A moribund[1] Smalltalk still alive and kicking: The APIS VisualSmalltalk IDE[2]

Thomas Brey, Heiko Wagner

Thomas.Brey@apis.de; Heiko.Wagner@apis.de

APIS Informationstechnologien GmbH
Im Haslet 42, D-93086 Wörth a.d. Donau, Germany

Abstract: After we briefly described the history of Visual Smalltalk (VS), we show how some proprietary rather complex derivates of VS emerged over the years. Despite the fact that the last official major release was in 1995, there are still successful products actively developed with VS. We show in detail the current status of VS at the APIS company, focussing on usability improvements of the IDE as well as on the proprietary Java Interface, which was developed independently from JNIPort and JavaConnect, featuring a high level of automation and integration (fully automated import of jar-files, browse and compile Java in VS). We show that even a smaller company can succeed in keeping a 13 year old Smalltalk IDE up-to-date.

1 History and Background

A System called *Methods* introduced by Digitalk in 1984 is usually taken to be the first commercial Smalltalk available and it was precursor of Digitalk's more popular. *Smalltalk*/V, first published in 1986 for DOS-based OSs, later for OS/2 and Windows.

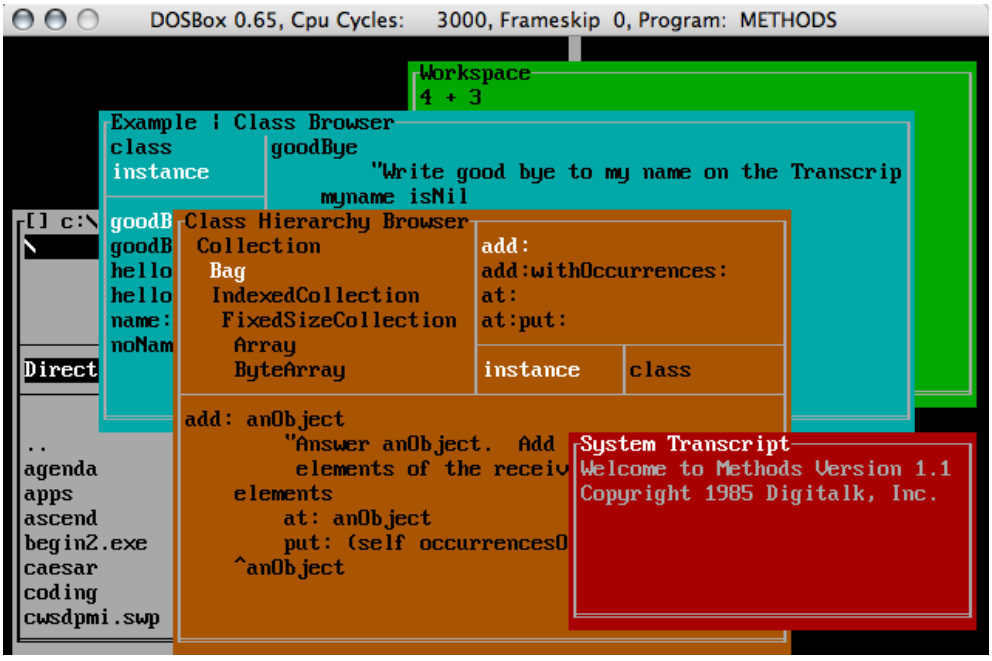

Fig. 1: Digitalk Methods 1.1 from 1985 (screenshot by Aaron Reichov[3])

Until 1995, *Smalltalk/V* was the main competitor to the ParcPlace (PP) Smalltalk Systems, *ObjectWorks* and later *VisualWorks*. Compared to PP Systems, *Smalltalk/V* was not platform-independent but deeply integrated into and optimized for the supported platforms. It targeted the expanding PC market and compared to PP, it's pricing was moderate and especially the 16-bit Version 2.0 for Windows released in 1992 (later given away for free as *Smalltalk/Express*),

