Reusable Software Components for Performability Tools, and Their Utilization for Web-Based Configurable Tools*

Aad P. A. van Moorsel[1] and Yiqing Huang[2]

[1] Distributed Software Research Department
Bell Laboratories Research, Lucent Technologies
600 Mountain Ave., Murray Hill, NJ 07974, USA
aad@bell-labs.com
[2] Center for Reliable and High-Performance Computing
Coordinated Science Laboratory, University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL 61801
yhuang@crhc.uiuc.edu

Abstract. This paper discusses software reusability strategies for performance and reliability modeling tools. Special emphasis is on web-embedded tools, and the potential interaction between such tools. We present the system analysis tools (SAT) application programming interface, which allows for quickly embedding existing tools in the web, and generally simplifies programming analysis tools by structured reuse. We also introduce the FREUD project, which has as primary aim to establish a single point of access to a variety of web-enabled tools. In addition, FREUD facilitates configurable web tools by allowing a user to select from the registered modeling formalisms, solvers and graphics tools, and by providing glue between the tools through scripting. We will argue that this form of reuse is particularly suitable for performability modeling tools because of their predictable usage pattern.

1 Introduction

As witnessed by the contributions to the series of workshops on performance tools [23], the last decade has seen a proliferation of software support for model-based analysis of system reliability, performance and performability. Most software tools, however, support only the modeling formalism or solution methods under investigation, and do not provide a software platform that support future development of similar tools (possibly by other developers). As a consequence, practice shows that similar but incompatible software is being developed at various places, with only limited cross-fertilization through reuse.

The current status of software design and development methodology, and its deployment in the field, seems to create opportunities for some major changes

* This work was initiated while Yiqing Huang was with the Distributed Software Research Department, Bell Labs Research, Murray Hill, Summer 1997.

in this situation. First of all, readily available software packages like Tcl/Tk or Java's abstract windowing toolkit, already significantly simplify the programming of user interfaces. In addition, methodology underlying software development and reuse becomes more and more widely accepted [28], like that of components [17, 31], coordination [11, 22], frameworks [8] and patterns [10]. Based on these developments, we discuss in this paper domain-specific opportunities for reuse in the area of performance and reliability modeling tools.

Recently, there have been the first developments in the direction of providing programming interfaces for performability tools. The generic net editor system AGNES, developed at the TU Berlin is the first major effort we know of [20, 32]. It provides the necessary C++ classes to create front-ends for tools in the style of TimeNet [12], UltraSAN [30], DyQNtool [15], GISharpe [26], and the like. In the SAT application programming interface (earlier introduced in a hand-out [18]), we provide a subset of the functionality of AGNES, in Java. The SAT API also includes reusable software for creating web-enabled tools, and we will discuss this effort in more detail in this paper. Another recent development is the Mobius Java-based framework developed at the University of Illinois [29]. It takes things one step further than AGNES and SAT in that it also provides reusable software for algorithms, such as state-space generation algorithms.

Certainly other tool developers have considered forms of reuse, but the above are the ones we know off where software is designed with as a primary purpose to assist (other) tool developers with future projects. Of particular interest is the development of generic languages that can be used for specifying a variety of model formalisms, or can be used to interface with a variety of solvers. Examples are the FIGARO workbench [2] developed at Electricité de France, or the SMART language developed at the College of William and Mary [5]. However, languages are only potential (but very important and far reaching) *enablers* for software reuse, they do not deal with the software architecture itself. Similarly, the very successful efforts in the scientific computing community to provide standard implementations and templates for numerical algorithms are reuse enablers, and must be complemented by software solutions [3].

In this paper we discuss two complementary approaches to reuse we followed in providing performability tools over the web. The presented techniques use known reuse methods, tailored specifically to performability modeling tools. First, we introduce the system analysis tools API, which is constructed in the form of a framework [8]. It contains reusable Java classes, and aims at reuse at the source code level. Using the SAT API existing tools can be made web-enabled quickly, and new tools (web-based or not) can be developed very efficiently. Secondly, we introduce the FREUD tool configuration architecture, which allows users to configure a tool from components registered with the FREUD web site (components are for instance graphical modeling interfaces, core mathematical descriptors, solution engines, or display facilities). Each of the components has mature stand-alone functionality, but a combination of components is needed to solve a user's problem. Reuse of components is run-time, using scripting to glue components together [25]. The web-based FREUD implementation assumes that

all components are web-embedded, and uses JavaScript as glueing language, but similar approaches can be followed outside the web context.

Before discussing in detail the SAT API and FREUD, we first discuss characteristics common to performability tools. This will help identify what tool modules to consider when configuring tools.

2 Performability Tools Components

We identify typical building blocks for performability tools, and discuss requirements for these tools. We base the search of common structures of performability

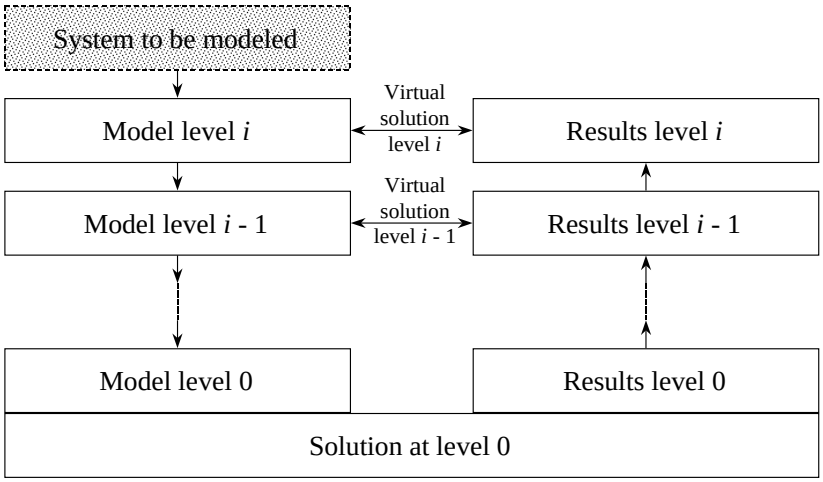

Fig. 1. General modeling tool framework, functional view.

