Embedded System Workflow Execution with jABC and jAWS

Benjamin Hanzelmann

Chair of Programming Systems, TU Dortmund

benjamin@hanzelmann.de

Abstract

This paper addresses the Java Application Workflow System (jAWS), which is inspired by the JavaABC framework and was designed in the context of the “Ricoh & the Sun Java Platform – Powered by U!” contest. JavaABC is a flexible and powerful framework for service development based on Lightweight Process Coordination. Users easily develop services and applications by composing reusable building-blocks into (flow-)graph structures that can be animated, analyzed, simulated, verified, executed, and compiled. This document explains how jAWS and JavaABC can be used for embedded system workflow execution.

1 The Setting: Enhancing the Ricoh MFP

The “Ricoh & the Sun Platform Powered by U!” contest was held for the second time in 2006. The contest aims at challenging developers to pioneer innovative Java-based applications for Ricoh’s Aficio multi-functional products (MFP). 32 universities and more than 120 students participated in this contest. There were no specific requirements regarding the type of application to be produced.

The Ricoh Aficio MFP supports the Java Mobile Edition[9] (J2ME) with the Connected Device Configuration (CDC) plus the Foundation Profile (FP). J2ME is a Java environment defined by Sun Microsystems for applications running on consumer devices. The different configurations relate to different classes of devices, ranging from small mobile phones to more powerful devices like Java-enabled printers. Profiles provide additional library support by defining sets of supported libraries. The Foundation Profile[8] is the most basic profile for CDC[7]. With CDC+FP, most of the basic Java libraries regarding networking, file operations and so on are directly supported.

Although the MFP is much more powerful than e.g. mobile phones, it still has limitations regarding CPU power and memory usage. This observation led to the idea to use external resources to execute tasks that need more memory or CPU-time than the printer is able to provide. Fortunately, the Electronic Tool Integration platform (ETI) developed at the Chair of Programming Systems, University of Dortmund, acts as an easy to use communication layer between Java clients and remote 3rd party services. Further details on the ETI platform will be covered on page 3.

Another limitation of the device is the small screen and the lack of fast and easily usable input devices for the complex task of creating and configuring a workflow. These workflows consist of small reusable tools, combined in an execution sequence. A plugin for executing ETI workflows is already part of the JavaABC[6] (jABC) distribution. We decided to combine the possibility of executing a workflow directly from jABC with the generation of a description of the workflow, which can be used as a tool on the printer.

2 JavaABC

The JavaABC (or jABC) is a framework for service development based on *Lightweight Process Coordination*[5]. Predecessors of jABC have been in use since 1995 to design industrial telecommunication services and Web-based distributed decision support systems, among others.

jABC allows users to easily develop services and applications by composing reusable building-blocks (SIBs) into (flow-)graphical structures. The development process is supported by an extensible set of plugins that provide additional functionality in order to support all the activities needed along the development life cycle, e.g. animation, rapid prototyping, formal verification, debugging, code generation, and evolution.

It does not substitute but rather enhance other modeling practices like the UML-based RUP (Rational Unified Process[2]), which are in fact used in the modeling process to design the single components used in the model.

Lightweight Process Coordination offers a number of advantages for integrating off-the-shelf, possibly remote functionalities:

- **Simplicity**. jABC focuses on application experts, who are typically non-programmers. The basic ideas of the modeling process have been explained in past projects to new users in less than one hour.

- **Agility**. It expects requirements, models and artifacts to change over time, therefore the process supports evolution as a normal process phase.

- **Customizability**. The building blocks which form the model can be freely renamed or restructured to fit the habits of the application experts.

- **Consistency**. The same modeling paradigm underlies the whole process, from the very first steps of prototyping up to the final execution, guaranteeing traceability and semantic consistency.

- **Verification**. With techniques like model checking and local checks the user is supported while consistently modifying his model. The basic idea is to define local or global properties that the model must satisfy and provide automatic checking mechanisms for these properties.

- **Service orientation**. Existing or external features, applications, or services can be easily integrated into a model by wrapping them into building blocks used inside the models.

- **Executability**. The model can have different kinds of execution code. These can be as abstract as textual descriptions (e.g. in the first animations during requirement capture), and as concrete as the final runtime implementation.

- **Universality**. Thanks to Java as a platform independent, object-oriented implementation language, jABC can be easily adopted in a large variety of technical contexts and application domains.