[1] James Robertson from Cincom once applied this term to VS
[2] Submitted to ESUG 2008
[3] http://bitquabit.com/~revaaron/old/

became widely used. With Version 3.0, the product was renamed to *Visual Smalltalk*; it was Windows95 certified and released in 1995, shortly before PP and Digitalk merged to ParcPlace Digitalk (PPD). In 1996 PPD released Version 3.1 available with numerous add-ons as *VisualSmalltalk Enterprise* (*VSE*). One was the *PARTS* Workbench for visual programming, which had some impact on IBM's *VisualAge*, another was *Team/V* for team programming and version control, whereas *ENVY*, another widely used team and version control environment available for nearly all Smalltalk Versions, was no longer supported in *VSE*. Although it was a declared goal of PPD (renamed 1997 to ObjectShare), to unify the *VisualWorks* and *VisualSmalltalk* product lines, the company completely failed and was dissolved in 1999. Whereas complete IP Rights on *VW* were sold to Cincom, Cincom aquired only the rights to sell and maintain existing versions of *VSE*, all other rights on *VSE* were sold to a company called Seagull (which was not really interested in *VSE* but only in tools developed with it). This deal and the resulting "license deadlock" actually killed official development of *VSE* and although Cincom published the maintenance release 3.2 aka VSE2000 in 1999 containing some fixes but no new features, the last patch for the VM published by Cincom dates back to 2002 and meanwhile support for *VSE* is officially discontinued.

But since *VS* once was a rather successful product, several companies decided in the early 90's to realize their projects and products with *VS*. Some of these products established a strong market position and are still actively developed in *VS(E)*. Since active development of *VS(E)* itself stopped at the end of the decade, a community of engaged *VSE*-Developers emerged which is still active on the VSEW mailing list sharing bug fixes, enhancements and customer components. Most notable contributions to the community[4] are the "*VSE* Goodies" provided by A. Reimondo[5] and Tec4[6], both still maintain their download sites. Whereas these goodies were designed for general usage, many companies adapted VS to their needs over the time leading to rather complex, proprietary derivates of *VS(E)*. Without doubt, a major contributing factor making these systems possible is the fact that Smalltalk is an open system where you can modify and extend nearly everything; except some primitives, even core classes like String etc. can, in contrast to Java e.g., be modified. The price for this flexibility enabling even small companies to keep their *VS* up to date, is the higher effort to share custom modifications and components between different *VS* derivates, because they are usually based on different language cores and windowing frameworks. But a still harder problem the remaining *VS(E)* community is continuously faced with, are changes in the architecture and APIs of the underlying Windows platform. Often, these changes would require adaption of the virtual machine (VM), which is not possible for the ordinary *VS(E)* developer. Although Seagull never showed any interest to support the *VS(E)* community (e.g. open source the VM), some companies individually succeeded in buying the source code and the right to modify the VM from Seagull; this in turn made the necessary modifications of the VM possible and thus helped to keep the community alive (not by sharing the VM, but the knowledge to patch it). Lesser-Software[7] took another road out of nowhere: instead of modifying the VM of *VS*, they developed an enhanced but byte-code compatible VM from scratch.

APIS started developing risk and quality management software based on *VS* and *ENVY* in 1992. Since *ENVY* was used for team programming and version control, APIS never upgraded to *VSE* but still uses *VS* 3.0 from 1995 (see Fig. 2).

From the very beginning, usability and an intelligent user interface was a major target and the visualization techniques developed at APIS are still a distinguishing feature compared to competitors and helped to establish a market-leading position. This in turn made it possible for

[4] I want to note some persons from the VSEW mailing list, who contributed valuable help through ongoing discussions: Henrik Hoyer, Manfred Möbus, Todor Todorov, Dan Poon, Frank Lesser, our collegue Andreas Rosenberg and last but least Thomas Muhr who maintains the list.

[5] http://www.smalltalking.net/Goodies/VisualSmalltalk/index.htm

[6] http://www.tec4.ca/Smalltalk/

[7] www.lesser-software.com

APIS to expand continuously, so APIS can not only afford to enhance the IDE besides developing the products, but also succeeded in buying the source code for the VM and implementing some major improvements to the VM like Unicode capability. In section 2 we briefly describe enhancements incorporated in the products, section 3 deals with enhancements of the IDE and in section 4 we briefly illustrate our Java Interface, which allows one to use Java classes and components within Smalltalk thus also extending the limited capabilities of the old original VS.

