Shrink-Wrapped Java in Education

Deploying The Interactive Geometry Software Cinderella

Ulrich H. Kortenkamp and Jürgen Richter-Gebert

Institut für Theoretische Informatik, ETH Zentrum, CH-8092 Zürich, Switzerland,
{kortenkamp|richter}@inf.ethz.ch

Abstract. Java has been proven useful as middleware solution, for e-business applications and intranet applications. We present another project, *The Interactive Geometry Software Cinderella*, where the portability of Java is used for a product that is sold through standard distribution channels. The resulting problems which had to be solved are described and solutions are given. The experiences made are important and valid for other projects, too.

1 Introduction

Java has been accepted as a language which is very useful for middleware solutions, database front-ends and other, network related applications. Its widespread availability, the security model, and, most of all, its ease-of-use in programming make it the perfect language for almost all scenarios. With the introduction of Java 1.1 and later the Java 2 platform it has graduated from a "toy language" for web page gimmicks to a serious and important tool.

Despite its all-purpose suitability it seems like the language is not used for anything beyond these network based applications, and most Java software is deployed individually. In this article we want to present a non-mainstream Java project, *The Interactive Geometry Software Cinderella*. Cinderella is a computer based teachware (CBT) for geometry education on a high mathematical level which is completely written in Java. It uses the portability and web integration of the language to offer an easy method to communicate geometric constructions and to create interactive construction exercises and animations.

The step from the working prototype to the final *shrink-wrapped* product turned out to be very challenging and time consuming. Our experiences there are important for everybody who wants to create a Java application which is not only used internally, but which is distributed among a lot of different customers. The main issues are

- Code packaging and optimization, to decrease download time and enhance performance,
- Obfuscation, to protect intellectual property
- and Installation across different platforms.

In this article we describe how we could address theses issues using two third-party tools, Jax from IBM alphaworks[1], and InstallAnywhere from ZeroG Software[2]. This complements our report in [3].

2 Cinderella

2.1 Overview

Cinderella is a new computer based teachware for geometry. Its features are comparable to the two currently available geometry packages, Geometer's Sketchpad [1] and Cabri Geometry II [2].

A geometry software tool is used in education to create constructions with points, lines, circles and other geometric objects on a computer. The user should be able to "play" with these constructions later. This is the important difference with respect to ordinary drawing tools: It is possible to move freely placed points and lines, and the relations between the elements (like incidences, orthogonality, parallelism, etc.) are preserved. Using this "move mode" it is possible to explore the geometric properties of a construction (see Fig. 1).

Cinderella can handle more than only points, lines and circles. It also supports arbitrary conics, i.e. ellipses, parabolas and hyperbolas. Another feature is the automatic generation of loci: What is the path of a point when some other point is moved along a line or circle? An example is the mechanical three bar linkage shown in Fig. 2. The three black bars have a fixed length, the left and the right one are mounted at one end. The third bar connects the two others. The figure shows all possible positions of the center point of the third bar, which were automatically generated by Cinderella.

Another important possibility with Cinderella is the support of multiple, simultaneous views. The same construction can be displayed and manipulated in several windows at the same time. These views can be standard views like the Euclidean plane, but there are also display modes that support projection onto a sphere or the Poincaré disc model of hyperbolic geometry. For a more detailed description of the MVC architecture of Cinderella see Chap. 16 in [3].

The most exciting and Java-related part of Cinderella is the possibility to export Constructions, Animations and guided self-checking construction exercises to HTML combined with a runtime applet. This enables everybody to write interactive books or to publish his or her research easily on the web (see Fig. 3).

A demo version of the software as well as further information material can be found on the Cinderella website at `http://www.cinderella.de`.

2.2 History

The software has been developed in Java since August 1996, based on the experience made with a predecessor written in Objective-C on the NeXT platform.