3 ETI

The Electronic Tool Integration Platform[1][3][4] (ETI) is an online remote execution framework designed to support the distributed use of and experimentation with tools over the internet. ETI is unique in allowing users to combine tools of different application domains to solve problems a single tool never would be able to tackle.

jETI is the current implementation of the ETI platform. It enhances other applications and frameworks by easy integration, organization and execution of remote functionalities. It provides a plugin for jABC that allows to embed of jETI SIBs into graphs. On execution of the graph, these SIBs are not executed locally as normal SIBs are. They represent a function call to a jETI server that is contacted at each execution of the SIB.

The server-side Tool Executor connects the outside interface, which is reachable by SOAP[10] remote procedure calls, with the executable tool itself. The necessary configuration is done by naming the Java class to execute and the parameters it accepts. This can be done via the HTML Tool Configurator, which allows registering new tool functionality by filling out a simple template form. It is also possible to retrieve the configuration from a server and generate SIBs that can instantly be used to design workflows in jABC.

The generated SIBs have the same parameters as the server-side tool. On each execution of an jETI-SIB, the input data for the tool is transferred to the server and the tool is executed with the parameter-values given in the SIB. The result of this is transferred to client. All this is happening transparently, the jABC user does not need to know which SIBs are executed locally or remotely.

4 jAWS

The Java Application Workflow System (jAWS) utilizes the same techniques as the jABC with the jETI plugin. The Tracer plugin for the jABC executes a workflow by executing the building blocks one after another, deciding in each step which path to go next. The chosen path depends on the return value of the executed code. The jETI plugin provides a delegation for remotely executing the SIBs marked as an ETI service on the appropriate server.

The jAWS plugin does not directly execute workflows defined in the jABC. It rather translates them into process descriptions for the jAWS tracer engine running on the MFP. The SIBs used for building workflows have to be compatible with jAWS, which means they have to be provided by the jETI or jAWS plugin. These types of SIBs reference tool definitions and can be executed by the jAWS tracer on the Ricoh MFP.

jAWS distinguishes three kinds of tools for the execution of workflows on the MFP itself:

- **Atomic Tool**. These tools do local work using the capabilities of the MFP, like scanning and printing, or reading and writing data using the local filesystem or network servers.
- **ETI Tool**. An ETI Tool communicates with an ETI server. It enables the MFP to use services provided by a remote server, using the SEPP protocol for communication.
- **MetaTool**. A MetaTool combines other tools. It encapsulates a sequence of tools and can be used inside a workflow, where it represents the contained sequence.

The generated workflows are processed by the jAWS tracer, which orchestrates the control flow in a sequence. It executes the following steps:

- *Expand* MetaTools in the workflow.
- *Compute* data paths and determine which data has to be downloaded from the jETI server.
- *Execute* the tools one after another.

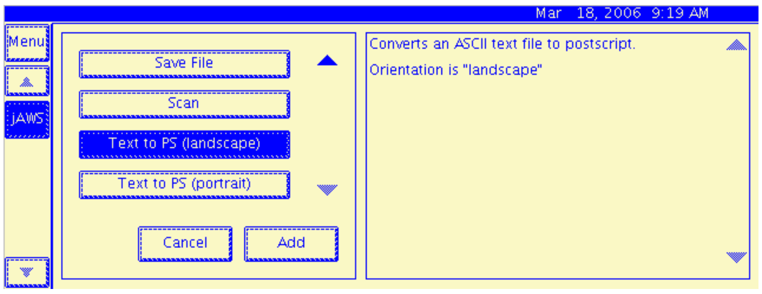
By inserting MetaTools into generated workflows they can be built hierarchically. Any previously built sequence can be used as a MetaTool, introducing the possibility of reusable modules encapsulated in the process descriptions.

4.1 Design Process

The first version of jAWS used the touch screen on the MFP to create and manage workflows. That means rearranging tools inside a workflows and editing parameters of these tools, setting defaults and so on. Even those quite simple operations turned out to be too complex to be represented on the MFP, rendering more advanced features like real hierarchy and branching functionality nearly impossible to use.

The touch screen and the rather high latency while handling user input unnecessarily complicated the design process, and led to the development of the jABC plugin (see page 6).