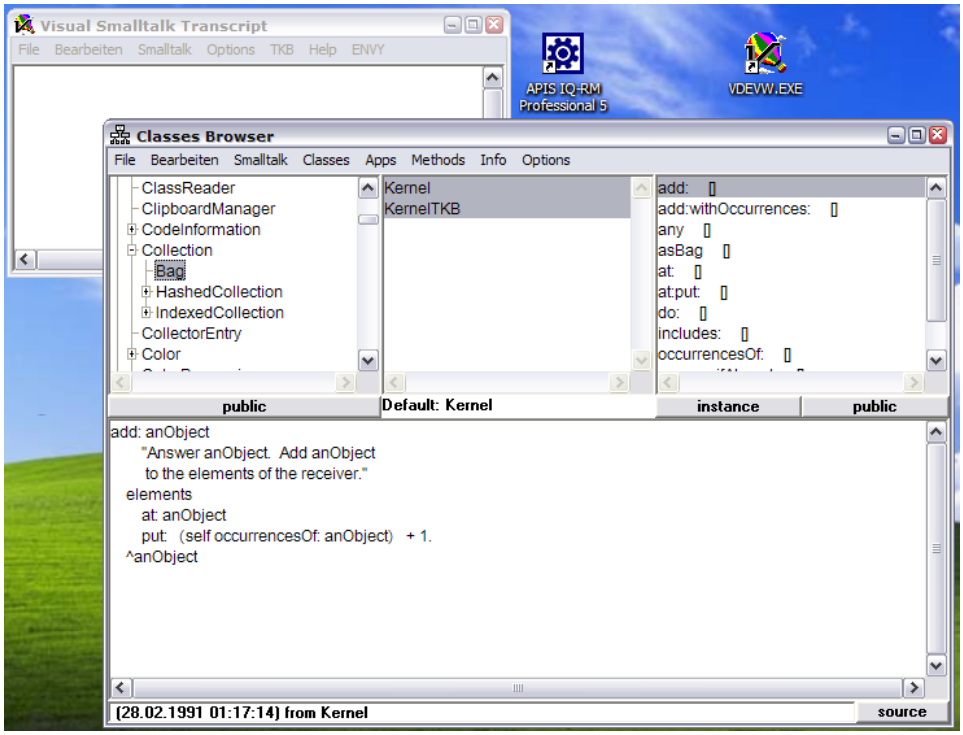

Fig. 2: Digitalk VisualSmalltalk 3.0 from 1995

2 APIS Frameworks and Enhancements

As already noted, the sophisticated UI of APIS products had a major impact on the success of these products. We pick out two examples, which were developed very early: APIS TreeView and APIS TableVision. As shown in Fig. 3, the TreeView control not only supports multi-colored text for a single tree item; it also supports multiple icons for a single item, where each icon can trigger a different action when clicked.

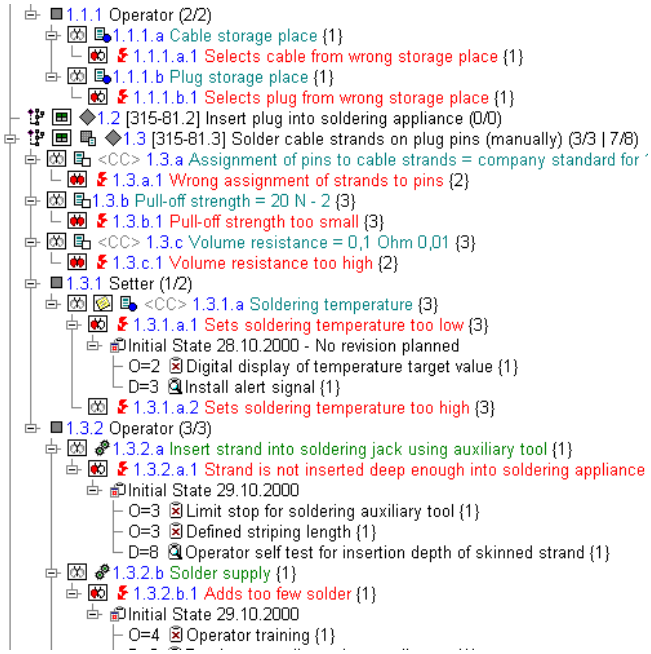

Fig. 3: APIS TreeView Control

APIS TableVision is a framework for rendering arbitrary complex objects in arbitrary nested tables:

