Techniques for Monitoring Runtime Architectures of Socio-technical Ecosystems

Amber Lynn McConahy
Institute for Software Research
School of Computer Science
Carnegie Mellon University
amber.mcconahy@cs.cmu.edu

Ben Eisenbraun
SBGrid Consortium
Harvard Medical School
bene@hkl.hms.harvard.edu

James Howison
School of Information
University of Texas
jhowison@ischool.utexas.edu

James D. Herbsleb
Institute for Software Research
School of Computer Science
Carnegie Mellon University
jdh@cs.cmu.edu

Piotr Sliz
SBGrid Consortium
Department of Biological Chemistry
and Molecular Pharmacology
Harvard Medical School
sliz@hkl.hms.harvard.edu

ABSTRACT
This paper discusses the value of software instrumentation in architecting the runtime behavior of complex, socio- technical ecosystems. Our solution is targeted at gathering metrics from scientific software communities in order to better understand them. We hope that by gathering information, such as usage frequency, operating systems in use, and execution time, that we can provide a valuable tool that helps administrators in their decision-making process with regards to potential evolutionary tracks under consideration.

Author Keywords
Scientific software, dynamic architecture, software instrumentation

ACM Classification Keywords
H.5.3 Group and Organization Interfaces: Computer- supported cooperative work

INTRODUCTION
Socio-technical ecosystems are proliferating in every business domain. These complex systems are comprised of diverse and heterogeneous components that interact at runtime in specialized and individualized compositions. Although they can be evaluated using static architectural methods, such as traditional class diagrams, constructing a viable dynamic architecture of these systems is exceedingly difficult, and the inability to visualize the ecosystem at runtime hinders the ability of administrators to ensure the maintainability of the system. This is further compounded when trying to design an ecosystem from the bottom up. In these scenarios, the ability to determine the key, core components in a nascent community can facilitate architecting future evolutions in the ecosystem, yet without knowledge to identify the core components, decision-making is cumbersome.

We argue that software instrumentation can aid in constructing a dynamic view of an emergent ecosystem. By having explicit knowledge of which software components are used most frequently and in conjunction with one another, we believe that we can identify potential cores for evolving ecosystem platforms. Our instrumentation strategy, Socio-technical Ecosystem Analysis Metrics (STEAM), is geared towards mapping software usage in nascent scientific ecosystems with our first

CSCW'12, February 11–15, 2012, Seattle, Washington, USA.

implementation being deployed within Harvard's SBGrid.

PREVIOUS RESEARCH

Research domains focusing on socio-technical ecosystems, scientific software, and software instrumentation provide the necessary background leading into our research.

Socio-technical Ecosystems

The concept of a socio-technical ecosystem is rooted in the notion of biological ecosystems. Linda Northrop et al. [4] describe socio-technical ecosystems as a dynamic community of competing and interdependent people, organizations, and computing systems operating in a complex, capricious environment. Thus, the following properties are common within socio-technical ecosystems:

- Complex and difficult to monitor
- Decentralized control
- Unpredictable interactions and dependencies that could lead to "Big Ball of Mud" as described by Foote and Yoder [3]
- Competition for resources
- Need for appropriate incentives to ensure survivability
- Rules in place to guide effective resource use
- Evolutionary and responsive to change

Design and development of these large-scale, complex systems is extremely difficult, and the extrapolation of many of the software engineering processes used in the design of a single system is ineffective. Designing socio-technical ecosystems from the bottom-up is an open research area, and viable processes and methodologies have not yet been definitively identified.

At the core of any socio-technical ecosystem lies a platform. The term platform is often a source of confusion in the technical domain due to a variety of definitions. In this context, we are using Baldwin and Woodward's [2] description of platforms as a set of core components forming an infrastructure upon which derivative products can be launched. Similarly, Iansiti and Levien [6] describe keystone members of a network or ecosystem that form the core vital to the survival to the peripheral members. This core is analogous to a platform where the peripheral members depend on the platform for survivability. Therefore, it is essential to identify the appropriate candidates for a platform and design the ecosystem around this core platform.