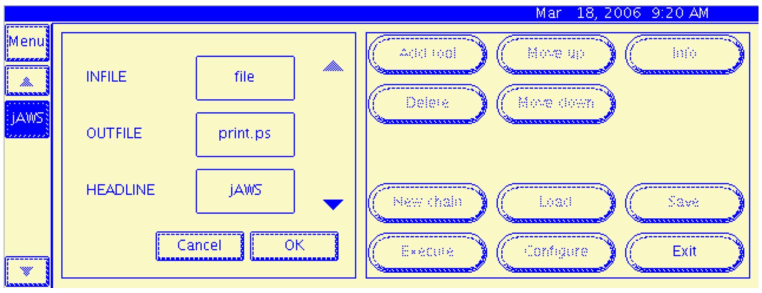
Editing workflows at the MFP was possible in the first version of jAWS. The user interface developed for the printer gave the user control over all designed processes saved at the printer. There was an overview of known (i.e. locally saved) workflows that could be edited locally. It was possible to rearrange the sequence of tools inside a workflow and edit all parameters of every tool.



Converts an ASCII text file to postscript.
Orientation is "landscape"

Figure 1: The list of tools on the MFP.

To create a workflow, the user would have to choose the tools to add from the set of all available tools on the MFP. Figure 1 shows this view with the *Print*-Tool selected. A brief description of the tool was displayed on the right. After adding the tool to the workflow, all parameters had to be set to sensible values (Fig. 2).



INFILE file
OUTFILE print.ps
HEADLINE jAWS

Figure 2: Parameters of the *Print*-Tool.

A workflow was displayed as a sequence of tools (Fig. 3). This sequence could be altered by adding, moving, removing and setting parameters of tools. Every parameter was visible, so altering the workflow required profound knowledge of the inner workings of the system in general and especially the tool and workflow itself. This is necessary when the workflows are designed, but it was not possible to design a workflow that allowed the user to just edit a few selected parameters like e.g. the number of copies without exposing all of the internal structure.

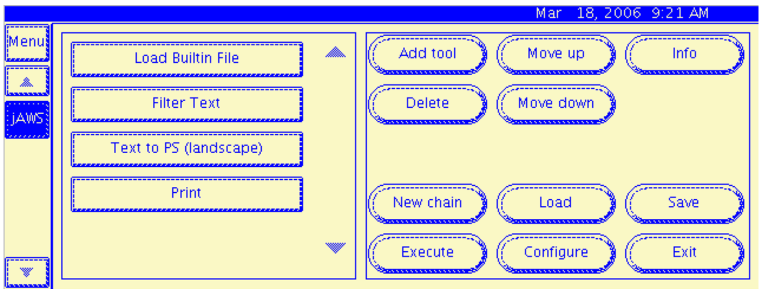


Figure 3: A workflow.

The jABC plugin allows users to easily design even complex workflows. By providing an intuitive user interface and plugin API, the jABC is very well suited as a GUI frontend for the jAWS engine. With the developing of the jABC plugin even more features were supported. The user interface on the MFP itself was altered to reflect the change in the editing paradigm. Workflows could now be be added to jAWS by downloading process descriptions. These process descriptions are generated from a graph designed in the jABC. This graph may contain SIBs resembling local tools or predefined MetaTools in jAWS, and ETI SIBs that are executed on a remote server (see figure 4).

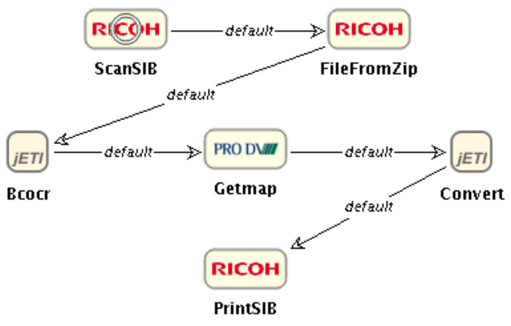
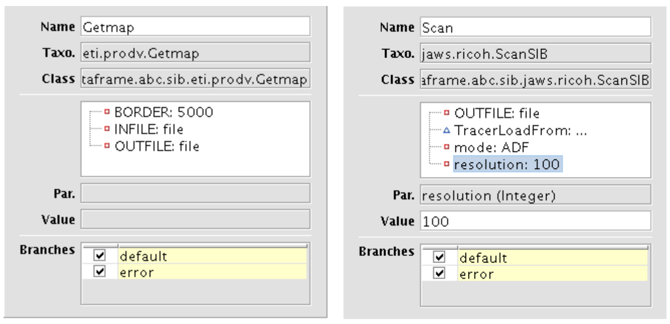


Figure 4: The example workflow: Read a postal code and get the corresponding map.