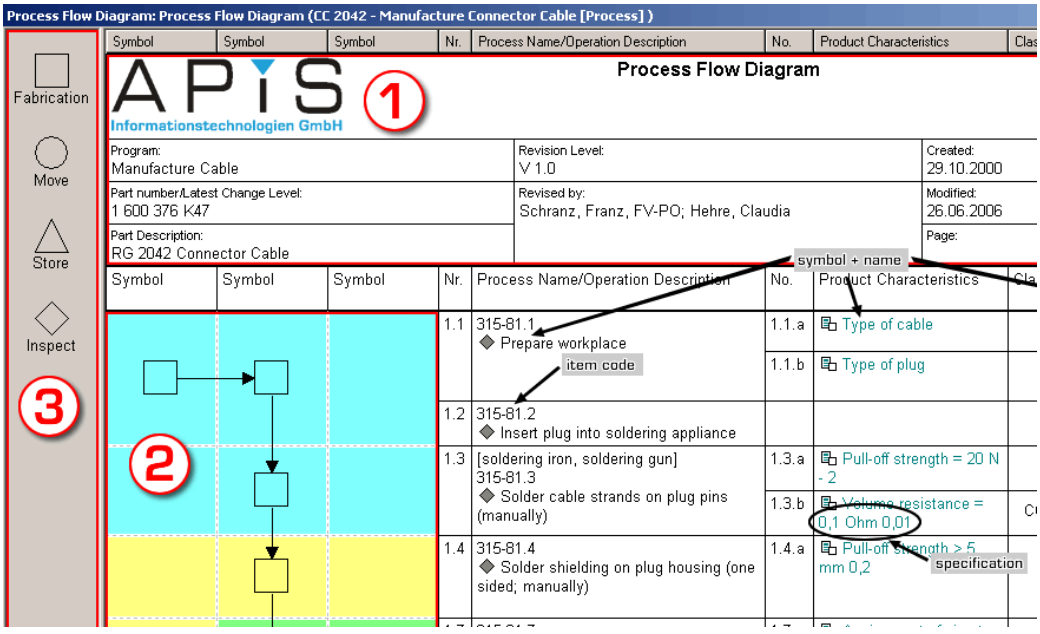

Fig. 4: APIS TableVision

Since APIS products are used worldwide, localization became an important goal. Whereas a framework supporting multiple languages for the data as well as for the UI was already implemented at the beginning, the Base System of VS (core classes, windowing system and VM) did not support Unicode, making it impossible to render e.g. Chinese or Russian correctly. In an ongoing effort taking 3 years, APIS finally succeeded in the middle of 2007 in implementing full Unicode support for the Base System through a very transparent framework[8].

[8] partly inspired by previous work of Frank Lesser

Keeping in mind, that APIS does not use VSE but VS, which lacks some essential components, we conclude this section by simply listing some more enhancements which were developed over the years:

- a proprietary, object-oriented database
- a framework for parsing/generating SGML,XML and HTML
- TCP/IP and UDP based networking support, SMTP Client etc.
- Support for various native widgets, MAPI, ODBC, the windows registry etc.

3 Development Support

Although ENVY already extended the plain VS IDE with several Browsers, e.g. an ApplicationsBrowser similar to the PackageBrowser of VSE (a bundle of classes and methods what is usually called a Package is called an Application in ENVY), the original VS/ENVY IDE (see Fig. 2) lacks essential features of modern IDEs like Autocompletion, Code Highlighting and Formatting, Refactoring Support, intelligent Find&Replace of code fragments. To make work more comfortable, APIS was always engaged in integrating such features into the IDE.
First, an early version of the Refactoring Browser was ported from VisualAge and continuously improved. Then the SUnit Framework was implemented with visual support through our UnitTest Browser. Finally, an enhanced ClassesBrowser was developed from scratch. Since our IDE now contains several different browsers, we currently work on generic frameworks which allow us to use new features in any browser. One example is our IDEToolbar framework, which enables us to easily attach toolbars for basic functions to any window of the IDE (see Fig. 5). Most toolbars contain state-of-the-art controls for navigating back and forwards in the browsing history and for searching Text or Classes, where the Comboboxes (e.g. for Class Search) are able to filter their contents (up to 25000 entries) in real-time during typing. Another example for a generic concept which can be applied to all browsers is our CodePane, a Text control optimized for rendering Smalltalk code and providing not only basic support like Autocompletion, Highlighting and Formatting but also more sophisticated features like visual feedback on problems found in the code, assistance to resolve them or direct in-place renaming (e.g. instead of doing a find&replace over all occurrences of a variable in a dialog box, you simply edit the variable occurrence in the declaration and all occurrences in the text are edited concurrent and automatically). All features of the CodePane are based on a general framework for Code Processing, which emerged through continuously extending the RBParser component from the RefactoringBrowser (ver. 3.0), but independent form development in other Smalltalk Systems (e.g. RBHighlighter in VisualWorks). Goal of our CodeProcessing Framework is to support the developer as much as possible and to detect possible runtime errors and other potential problems already at the time a method is edited or saved. This, of course, is hard when dealing with an untyped language like Smalltalk. We therefore work on a Type Subsystem, which similar to Strongtalk, tries on the one hand to infer type information from assignment and return expression found in a message chain and on the other hand supports (optional) type declaration for return values, arguments and variables. Information collected this way can then be used to improve Autocompletion or to detect “type errors”, which would throw a MessageNotUnderstood error at runtime.

