Reducing Java Internet project risks: a case study of public measurement of client component functionality in the user community

Tomas Hruz
Institute for Theoretical Computer Science
ETH Zurich
8092 Zurich, Switzerland
tomas.hruz@inf.ethz.ch

Matthias Hirsch-Hoffmann
Institute of Plant Sciences
ETH Zurich
8092 Zurich, Switzerland

Wilhelm Gruissem
Institute of Plant Sciences
ETH Zurich
8092 Zurich, Switzerland

Philip Zimmermann
Institute of Plant Sciences
ETH Zurich
8092 Zurich, Switzerland

ABSTRACT

A major risk for Internet software projects which have server and client components are decisions related to availability and features on client computers in the user community. Specifically, bioinformatics software developers intending to use Java face critical decisions about which Java version to implement, but few statistics are available about Java presence on user machines. To obtain this information, we implemented a measurement system to detect the presence, functionality and version of Java Virtual Machines on client computers of a large base of users from the biology community. We show that our system effectively collects the necessary information and provides decision-relevant statistics. Measurements performed on 1753 client computers showed that Java presence is high and dominated by the most recent Java versions. The proposed empirical approach can be used to reduce decision risks in any type of Internet software project with low level of control on client equipment and high demands on client interaction and performance. More details together with source code and measurement results can be obtained from the J-vestigator survey page (https://www.genevestigator.ethz.ch/index.php?page=jvestigator)

1. INTRODUCTION

Most Internet software projects which have a client software component face critical decisions in the early project phase when assumptions about client computers in the user community have to be taken. A distinctive feature of Internet software projects compared to other industrial software productions is that the platforms and features installed on the user computers are not centrally controllable and software project decision makers can influence the user community only in a very limited way.

The present case study illustrates a method about how to reduce the above-mentioned risk using a measurement system which provides systematic data about components installed on user computers in the community where the new software is supposed to be used. Once the structure of components installed on these computers is known, a fact-based decision concerning client software technologies can be made. Moreover, our method classifies user computers to well defined groups of potential problems that users will encounter when using the client software product. The product penetration and functionality can therefore be increased with a targeted information campaign for such user groups, providing detailed explanations and procedures how to install or upgrade necessary components on their computers.

More specifically, the problem of deciding about a client software technology in a distributed client/server system has three important parameters leading to a three dimensional decision space. The first parameter represents the complexity of interaction on the client side of the application. If a very complex interaction is projected, a more powerful programming platform like Java compared to Web browser is necessary. The second parameter is the server load. A powerful programming language on the client side is needed if users repeat with high frequency cpu intensive computing. In such cases, a lack of possibility to compute on the client side leads to intractable load on the servers. The third important parameter is the user environment controllability. This describes to what extent the application provider can control which platform features, and which versions are installed on user computers. In large companies with homogeneous configuration control, application providers can prescribe the environment and therefore possess a very high client computer environment controllability. In Internet, however, the controllability is almost null. Normally, it is advised to use a platform like Java only in situations where complex interactivity and/or high server load are combined