[1] `http://www.alphaworks.ibm.com`
[2] `http://www.zerog.com`

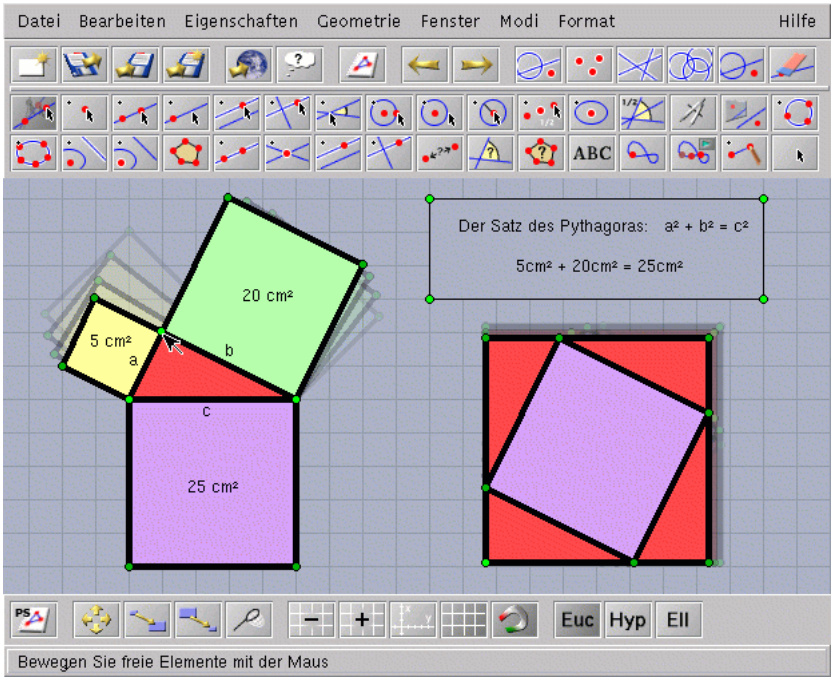

Fig. 1. A typical Cinderella session

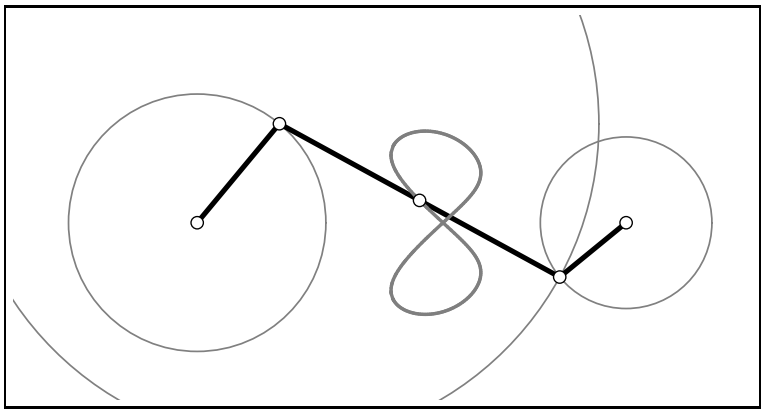

Fig. 2. The locus of the center point of the second bar in a three bar linkage. The circles show the lengths of the bars.

A first working prototype was done after three weeks, an improved version was submitted to the Multimedia Transfer '97 awards, where Cinderella was named "most innovative multimedia software" in January 1997, after a total of 18 man-weeks.

After that event we were trying to find a publisher for the software. Our intention was to just clean up the user interface and fix the known bugs to get a final product soon. Meanwhile Java 1.1 was released and we decided to switch to that version.

During 1997 we started negotiations with Springer-Verlag, Heidelberg, and HEUREKA-Klett, Stuttgart, which were willing to publish the software. Since the product is suitable for both academic and school use the publishers finally agreed to distribute two versions separately, one at the university level by Springer, one at school level by HEUREKA-Klett. The distinction between these versions is made mostly through the accompanying documentation, but they are using the same code base.

A short time after the first deadline for delivery to the publishers had passed we had a major mathematical breakthrough, which on the one hand enabled us to enhance the program a lot, but on the other hand forced us to rewrite the mathematical kernel completely. This, together with implementing the new features now possible, caused a delay of at least six month. But even without that breakthrough the completion of the project took much longer than we expected. The Springer version [5] was announced for November 1998, and it was shipped in late May 1999, whereas the Klett version is expected for August/September 1999.

Why did the production phase take so long? We found several reasons for it: A lot of time was used for fixing bugs that occurred only with certain virtual machines, especially the VM's in Microsoft Internet Explorer and Netscape Navigator. Another delay factor was that we only had a small testing crew, which even did not test full time. So most of the testing had to be done by ourselves. The double occupation of programming and quality assurance slows down the development heavily. A third reason was that most of the tools we used were or are still pre-releases or in beta testing. So the available resources were spent partially on third-party projects.

2.3 Development Environment

Due to the lack of good Java Integrated Development Environments at the time the project was started, we did all coding and debugging with the plain Javasoft JDK and XEmacs[3] as editor. The IDEs that became available later did not satisfy our expectations (see [3] for a detailed discussion). The development was mostly done on Linux, parts of it also on Solaris, and was carried out completely by the two authors.

In March 1998 we began to use CVS[4] as a revision control system, which saved a lot of time that was used to merge the two version before.