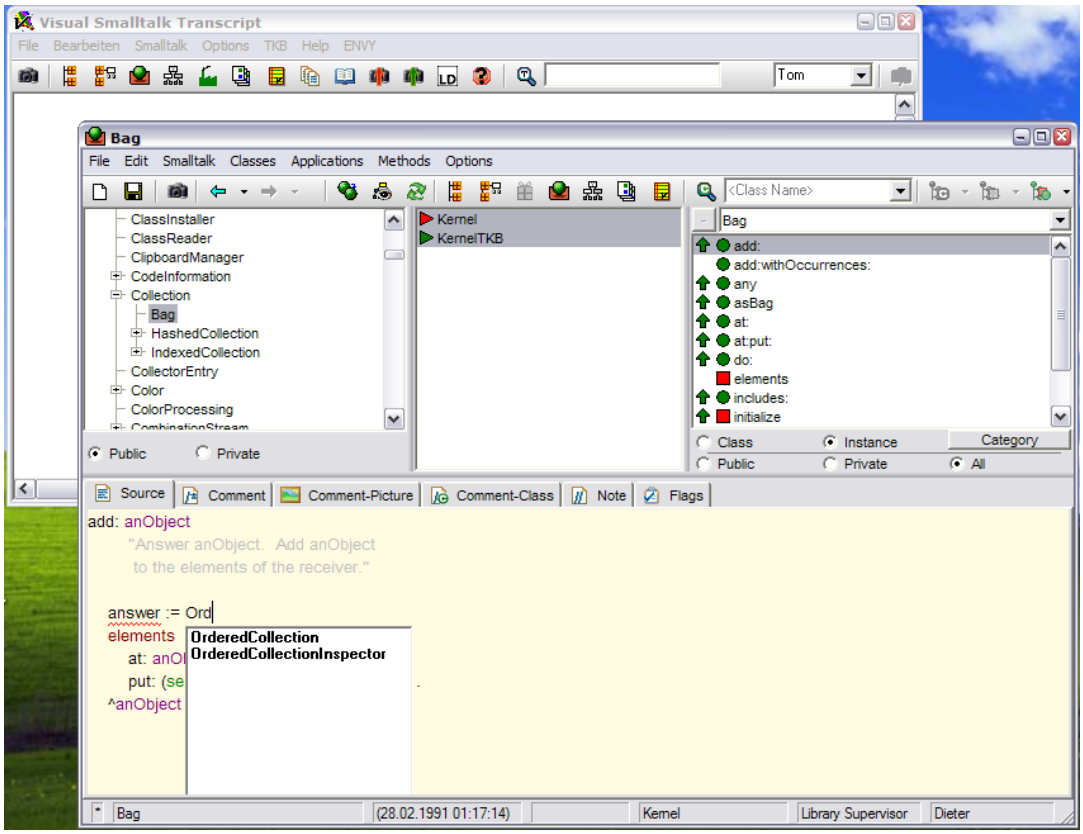

Fig. 5: APIS VisualSmalltalk IDE today

4 Java Integration

Since the beginning of 2007, we develop a framework for using Java within Smalltalk. At the time we started, we did not know of the two similar projects JNIPort and JavaConnect. All we knew was the prior work of A. Reimondo on the same topic, but his implementation is rather old did not install in our environment, so we quickly decided to build our Java Interface from scratch.