PPPJ 2006 August 30 - September 1, 2006, Mannheim, Germany

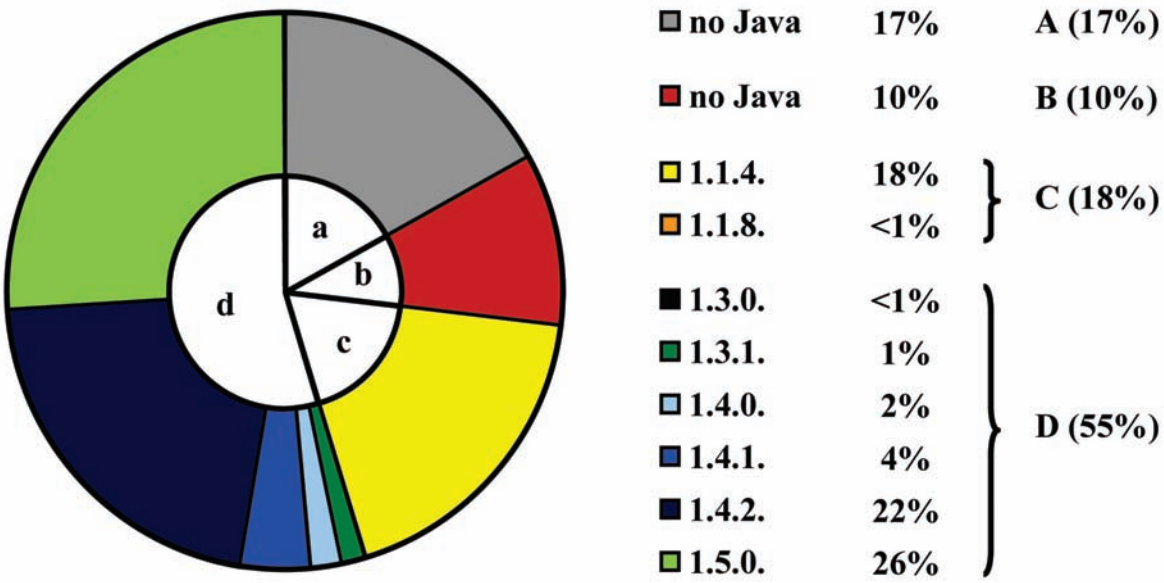

Figure 1: Results from the J-vestigator survey as of December 2005.

with high user computer environment controllability.

The presented method is directed to exactly the opposite situation, where low user environment controllability is combined with high server load and/or complex interactivity of the application. The correct measurement of software parameters on client computers can improve software development decisions, resulting in optimal choices for platforms and versions and to powerful applications increasing user satisfaction in the Internet community.

The method described in our article can be seen as an essential step on the path towards advanced and functionally rich Internet software applications. In our specific case, the correct identification of the future technology required the presence of a more simple web interface application. Generally, if we consider the development of an Internet application in a wider frame it has analogies with the "bootstrapping" process as known in the Unix development paradigm. One way how to incorporate the method proposed in this article in a technology bootstrapping is as follows: 1) An application version based on lower level technology (e.g. html, web interface) is used. This application version defines the user community and allows to measure the more advanced technology (e.g. Java). 2) The higher level technology features are identified with a measurement step. 3) The application version is replaced/enhanced with a version containing the new technology identified in step 2 and the steps starting at 1. are repeated.

2. JAVA TECHNOLOGY DECISIONS IN BIOINFORMATICS

The challenge of bioinformatics developers is to create widely distributable, highly performant and rapidly evolving tools that meet the needs of research biologists. Java [3] is emerging as a key player in bioinformatics due to its platform independence and its object-oriented programming nature, allowing to model highly complex biological information. Although traditionally the language of choice for many bioinformaticians has been Perl, more and more applications are being developed using Java technology, frequently connected to a relational database (e.g. [2, 1]).

The environment on user computers is highly heterogeneous. First, large variations occur with respect to hardware, software platform, operating system, and installed applications. Second, Java usage is strongly related to the installed browser technology, resulting in a panoply of environments in which the application must run (see the measurement results). Despite its many advantages, web-based Java applications request compatible versions of Java on client computers.

Furthermore, as Internet-based bioinformatics applications grow in complexity and in the number of users, resources required from servers may outgrow the capacities of many server infrastructures. A reasonable solution is to translocate parts of the data processing to the client computers. For this, Java technology can provide a powerful solution if it is correctly running on the client computers.

With respect to the high investment in resources for software development and the low level of control over client computers from the Internet, initial technological decisions represent a major risk factor for software developers. Knowledge about Java penetration and functionality in the biology community represents a considerable benefit in terms of investment, strategic planning, and software distribution for bioinformatics projects using Java.

While designing a new version of Genevestigator ([4]) with the goal of optimal performance and maximum compatibility, we faced the decision about whether to use JAVA, and if yes, which version would be most suitable. In this respect, two options are available: a) to program for Java 1.1.X, with which programs will run for a larger group of users.

