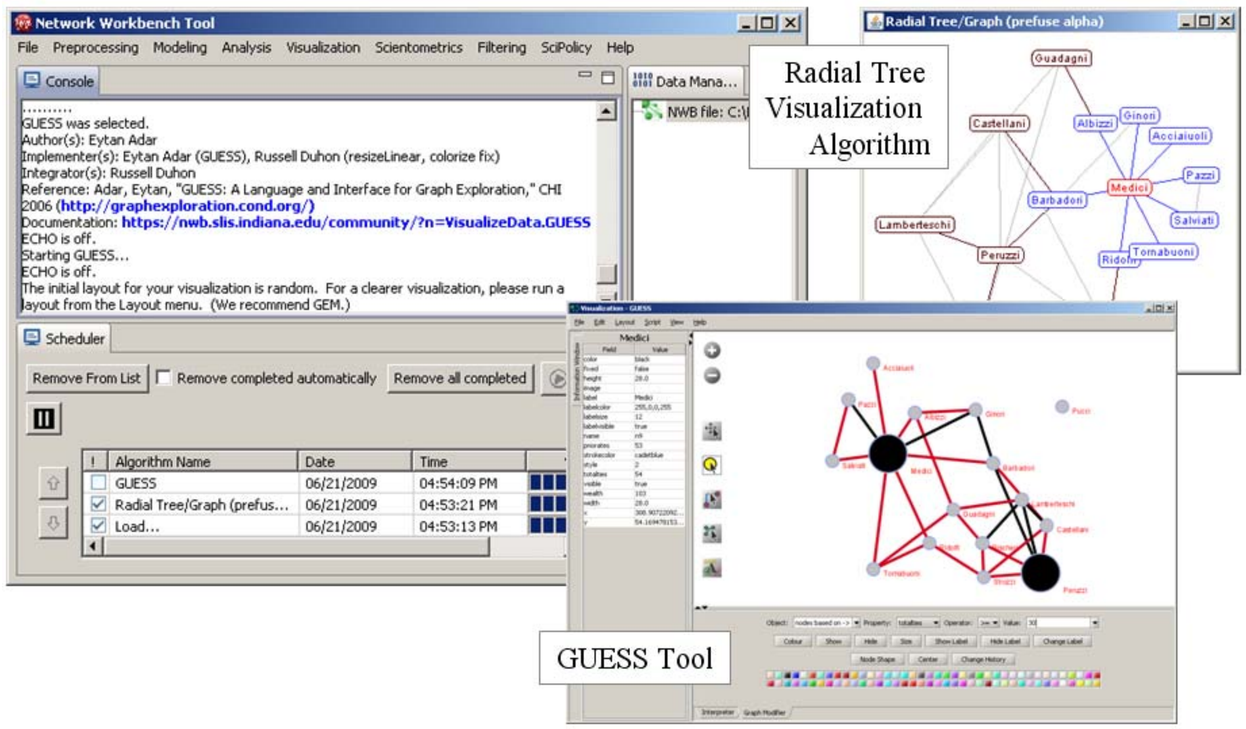
[7] Cyberinfrastructure For Network Science Center. CIShell: Cyberinfrastructure Shell. Dev Guide/Algorithm Developer's Guide. 2009. http://cishell.org/?n=DevGuide.HomePage (accessed on 10/27/2009).

[8] de Rosnay, J. *Le Macroscope: Vers une Vision Globale. Editions du Seuil.* . Harper & Row Publishers, Inc., New York, 1975.