[3] `http://www.xemacs.org`
[4] `http://www.cyclic.com`

The only third-party class library used is ANTLR[5], a public-domain Java/C++ parser generator. The file format of Cinderella is read by ANTLR-generated code.

The software consists of approximately 43.000 lines of code, excluding ANTLR, with around half of them unique. This code is contained in 334 classes.

The build process is fully automated using GNU make. The Java compiler used is javac, which produces more efficient code than jikes (a much faster Java compiler available from alphaworks).

3 Why Java?

The decision for Java was made after a demonstration given at a mathematics conference which almost failed due to the unavailability of the right hardware and operating system for the old, NeXTStep Cinderella. It was clear that on the long run it will be harder and harder to find the right platform for Cinderella, and thus it was necessary to develop a new version which could be used on more common computers than the NeXT. Since we were anxious to make the wrong decision at that time and we heard of Java as a cross platform language, we gave this new language a try. It turned out to be the right decision.

Java gave and gives us the opportunity to run Cinderella on almost any platform used in academia (which are all different flavors of Unix), and at the same time on Windows and MacOS, which are the dominant operating systems in K-12 education. And, we are able to do all of our development work on Linux.

This platform independence is a major benefit, but in practice it is not that easy. Subtle differences between the Java Virtual Machines make life hard, and testing can only be done for a small subset of all target architectures. Nevertheless, we are quite content with the situation-as-is, since it is still much easier than porting a C- or C++-application. With Java we can rely on the same code base for all "versions" of Cinderella, which is extremely important.

Another big feature of Java is the seamless integration into web browsers (clients), even on non-Windows/non-Intel operating systems, as opposed to other, proprietary component models. Cinderella supports the easy creation of constructions that are embedded into HTML, which can be displayed using any Java-1.1 capable browser. This is done using a "Cinderella runtime applet", which includes all the code necessary to load, display and move constructions. This runtime applet can be distributed freely, so the user of Cinderella can communicate his or her constructions to others using the WWW (see Fig. 3).

4 Using an application extractor

In the book "Erfahrungen mit Java" [3] one conclusion of the Cinderella chapter was that it does not make sense to use code compression tools like Jax or DashO-Pro, since the results are not satisfactory compared to the effort needed to use

[5] `http://www.antlr.org`

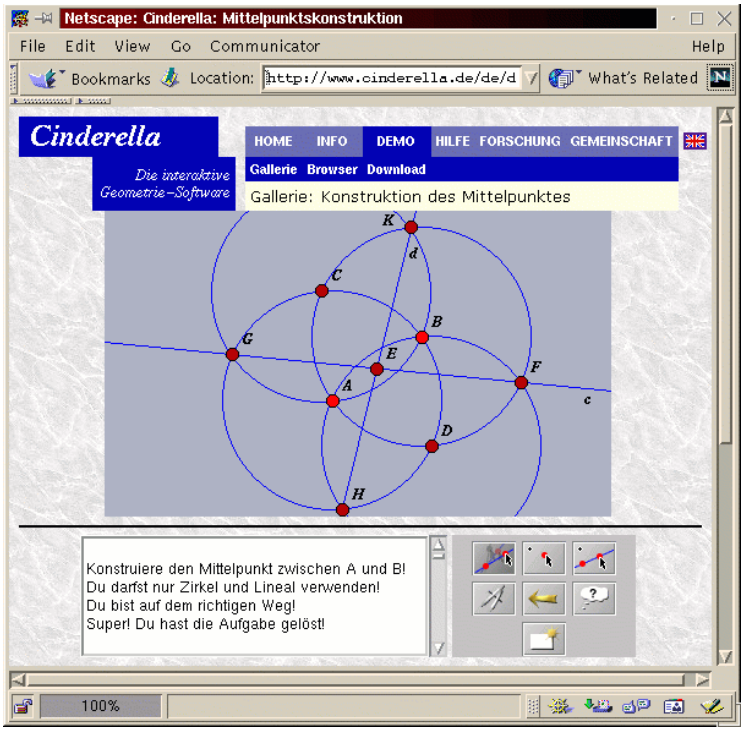

Fig. 3. An interactive exercise running in Netscape Communicator. Three separate applets are shown: The construction area (Euclidean view), a control panel and a message console. More examples can be found at `http://www.cinderella.de/demo/`

these tools correctly. Based on the experiences with newer release of Jax we made after this chapter was written we have to revise this completely.

An application extractor performs a class hierarchy analysis and a runtime analysis to remove unnecessary code from classes (for example due to overloading). It can also optimize the class code by inlining and other techniques. All these transformations are based on the observation that the dynamic linking provided by Java is only sparsely used in most applications. If it is known in advance which classes and methods have to support dynamic linking and virtual accesses, then all other references can be transformed to direct field accesses. To use an application extractor it is necessary to provide the names of the classes, methods and fields which are dynamically loaded or called. Only those classes, methods and fields which are not found by the runtime analysis have to be provided.