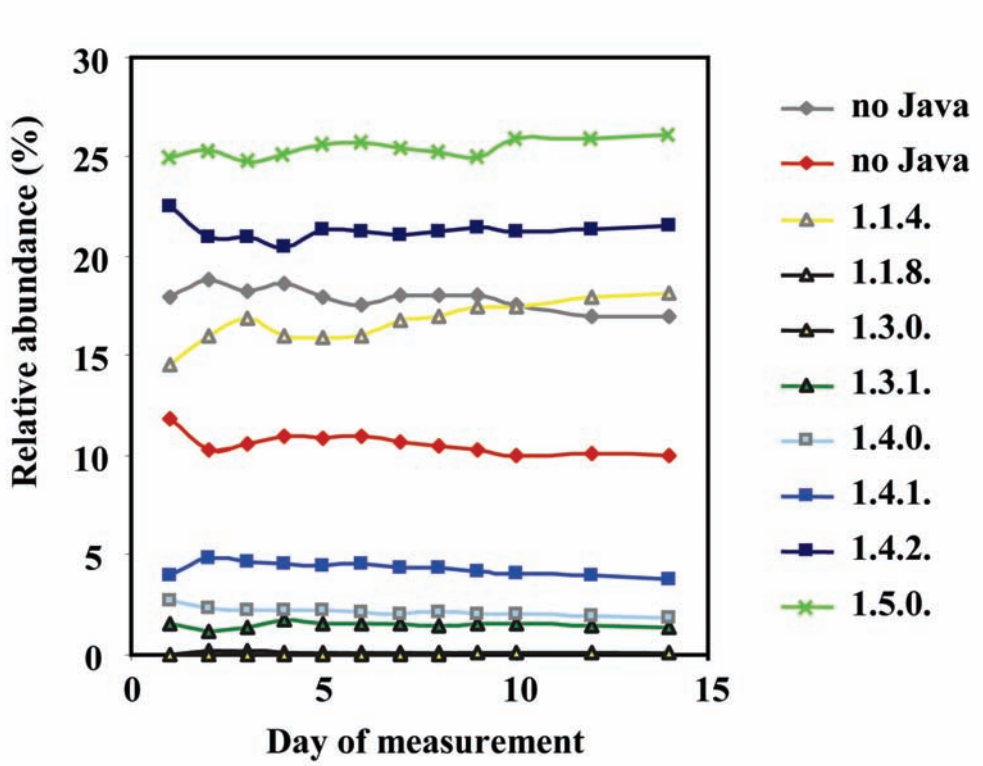

Figure 2: Evolution of the relative abundances of Java versions during the period of measurement (cumulative).

However, more difficulties occur at all levels of the product development cycle (programming, debugging, deployment etc.) because Java 1.1.X has less powerful libraries and a less efficient Virtual Machine, or b) to program for Java 1.3 or higher (referred to here as Java 2), with which the graphics and GUI elements are more powerful, the Virtual Machine is very fast, and the development tools are better integrated.

Currently, there are no public reliable statistics about Java presence, version and functionality on user computers in the biology community. To address this problem, we designed a system to measure such statistics built on Genevestigator, which has a large and world-wide user community from bioinformatics and biological research.

3. MEASUREMENT METHOD AND RESULTS

The Java measurement system consists of several layers. First, a MySQL database instance is required to store the data. Second, PHP scripts generate the appropriate Javascript code, the HTML code and a simple Java Applet to read and save the Java version of client computers. The Applet code is kept as simple as possible to be executable on all possible Java versions, and it only reports information if it runs correctly on the client machine. Therefore, to obtain information also for those cases in which Java is non-functional or absent, and to ensure that measurements can be obtained from most user computers, we developed and tested a combination of PHP, HTML, Javascript, and Java. More details about the code fragments and implementation can be obtained from the J-vestigator survey page (https://www.genevestigator.ethz.ch/index.php?page=jvestigator).