Scientific Software

It is important to evaluate the features of any software system in order to better understand its complexities and how it behaves under various circumstances. However, investigating a software system that is in its nascent stages of evolution is of particular interest. Scientific software fits well into this categorization. Although largely overlooked by software engineers, scientific software provides a useful paradigm for investigative software research due to its critical importance and unique characteristics. According to Howison and Herbsleb [4], scientific software communities are plagued with challenges that hinder the proliferation of platforms. Specifically, they found that software in science falls victim to the following shortcomings:

- Complex and difficult to manage
- Incentive rules that discourage collaboration
- Reimplementation rather than reuse, which leads to unnecessary competition
- Failure to adopt platform design strategies
- Creation of a disparate system of systems analogous to the "Big Ball of Mud" [3].

Due to the existence of these problems and the nascent state of scientific software ecosystems, STEAM instrumentation strategies will target these ecosystems in order to promote platform migration and evolution into a more stable ecosystem, thus eliminating many of these problem areas.

Specifically, instrumentation could identify the packages that are used most frequently and require regular maintenance. These popular packages are also potential candidates for future plans geared towards platform migration. In

other words, by understanding what software is being used and what other software it is dependent on, we can better assess the feasibility of a package for inclusion in a platform. Additionally, though instrumentation metrics, we could identify duplicate offerings and antiquated packages that are no longer useful to the ecosystem, thus removing bloat and redundancy in the system.

Software Instrumentation Strategies

Monitoring software is frequently used to gather data and metrics about the software processes running on a system. This information is useful in providing developers with insight into statistics, such as frequency of use, version, last access time, etc. Although many tools are currently available, none entirely meet the goals of our instrumentation strategy.

iNotify is one such tool. Although it can identify changes to a file, it relies on the use of the /proc directory, which is not available on many OSX operating systems. According to Robert Love [7], iNotify is a Linux kernel feature that monitors a set of files for changes using the concept of events. It was developed to be a lightweight replacement for the antiquated dNotify tool that only could monitor directories. iNotify is able to watch directories and identify the file inside the directory that has changed in response to an event. iNotify is written in the C programming language, but a counterpart is also available called pynotify that is written in Python.

Debian Popularity Contest (PopCon) also has properties that align with our instrumentation goals. For example, PopCon gathers some of the data that we wish to analyze. However, PopCon is designed for Debian systems and relies on the dpkg package management system, which prevents us from gathering data from other operating systems not utilizing dpkg. PopCon is a tool used to map the popularity of various Debian packages as described by Bill Allombert [1]. Voluntary participants can choose to install PopCon on their Debian systems, and PopCon collects various metrics related to the installation and usage of these packages. Metrics collected by PopCon include: package name, last access time, creation time, install location, and CPU architecture. These metrics are then extrapolated to provide relevant statistics related to the frequency of use for each package.

RESEARH METHODOLOGY

Through instrumentation of the software components constituting the ecosystem, we hope to ascertain the runtime architectures that commonly occur within a given community. We feel that the data gathered from this instrumentation will enable us to extrapolate usage patterns within a community, thus facilitating the identification of key components of the community that act as keystones or potential platform candidates as the ecosystem continues to evolve. We will achieve this goal through the construction of visualizations that enable us to delineate patterns in usage data that have target properties. For example, if a particular package is utilized frequently, we know that this package should be maintained and be included in future releases.

SBGrid Consortium

The SBGrid Consortium is maintained by Harvard Medical School and is our first target ecosystem for instrumentation. The SBGrid Consortium as described in [9] is an international group of both for-profit and non-profit groups engaged in structural biology research. Started at Harvard in 1999, the consortium has grown from a handful of members to 175 research groups across more than 50 institutions and 12 countries. Members pay an annual fee to support administration of a diverse collection of scientific applications used in structural biology research. The collection is currently comprised of 248 software titles and is available for 32 and 64-bit Linux as well as OSX. Although the grid computing services offered by SBGrid are a valuable asset, we will not be focusing on the grid components of SBGrid. Rather, our focus is centered on the software resources provided to users making use of their own computing services. One of the goals of SBGrid is to enable labs to spend more time focusing on science rather than wasting resources maintaining software. Consequently, SBGrid provides

services and resources including: software maintenance, computing access, and training. SBGrid also offers the SBGrid Developer Network, which allows consortium members to interact closely with developers. The SBGrid Developer Network strives to alleviate the strain on developers by providing resources, tools, and virtual machines that can be utilized by developers as a sandbox for testing and integration. The model employed by SBGrid has been quite beneficial to subscribers and has eliminated the need for dedicated support technicians in many member labs. SBGrid hopes to further improve this model and evolve their ecosystem to ensure its continued success moving forward.