The SIBs provide the same parameters that are configurable on the printer. The decision which parameters a jAWS SIB provides has to be made at compile time. At the moment there is no way to generate SIBs compatible with jAWS out of a tool description for the jAWS engine. So SIBs representing local tools have to be created manually, but the jETI SIBs can be generated out of the deployment descriptor of a jETI server (see fig. 5a).

In jABC, it is possible to mark parameters of SIBs as “model parameters”. These parameters cannot be set inside of the model they are defined in. If the graph is nested in another graph, only these model parameters are visible. jAWS uses these model parameters as “higher level” parameters, either editable by the user or set by the embedding process. Other parameters are hidden on the MFP.

The jETI plugin for jABC provides a delegation of execution of the ETI SIBs. The SIBs are just a representation of the function call at the jETI server. When the graph is executed inside jABC, these SIBs are not executed locally but the service they represent is used. The jAWS plugin does not work like that because of the necessary “real life”-interaction before the execution of workflows at the printer (e.g. inserting original documents into the scanner). It generates a description of the graph, compiling it into a workflow that can be downloaded at the printer.



Name: Getmap
Taxo.: eti.prodv.Getmap
Class: taframe.abc.sib.eti.prodv.Getmap
- BORDER: 5000
- INFILE: file
- OUTFILE: file

Par.:
Value:
Branches: ☑ default ☑ error

Name: Scan
Taxo.: jaws.ricoh.ScanSIB
Class: aframe.abc.sib.jaws.ricoh.ScanSIB
- OUTFILE: file
- TracerLoadFrom: ...
- mode: ADF
- resolution: 100

Par.: resolution (Integer)
Value: 100
Branches: ☑ default ☑ error

(a) A generated ETI SIB (b) The jAWS ScanSIB

Figure 5: Parameters of SIBs

An overview of the Architecture of the different systems working together is provided by figure 6. The jAWS engine on the MFP is the main controller for the processes, executing them step by step, invoking ETI services when they are needed. ETI servers allow asynchronous remote execution of tasks too heavyweight for the MFP. They process input data sent to them by jAWS, and provide the results for further processing. The jABC acts as a design frontend with prototyping capabilities, providing the process descriptions for the jAWS engine. These descriptions may also be executed inside the jABC with it acting as a ETI client.

4.2 Components

4.2.1 ETI SIBs

The properties of the Getmap-ETI-SIB are displayed in figure 5a. It is generated from the service description of a jETI-server. This description contains all tools provided

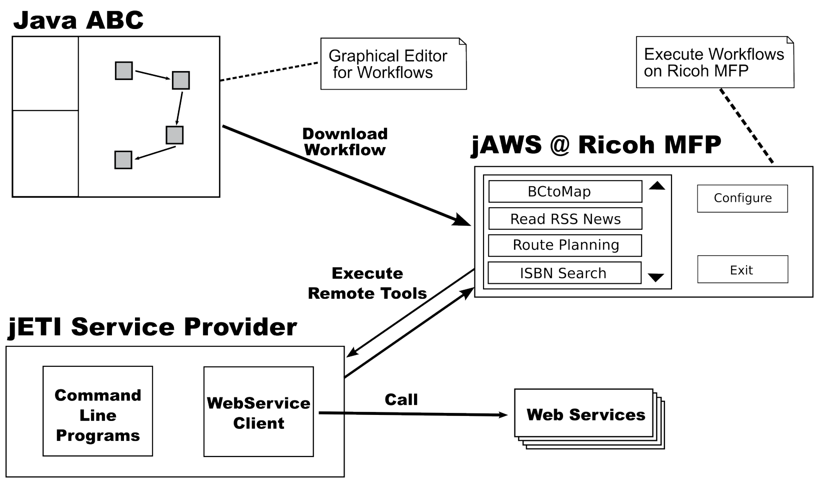


Figure 6: Cooperation between jABC, jAWS and jETI.

by the server with the parameters they take, input and output types. The three parameters (BORDER, INFILE, OUTFILE) are defined in this description and are also used by jAWS with the same names. The execution of these SIBs is delegated to the jETI server in the jABC as well as in jAWS.