The results from the analysis of 1753 user computers reveal multiple interesting aspects. First, the complexity of the application landscape on user computers was surprising, with 371 different browser and operating system variations in the test data. This means that on average only about 5 computers share the same configuration in the pool of 1753 computers. Second, with respect to Java, the results can be grouped into four categories (see Fig. 1): A and B) no Java is available (17% and 10%, respectively), C) a functional, lower version of Java (1.1.X) is available (18%), and D) a functional, higher version of Java (Java 2) is available (55%)

Altogether, Java was present and functional on 73% of client computers (see Fig. 1, groups C and D). Considering different Java versions, Java 2 had a larger penetration than Java 1.1.X versions (55% versus 18%). In most of the latter cases, this reflects a per default implementation of Java 1.1.X in the Microsoft IE browser. As compared to Java 2, programming for Java 1.1.X would therefore add only 18% to the set of user computers for which the users must not take any action to run a Java application. Alternatively, the availability of free Java software and good manuals can help to achieve a smooth transition of such users to Java 2.

Group A (17%) represents computers in which the <applet>tag implementation does not allow Javascript to be executed inside the tag if Java is not present. In this case,

an error is subsequently generated in Javascript code; however, we catch this effect and store the correct information in the database. This group contains mainly IE 6 browsers on Windows and IE 5 on Mac, both without any Java installed.

Group B (10%) represents computers in which the <applet>tag does allow Javascript but Java was not available to the browser. This can occur for the following three reasons: i) there is no Java installed on the computer, ii) the Java installation is not integrated into the browser or iii) it is deactivated by the browser configuration. For reason i) there are two major cases. First, some MS Windows versions are not provided per default with Microsoft Java Run-time. Second, Firefox browsers come per default without any Java Run-time.

To assess the robustness of our statistics, the relative abundances of each category during the period of measurement is plotted against time (see Fig. 2). After some minor initial fluctuation, the proportions tend to become stable after two weeks of measurements. More recent and longer-term robust statistics can be obtained from the J-vestigator web site.

4. CONCLUSIONS

From our case study we can draw several conclusions. Specifically, for the bioinformatics user community we conclude that Java is a good option for programming of more advanced bioinformatics applications because of its wide presence, modern design and powerful network concept. Probably the most interesting fact is the big penetration of the recent versions of Java (1.4. and 1.5) in the biology community. The decision problem (between Java 1.1.X and Java 2) which led us to the presented work is not specific for bioinformatics, however the decision result itself depends very much on the Java versions structure for this particular community.

From the general software development point of view the proposed method has brought more predictability in the decision process in the early phases of our project. We know much more about our users and about the environment where the application will be running. This knowledge gives us also a possibility to address the specific user groups with tailored help procedures. Therefore we propose to include a similar measurement step in every Internet software project where nontrivial client functionality is needed.

Moreover, with today's possibility to build similar measurement systems as shared platforms, where users share their data and know-how, crystallization kernels of a larger distributed measurement system can be provided, which would collect and share various characteristics of the Internet software environment on the client computers. Such knowledge (or even control) would reduce development risk and improve the functionality of Internet applications.

To start the work on the above idea we provide software developers and users with up-to-date statistics about Java running on client machines in a public survey which will serve as a continuous Java statistics exchange platform. Developers using the proposed measurement system in other communities are welcome to share their results through J-vestigator. Measured data are free to download over the Internet at the J-vestigator Web page.

5. REFERENCES

[1] B. Dysvik and I. Jonassen. J-express: exploring gene expression data using java. *Bioinformatics*, 17(4):369–370, 2001.

[2] J. e. a. Johnson. Tableview: portable genomic data visualization. *Bioinformatics*, 19(10):1292–1293, 2003.

[3] S. Microsystems. Java.sun.com: The source for java developers. *http://java.sun.com*, December 2005.

[4] P. e. a. Zimmermann. Gene-expression analysis and network discovery using genevestigator. *Bioinformatics*, 10(9):407–409, 2005.