Client instrumentation

It is not always feasible or possible to tailor a "one-size fits all' instrumentation strategy that is useful for all ecosystems. Therefore, an instrumentation strategy should be tailored specifically for the target ecosystem, thus supporting an implementation that aligns with the ecosystem's technical capabilities and desired quality attributes. The SBGrid software environment is architected for use on UNIX-style operating systems and all configuration is done through the traditional UNIX shell (sh/csh). The software distribution is sizeable; it currently includes about 150 GB of compiled and configured software encompassing almost 1100 versions of the 248 applications. These factors led us to determine that simple shell script wrappers offered the most practical solution to data collection. Bash shell scripts are lightweight, flexible and well-supported on our target operating systems, each time a program is run the wrapper script will gather the following metrics and send them to the STEAM server for analysis:

- Program name and version
- MD5 obfuscated username and hostname
- Execution time
- Operating system and version
- Site/install name (identifies lab or institution)
- Exit status
- External dependencies
- Install directory

Server Design

Our server design will provide dashboard visualizations that enable administrators to view usage metrics for the target ecosystem. The server is designed to receive HTTP POST messages from clients when applications within the ecosystem are used. Upon exiting a program, the relevant data is sent to the server where it is parsed and subsequently stored in a MySQL database. The visualization dashboard consists of a web interface providing graphical representations of useful metrics that can help reconstruct the runtime architecture, such as:

- Frequency of application use
- Frequency of use over time
- Proportions of various operating systems in use
- CPU architectures utilized
- Length of time for a single application use
- Most prolific users (sites and institutions)
- Number of unique sites or institutions
- Number of unique software packages used
- Software usage by user
- Version of software used
- Dependencies for use of an application
- Packages frequently used together
- Frequently used software complements of an application

This list is not all-inclusive, and we are confident that as more data is available the need for additional visualizations will arise. As these needs arise, the web infrastructure will easily facilitate enhancements and additional visualizations.

PRELIMINARY RESULTS

SBGrid has provided us with some preliminary log data so that we can test our design for the analysis of data that is similar to the desired instrumentation data. Utilizing this data, we have constructed a prototype to test the viability of our planned implementation. Prototyping has indicated that the usage data provides relevant insight that is helpful in constructing the dynamic architecture of the target ecosystem. We are currently in the process of designing dashboard

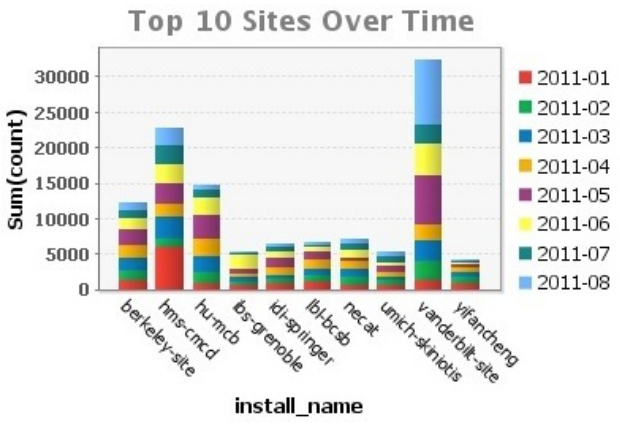

Figure 1: Top 10 institutions utilizing SBGrid and their usage patterns over time

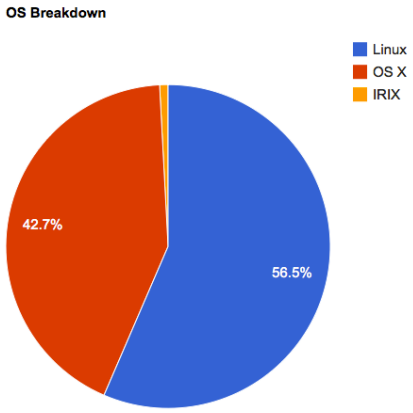

Figure 2: Distribution of operating systems of SBGrid users

visualizations that facilitate a better picture of a socio-technical ecosystem during runtime. As shown in Figure 1, historical log data from SBGrid enables us to pinpoint the ten SBGrid participant sites or institutions that utilize the system most frequently. Through this identification, SBGrid administrators are able to ensure that future upgrades due not degrade the usage frequency of these institutions. In other words, this data enables SBGrid to better plan the evolution of the ecosystem by ensuring that the needs of the most prominent customer base are upheld. Similarly, Figure 2 provides developers with relevant data that enables them to ascertain a dynamic picture of the ecosystem. In this visualization, we see that the two dominant operating systems in use in the SBGrid ecosystem are Linux and OSX. It is obvious that from this pie graph that providing support for Linux and OSX versions of structural biology tools is crucial to the health of the ecosystem. Through this data, SBGrid administrators can ensure that updates and patches for these operating systems are kept current. In short, the value of these metrics is evident through these basic visualizations, and we are confident that the other aforementioned visualizations will provide benefits as well. Using this analytical data, we hope to be able to identify patterns that will help to eliminate redundancy, identify platform candidates, assess dependencies, and provide strategies for the evolution of the community.