4.2.2 jAWS SIBs

The jAWS SIBs are designed to simulate behavior on the printer when executed within jABC. For example, scanning on the MFP creates a zip file with the scanned pictures. If you start the example workflow (see figure 4) in jABC, you have to provide a zip file with pictures in it to simulate the scanning process (the properties of the ScanSIB for jAWS are shown in figure 5b). The parameter “TracerLoadFrom” is not available on the MFP, because it holds the location of the zip file for local execution.

These SIBs cannot be generated and have to be handcrafted, because the simulation of the printer tools on a normal desktop computer is not generic.

4.2.3 Workflows

Figure 4 is an example for a workflow designed with jABC. It analyzes a business card and prints a map of the surroundings of the scanned address. This scenario was shown at the CeBIT 2006, people interested in the demonstration would have their business card scanned and searched for a postal code. The scale of the printed map would be chosen at the MFP, and a third-party GIS service was used to get the map segment. One result of this workflow is shown in figure 7.

If it is executed by the Tracer-plugin from within the jABC, the jAWS-SIBs (*Scan*, *FileFromZip* and *Print*) act as local helpers, providing a behavior that simulates the functions on the MFP. The example workflow goes through the following steps:

- **Scan**. The Scan-tool provides a zip file with scanned images on the MFP. For an execution inside the jABC, a zip file has to be provided.
- **FileFromZip**. The SIB does the same things locally as the tool in the jAWS engine: Extracting one file from a zip archive.
- **Bcocr**. This SIB references an ETI service. It sends the unpacked file to an ETI server and uses it as an input to the service called "bcocr". This service uses OCR to retrieve a postal code from the scanned image and returns it.
- **Getmap**. This is also an ETI SIB. With the "getmap"-service, a map belonging to the obtained zip code is returned.
- **Convert**. This service embeds an image file, in this case the map returned from "getmap", into a postscript file ready for printing.
- **Print**. The generated postscript is printed using the facilities on the MFP or the default printer provided by the operating system.

The jAWS tracer engine executes the same steps as described above for a process generated from the workflow. The ETI tools will be executed on the configured server, and local SIBs will be translated into local tools in jAWS and executed using the capabilities provided by the MFP.

5 Implementation

In writing the code for the MFP there were two main challenges: First, the limited resources that could be used in terms of memory and CPU, and the limited user interface.

5.1 Limited Resources

The one thing that is common for virtually all JavaME-enabled products is their limitation regarding (working) memory size and CPU power. Although the MFP is at the upper end of the JavaME-profiles, it is under that restriction too.

These restrictions were not seen at first, because the emulator run on a normal desktop computer and did not enforce any memory or language limitations. So the first, naive tests and implementations were bound to fail at the printer, especially because it only supports the Java language up to version 1.3, while the emulator supported the current language specifications and libraries.

5.1.1 MFP as an ETI server.

The initial goal was to incorporate the MFP as an ETI server. That means it should receive files, process them in one way or the other and send an answer back. This was not feasible not only because of the necessary “physical” interaction (if nothing goes wrong at least you have to get the printout somehow), but also because anything worth doing on a server was either not possible or taking way to long. Additionally, there were only a few third-party libraries that could be used as-is in a JavaME environment. Although the MFP supports the CDC profile, there are classes in the Java standard library that are not available, either because they are not included in the profile, or because they were introduced later than Java 1.3, which still is the version JavaME supports, rendering many of the additional libraries useless.

5.1.2 MFP as an jETI client.

Theoretically, to use an jETI server, you just need the client libraries and instantiate the default connection provided by them. The default protocol used for messaging between client and server is the SOAP XML-dialect. The problem with SOAP was that it encodes binary data in a message as text, giving a substantial blowup in size. In the client library, messages were assembled in memory. This did not work very well for image data, given the MFP has just a few MB of RAM. So, instead of implementing an external memory client-side message handling, a very simple message format was added to the ETI server. The MFP did no longer need to encode the whole message at once, but could easily stream the data. Additionally this got rid of the size blowup because there was no need to encode binary data as text.