Low Level Interfacing with the Java Virtual Machine using JNI

The foundation of the interaction between the Java VM and Smalltalk is Java Native Interface (JNI). It allows invoking Java functionality from native code. It has originally been designed to allow interaction with the programming language C/C++, but the Smalltalk VM was enhanced to take use JNI as well, thus allowing Smalltalk to interact with Java objects as they just were regular Smalltalk objects. Enhancements of the Smalltalk VM include functionality to directly call Java VM code using function pointers, as well as adding support for 64bit integer types and IEEE 754 floating point formats not originally available in the Smalltalk VM.

Support for Java Types

The full set of Java types is supported, this covers any Java object and all basic types like byte, char, int, long, float and double. Also arrays, or nested arrays, containing Java objects or basic Java types are possible. Unicode support has been added to the Smalltalk platform to take advantage of internationalisation (i18n) facilities in Java. Special support is also provided for the Java BufferedImage class. Instances of this class can be converted to a Smalltalk Bitmap instance and vice versa. This gives access to a wide range of Java functionality like Java Image IO, Java Advanced Imaging and Java 2D.

High Level Tool Support

In addition to the low level support, a set of tools was created to make using any Java library a simple process. By providing the possibility to automatically generate all needed Smalltalk wrapper code from arbitrary any Java library by simply importing a set of JAR files (see Fig. 6), so the complexity is completely hidden and the Smalltalk developer can focus on writing just Smalltalk code. Access to the entire Java SE 6.0 runtime has been provided by this way, thus providing several thousand classes containing several ten thousands of methods. The Smalltalk IDE class browsers have been augmented to provide capability of displaying the source code provided by the Java libraries (see Fig. 7). It is even possible to compile a Java class directly from within the Smalltalk IDE, thus providing the tightest integration of Smalltalk and Java.