LIMITATIONS/FUTURE PLANS

Utilizing software instrumentation to visualize dynamic behavior of components in an ecosystem is not without limitations. Specifically, understanding the complexities of an ecosystem is not fully possible though the use of architectural data. In other words, facets of the socio-technical ecosystem, such as identification of incentives and crediting of developers, are not possible through software instrumentation alone. Yet, our solution could potentially provide a wealth of data in other forms. For example, funding agencies would be able to determine which packages should receive additional funding based on their popularity. This funding would serve as an additional incentive to developers who are likely to achieve a higher status and better reputation within the scientific community. Additionally, it is not possible to extrapolate information regarding the specific reasons behind choosing a specific piece of software over another or choosing to forego a patch or update. Rather, interviews and discussions with users are needed to ascertain this information. Hence, it is challenging to infer reasoning behind a given use, which could potentially give us a better understanding of the ecosystem. Although ascertaining a comprehensive picture of the system is challenging under these circumstances, we still believe that useful inference arise from the data gathered from our study.

Future releases of our solution help to address some of these issues and enable us to get a better

understanding of the inner-workings of such complex systems. For example, we plan to link publications with the software used in their preparation. This would provide a mechanism for crediting developers, whereby users would be able to see a comprehensive list of research that was accomplished through the use of a particular tool. We also would like to interview developers and users to ascertain reasons behind various decisions. Finally, we plan to expand STEAM to other scientific communities, so that we can get a better understanding of the dynamics of ecosystems in a domain – science -- that is largely ignored by software engineering research. We also plan to explore the utility of our approaches in other kinds of socio-technical ecosystems, such as open source communities and the internet of things.

CONCLUSION

In summary, we believe that the research into software instrumentation as a means to reconstruct runtime architectures is supported by the prototyping results. We are anxious to continue this research and hope that our future endeavors uncover a strategy for ecosystem evaluation that is capable of retrieving useful data in a variety of settings. We believe that instrumentation in conjunction with visualization can eliminate much of the ambiguity associated with socio-technical ecosystem evaluation and analysis. If correct, the value of thee findings could alleviate much of the confusion associated with such large-scale systems.

ACKNOWLEDGMENTS

This material is based in part upon work supported by the National Science Foundation under Grant No. SMA1064209 and the National Science Foundation Research Coordination Network Grant No. 0639193 (to P.S.).

REFERENCES

1. Allombert, Bill. Debian Popularity Contest. http://popcon.debian.org/
2. Baldwin, Carliss Y. and C. Jason Woodward. The architecture of platforms: an unified view. *Platforms, Markets, and Innovation.* Edited by Annabelle Gawer. Edward Elgar Publishing Limited, Cheltenham, UK (2009).
3. Foote, Brian and Joseph Yoder. Big Ball of Mud. *Forth Conference on Patterns, Languages of Programs.* (1997).
4. Herbsleb, James and James Howison. The Scientific Software Network Map. *NSF Grant #1064209* http://www.nsf.gov/awardsearch/showAward.do?AwardNumber=1064209
5. Howison, James and James D. Herbsleb. Scientific software production: incentives and collaboration. *Proceedings of the ACM 2011 conference on Computer supported cooperative work* (2011), 513–522.
6. Iansiti, Marco and Roy Levien. *The New Keystone Advantage: What the New Dynamics of Business Ecosystems Mean for Strategy, Innovation, and Sustainability*. Harvard Business School Publishing, Boston, MA, USA (2004).
7. Love, Robert. Kernel Korner: Intro to iNotify. *Linux Journal.* (2005), http://www.linuxjournal.com/article/8478
8. Northrop, Linda et al. *Ultra-Large-Scale Systems: The Software Challenge of the Future.* Software Engineering Institute, Pittsburgh, PA, USA 2006.
9. SBGrid Consortium. About Us. http://sbgrid.org/about/general