Further compression steps include the mapping of internal class, method and field names to shorter names. This "obfuscates" the class code, in order to make decompiled code less readable. Also, since the references stored in the class files are the full names of classes, methods and fields, it can save a substantial amount of memory. Another optimization step is the merging of classes and interfaces.

We now use Jax, which is available from IBM alphaworks[6]. alphaworks presents technologies created by the research division of IBM, and which might become commercial products later. Most of these technologies are Java related. Unfortunately, most of them may only be used non-commercially, but in our case it was possible to get a commercial license in exchange for the extensive testing we did with Jax.

For us, the use of Jax addresses several problems:

1. **Same code base for application and applet.** The Cinderella applet should be able to display all Cinderella constructions, but it should not contain all the code of the standalone application, since it may be redistributed freely. By using Jax we can specify which classes are dynamically loaded by the application resp. the applet and only these will be included in the final Java archive. Moreover, the runtime analysis of Jax determines which methods and fields of these classes are used in the application or in the applet, and only these will be available. This eliminates, for example, the construction saving code.

   This means that we can use the same code base for the application and the applet, without having to worry that the free runtime contains code which should be available in the application only. The advantage of a single code base is clear: Both versions stay compatible all the time and bug fixes are applied to both the applet and the application.
2. **Code obfuscation.** For most developers it is a drawback of the Virtual Machine concept that the Java byte code is easily decompilable into human readable Java source code. Even some IDEs offer this option. It is a major concern that intellectual property cannot be secured as it has to be.

[6] `http://www.alphaworks.ibm.com`

There exists several "obfuscators" which rename class, method and field names to short and meaningless names, and again this is build into some IDEs. But these obfuscators make it only a little bit harder to understand the code.
Jax does code obfuscation, too, but together with the other class transformations applied by Jax the byte code becomes un-decompilable. Even if a special decompiler is available – which translates back the direct field and method accesses – then the Java source will not at all be useful, since it does not necessarily reflect the class structure of the original code. Without the log files of the Jax transformations it is virtually impossible to map the compressed code to the original source.
This protection could be one of the most important applications of Jax, and we recommend Jax to everybody who wants to protect his intellectual property.

3. **Code optimization.** The removal of virtual calls and field accesses as well as the inlining of methods speed up Java applications by a significant factor. This is especially true for server applications, or, vice versa, the performance gain is not that significant for graphics intensive applications, since these do not benefit from this optimization.
Another important factor is the size of the Java byte code, especially for the applet which is downloaded across the Internet. Here Jax did a great job: The final jar file is about one third of the size of the original one (see Table 1).

Table 1. Code compression with Jax

	Cinderella Applet				Cinderella Application			
	zipfile	methods	fields	classes	zipfile	methods	fields	classes
before:	896758	3927	2754	383	896758	4033	2817	412
after:	306176	1985	1530	247	398657	2527	1795	292
savings:	590582	1942	1224	136	498101	1506	1022	120
	65%	49%	44%	35%	55%	37%	36%	29%

More information on successful compression with Jax can be found in the research report by the Jax authors [6]. This report also covers the compression of Cinderella, since Cinderella was used by the Jax team for performance and regression tests. As a side remark we want to mention that the size reduction could be increased further if another class file layout could be used (e.g., as suggested by Pugh [4])

When using Jax for your own development you should keep in mind that it is still pre-beta software. It is not guaranteed to work at all, and even if one

version worked for you, you cannot rely on the fact that the next update will still work. We were successful with a pre-version of Jax 4.1 and with Jax 5.2, all other versions of Jax produced incorrect code or no code at all. Despite this warning, we are sure that tools like Jax will help Java as a language to overcome two serious problems: the readability of byte code and the performance penalty due to virtual accesses.

5 Preparing for distribution

The targeted user group of Cinderella is very inhomogeneous. We cannot assume anything about the platforms Cinderella will be used on. Despite the fact that we could make Cinderella run on any platform that supports Java 1.1, we cannot hope that the users will be able to make Cinderella run if we only provide the jar file containing all the classes. In that situation the use of a third party tool to create a cross platform installer seemed to be a good choice.

We found the following requirements for us:

- The installer must be able to install a Java Virtual Machine on Windows 95/98/NT and MacOS, for the case that there is no JVM available.
- The installer must run on any Java-1.1 platform.
- The installer must be internationalized, in particular it must provide a German install.
- The installation should "look like any other installation on Windows", that is, a Windows user must not be confused with details about Java.