As the application evolved, many unnecessary up- and downloads of the same files were encountered. As the underlying communication with the ETI server should not be visible to the user, the workflow designer could not influence this behavior, so a simple data flow analysis for these workflows was written. This analysis marked the files used by the single tools to be transferred from or to the server, if necessary.

5.2 User Interface

The user interface of the printer is a small rectangular touch screen. There were no automatic layout managers for the AWT-based graphics library available, so the first thing to do was to produce layout manager needed for the display of workflows, like scrollable lists, a split layout and so on.

The problem with this screen was the impreciseness of the “click detection” and the rather slow reaction of the system. The first version of jAWS relied on the GUI of the printer to create workflows. This was feasible for small examples, but when the scenarios grew more complex, it became more and more obvious that maintaining workflows directly at the printer was a very error-prone and tedious task, not only because of the GUI design with no possibility to view all needed informations at once in the workflow context, but also because of the sloppy hardware. This led to a redesigned user interface relying on the jABC plugin to design workflows and offer to download them to the printer.

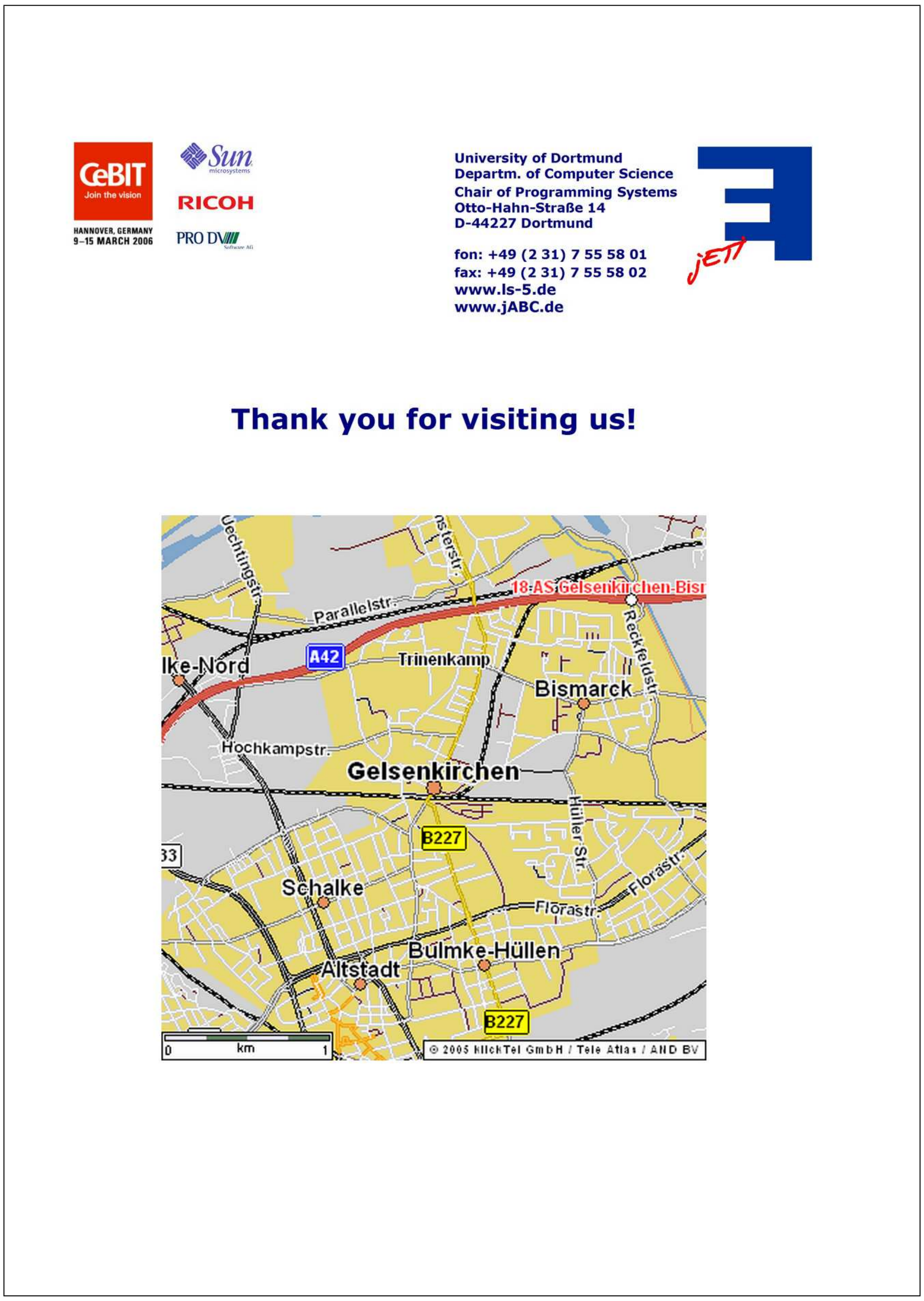
6 Future Work

The jABC Tracer plugin supports alternative named outgoing edges to follow after execution of code, but at the moment jAWS is limited to linear sequences. It supports only an IPO "input file – process – output file" execution flow.

Another feature of jABC are GraphSIBs, building blocks that contain other workflows. At the moment, these GraphSIBs are not supported by the jAWS plugin, although nesting workflows can be used if the workflow description is created manually.