Plug-and-Play Macroscopes

Katy Börner
Cyberinfrastructure for Network Science Center
School of Library and Information Science, Indiana University
Bloomington, IN 47405
812-855-5120
Email: katy@indiana.edu

Figures:



```
..........
GUESS was selected.
Author(s): Eytan Adar
Implementer(s): Eytan Adar (GUESS), Russell Duhon (resizeLinear, colorize fix)
Integrator(s): Russell Duhon
Reference: Adar, Eytan, "GUESS: A Language and Interface for Graph Exploration," CHI
2006 (http://graphexploration.cond.org/)
Documentation: https://nwb.slis.indiana.edu/community/?n=VisualizeData.GUESS
ECHO is off.
Starting GUESS...
ECHO is off.
The initial layout for your visualization is random.  For a clearer visualization, please run a
layout from the Layout menu.  (We recommend GEM.)
```

!	Algorithm Name	Date	Time
☐	GUESS	06/21/2009	04:54:09 PM
✓	Radial Tree/Graph (prefus...	06/21/2009	04:53:21 PM
✓	Load...	06/21/2009	04:53:13 PM

Figure 1: The Network Workbench Tool interface and two visualizations of the same network dataset using an algorithm plug-in and the GUESS tool plug-in.

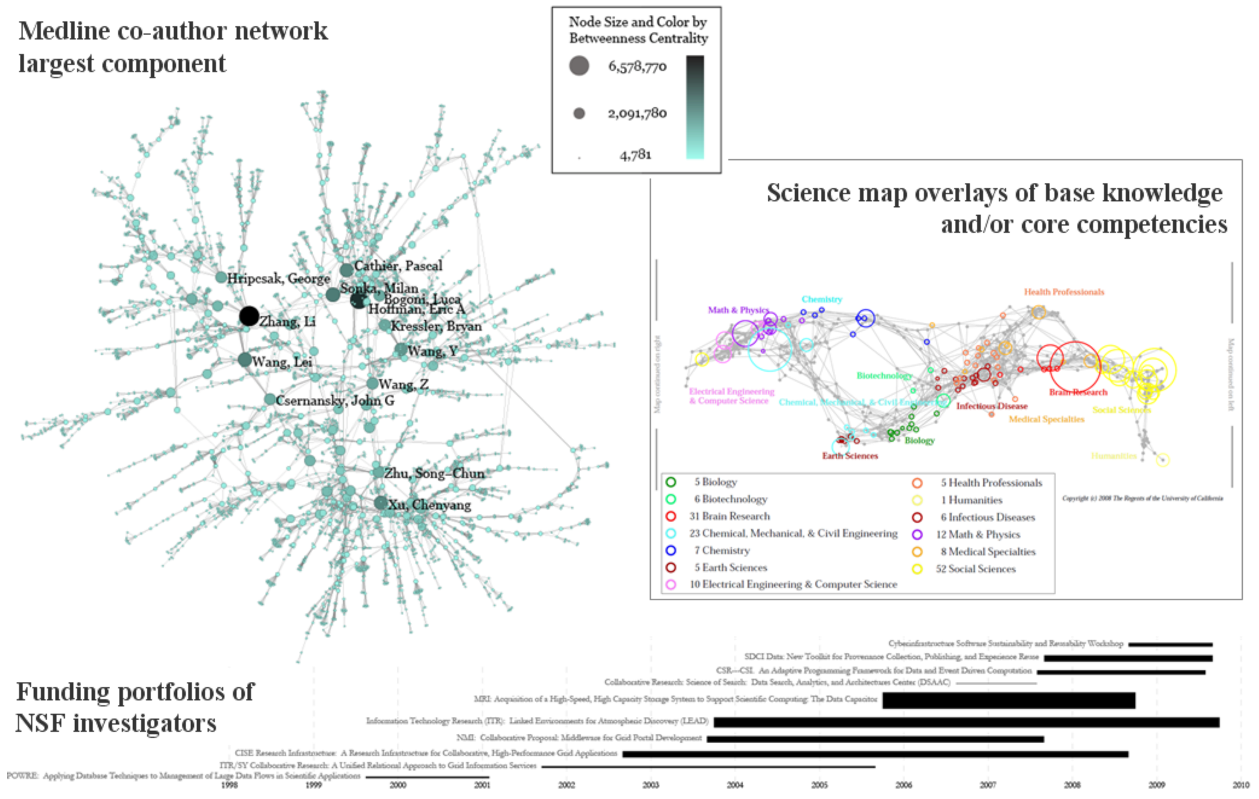


Figure 2: Results of three Sci^2 tool workflows.