There are several cross platform installers for Java available (InstallShield for Java[7], J'Express[8], InstallToolkit[9] and InstallAnywhere[10]). Of these, only InstallAnywhere is able to install a JVM on MacOS. This was reason why we chose InstallAnywhere (IA) as installation toolkit for Cinderella. It also supports all the other requirements we listed above and even has some more interesting features. The installers created by InstallAnywhere are of high quality and integrate seamlessly into the Windows operating system, and offer the same ease of use on other platforms, including Unix. This makes them usable also for non-experts.

Despite its features, IA has several drawbacks, which we want to mention.

1. **Not fully scriptable.** Our development environment is based on GNU make and almost everything in the build process is automated. IA can be run from the command line, but it is not possible to pass a machine generated configuration script to it. All configurations of the installer have to be done using the graphical user interface of IA, which is very time consuming and error prone.

[7] `http://www.installshield.com`
[8] `http://www.denova.com`
[9] `http://www.alphaworks.ibm.com`
[10] `http://www.zerog.com`

IA offers a concept called "speedfolders", which can partly automate the build process, but they are not powerful enough to suit our needs. A speedfolder defines a directory which should be recursively included in the distribution. Inclusion or exclusion rules (but not both) based on suffix matching can be defined. Even with these rules it can happen that empty directories are included.

2. **No version control support.** The binary file format of the IA configuration files is not suitable for true versioning with CVS. Even worse, IA saves its localization information in a separate directory, and if this directory is added to the CVS system the extra `CVS` directory in that directory causes IA to crash. These problems could be avoided with a fully scriptable version of IA.
3. **Incomplete internationalization support.** Although the most recent version of IA supports creating installers in six languages (additional languages are available separately), the internationalization support is far from being perfect. It is not possible to create an internationalized version of an already existing installer. There is no built-in support for different custom messages for the different locales, this is only available via post-editing by hand of some configuration files.

   With respect to internationalization we have to recommend J'Express, which is much more powerful in that area. Unfortunately, J'Express does not support installing a JVM on MacOS. Other remarkable features of J'Express which make it a good alternative if you can do without MacOS support are silent installs (which can be run unattended) and automatic updates.
4. **Incomplete JVM support.** With the current version IA 2.5.3 it is possible to use different Java virtual machines than the included ones. However, it was not possible to replace the Windows JRE (Java Runtime Environment) shipped with IA with the new IBM high speed JRE, probably due to incompatibilities with native calls[11]. More annoying is the fact that it is not possible to replace the MRJ (Macintosh Runtime for Java) 2.0 shipped with IA with the MRJ version 2.1.1, since this MRJ needs a reboot during install, which is not supported by IA. This means that an installation with IA cannot install the best possible JVM on either Windows or MacOS. If these JVMs are installed prior to running the installer of Cinderella they can be used, but this needs another user interaction which we wanted to avoid in the beginning.

With InstallAnywhere we were able to create an installer which can be run from CD-ROM or even remotely via the web[12]. We are very satisfied with this installer, although the creation of it is not as easy as it could be.

[11] ZeroG Software told us that they are working on that issue.
[12] You can test it at `http://www.cinderella.de/demo/download.html` .

6 Conclusion

It is possible to deliver Java software to a broad audience. The users may even not recognize that they are using a Java product. This makes Java a valuable language for applications that have to address a lot of different platforms and users, like academic software. The installation tools that are currently available are a good help for this task, although they could be improved.

A wise decision is to use a code compression tool like Jax, since it reduces the application size, optimizes virtual calls, and obfuscates the program to protect intellectual property.

References

1. Jackiw, N.: The Geometer's Sketchpad, Key Curriculum Press, Berkeley (1990–1995)
2. Laborde, J.-M., Bellemain, F.: Cabri-Geometry II, Texas Instruments, Dallas (1993)
3. Maffeis, S., Toenissen, F., Zeidler, C.: Erfahrungen mit Java, dPunkt-Verlag, Heidelberg (1999).
4. Pugh, W.: Compressing java class files, in: Proceedings of the ACM SIGPLAN '99 Conference on Programming Language Design and Implementation, Atlanta, GA, (1999).
5. Richter-Gebert, J., Kortenkamp, U.: The Interactive Geometry Software Cinderella, Book & CD Edition, Springer-Verlag, Heidelberg (1999).
6. Tip, F., Laffra, C., Sweeney, P.: Practical Experience with an Application Extractor for Java, Tech. Rep. RC21451, IBM T.J. Watson Research Center (1999).