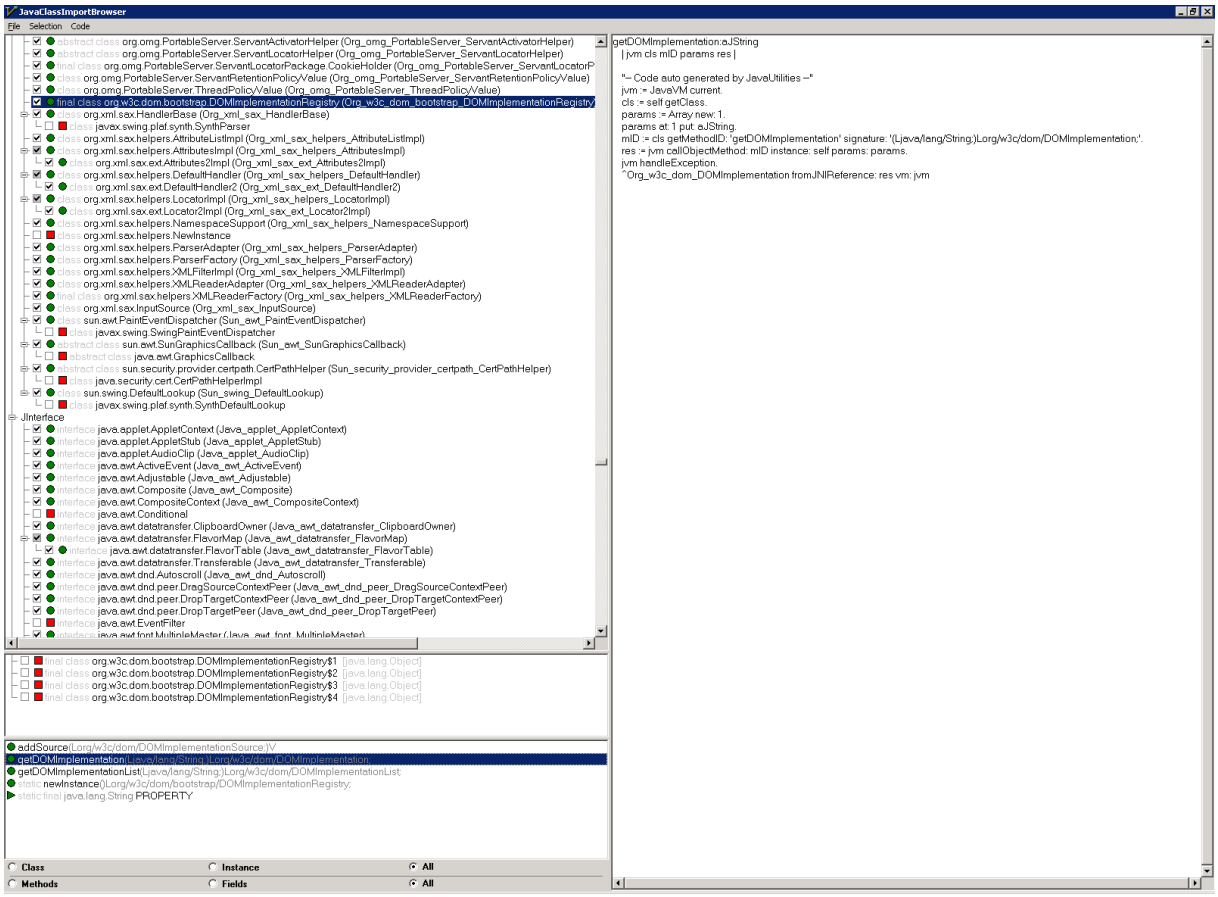

Fig. 6: Java Import Browser

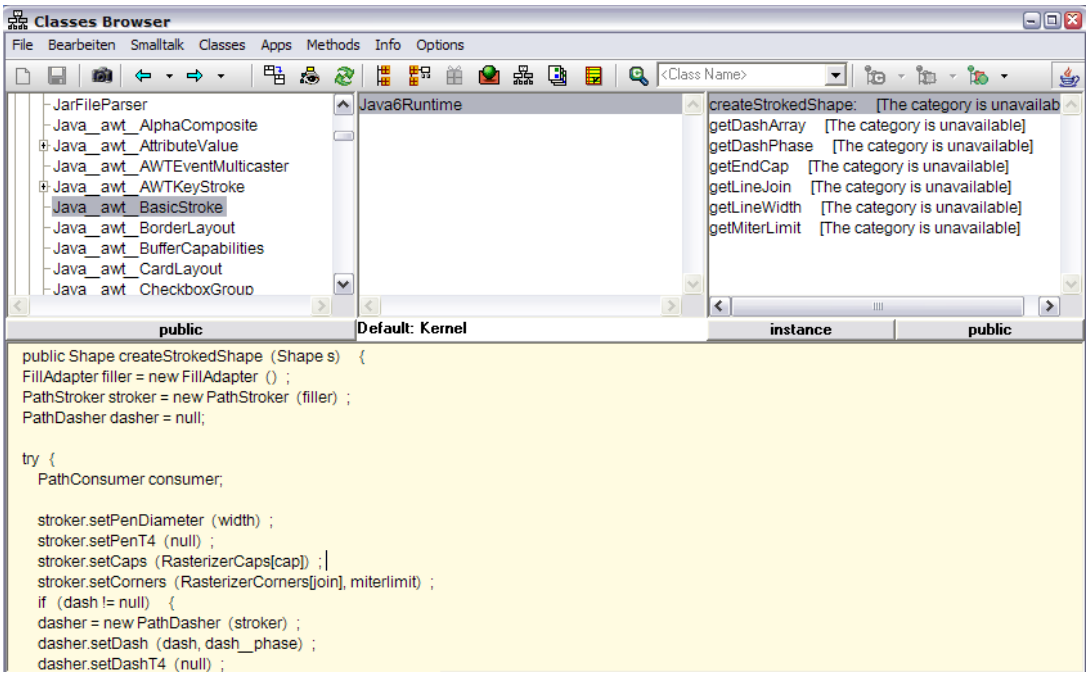

Fig. 7: Browse+Compile Java Code in a Smalltalk Browser

Integration of Java UI Components using AWT native interface

Besides the integration on the code level there is also a complete integration of Java user interface into Smalltalk. Java user interface components like Swing components can be used in any Smalltalk GUI just like ordinary Smalltalk components. A bridge has been created to allow Smalltalk programmers to register event handlers for these hosted Java components just like they do for ordinary Smalltalk components, so Java components integrate seamlessly into any Smalltalk user interface.