There should be a possibility to generate the jAWS SIBs out of the actual tool classes that run at the MFP, like the generation of ETI SIBs out of the service description.

After supporting more than one execution path there should be tools that interact with the user (e.g. for confirming results sent by an jETI server) to get beyond batch processing.



University of Dortmund
Departm. of Computer Science
Chair of Programming Systems
Otto-Hahn-Straße 14
D-44227 Dortmund

fon: +49 (2 31) 7 55 58 01
fax: +49 (2 31) 7 55 58 02
www.ls-5.de
www.jABC.de

Thank you for visiting us!

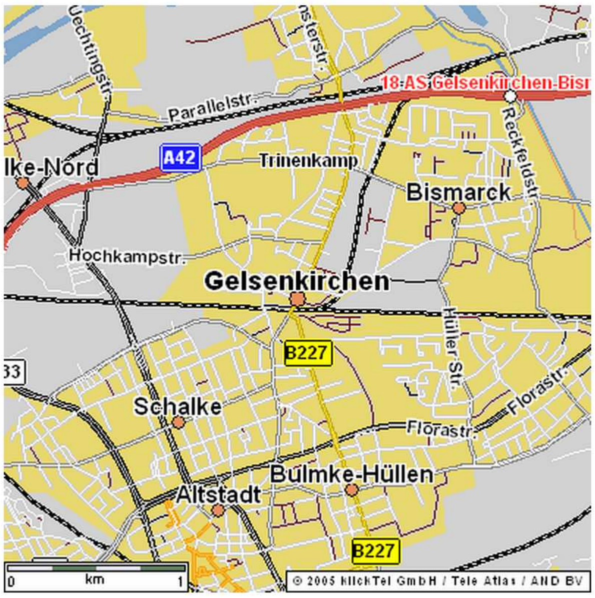


Figure 7: The result of the workflow shown at the CeBIT 2006 in Hannover.

References

[1] BERNHARD STEFFEN, TIZIANA MARGARIA and VOLKER BRAUN: *The Electronic Tool Integration Platform: Concepts and Design.* STTT, 1(1-2):9–30, 1997.

[2] IBM: *Rational Unified Process.* `http://www-306.ibm.com/software/awdtools/rup/`.

[3] MARGARIA, TIZIANA: *Electronic Tool Integration.* `http://jeti.cs.uni-dortmund.de`.

[4] MARGARIA, TIZIANA, RALF NAGEL and BERNHARD STEFFEN: *jETI: A Tool for Remote Tool Integration.* in *TACAS*, pages 557–562, 2005.

[5] MARGARIA, TIZIANA and BERNHARD STEFFEN: *Lightweight coarse-grained coordination: a scalable system-level approach.* STTT, 5(2-3):107–123, 2004.

[6] NAGEL, RALF: *Java Application Building Center.* `http://www.jabc.de`.

[7] SUN MICROSYSTEMS: *Connected Device Configuration (CDC); JSR 36, JSR 218.* `http://java.sun.com/products/cdc/`.

[8] SUN MICROSYSTEMS: *Foundation Profile.* `http://java.sun.com/products/foundation/`.

[9] SUN MICROSYSTEMS: *Java Mobile Edition.* `http://java.sun.com/javame/overview.html`.

[10] W3C: *SOAP Version 1.2 Part 1: Messaging Framework.* `http://www.w3.org/TR/2003/REC-soap12-part1-20030624/`, 2003.