tools on the general modeling tool framework, given in Figure 1. Haverkort [14] formulated the GMTF to classify and compare performability tools with regard to their user-perceived functionality (note that Figure 1 is some what less detailed than the GMTF presented in [14]). In the GMTF a system model is formulated at level i, then transformed into a level $i-1$ formalism, etc. At level 0 the actual solution takes place, after which the results are translated back to the original model level (level i) formalism, so that they can be directly interpreted by the user. As an example, a stochastic Petri-net tool first creates a Petri net description (level 1), then maps it on a Markov chain description (level 0), then solves the Markov chain and gives results in terms of probability distribution of markings in the Petri net.

The above framework is based on a functional view of tools, from a user perspective, but we are interested in discussing tools from an implementation (or software design) perspective. Therefore, we transform Haverkort's 'functional view' GMTF into a 'software view' GMTF, as depicted in Figure 2. In Figure 2

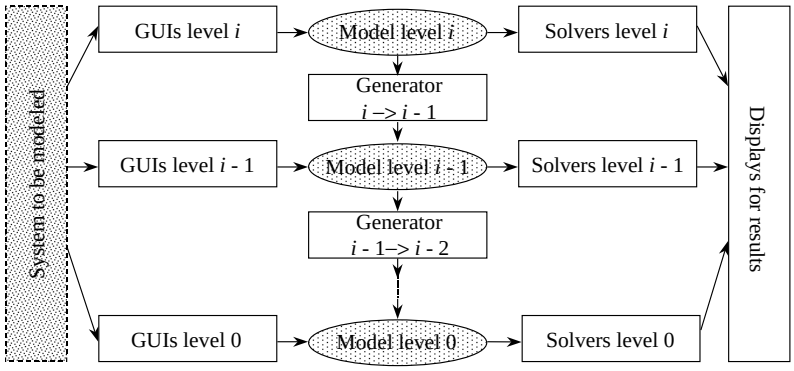

Fig. 2. General modeling tool framework, software view.

the (transparent) boxes now correspond to software modules or tool components, while the ovals indicate a modeling formalism (at some level). So, the ovals do not indicate software components, but indicate that the components surrounding an oval operate on a corresponding formalism.

There are several differences between the functional and software GMTF. First, it is observed that a 'generator' module is required to map a level i formalism to a level $i-1$ formalism. For instance, a state space generator to create a Markov chain from a Petri net specification. Secondly, we choose not to distinguish different levels of results. Although this is important from a functional perspective [14], from a software perspective the display of results will typically consist of the same component, independent of the level. Thirdly, more subtle, the functional view GMTF in Figure 1 must be filled in on a 'per tool' basis, that is, the levels will have a different meaning depending on what tool is fit to the GMTF. In the software-view GMTF, we want to fit different tools into one picture, so that different GUIs, different model formalisms, different solvers can be related to each other. Thus, multiple GUIs are possible in the modified GMTF, as are multiple solvers. Note also that solvers on different levels can be applied (for instance on-the-fly solutions [6] or matrix-geometric solutions [16]).

The software-view GMTF of Figure 2 is not claimed to be the only or preferred way of relating performability tool software components. It is set up so that it motivates the component approach we take in the FREUD configurable tools facility (see Section 4). We have for instance not included libraries, documentation and debugging software components in the GMTF, since it is beyond the scope of this paper to go in too much detail. The point we want to make is that performability modeling tools naturally lend themselves for a component-based software design. All components have clearly defined functionality, and are able to execute largely independently their respective tasks. Moreover, because they operate independently, different implementations of each component may be substituted for each other, without impairing the functioning of any of the other tools. As a consequence, special-purpose GUIs and solvers can be plugged in easily, provided the software is designed to allow for such enhancements.

3 An Application Programming Interface for Web-Embedded Tools

In FREUD, and the configurable tool facility of FREUD, it is assumed that all software components in the GMTF[1] can be accessed and used over the web. We therefore discuss in this section how to make these tool components web-enabled.[2] We introduce the system analysis tools API, which provides reusable Java code for making tools available over the web.

3.1 Web-Enabled Software

Web-embedded applications are being developed in various fields [3, 13], and performance modeling tools have been shipped to the web as well [18, 26]. The advantages of using tools over the web are plenty, both for the user and tool developer [18]. The major advantages come from the fact that only one version of the tool (put on a web site) is public, and can run in any suitable web browser. In addition, computing facilities on client as well as server side can be used. As a consequence, overhead caused by installation, updating and distributing, and performance degradation caused by inadequate user equipment do no longer play a role. If issues of reliability, security and network performance can be kept in control, and if the process of making tools web-enabled is simple enough, then the web is in many cases the preferred platform. In this paper we make no attempts to tackle quality of service issues, but we do attempt to simplify the development of web-embedded tools.

In its simplest form, a web-based tool is a stand-alone application running fully in the client's web browser; this can be in the form of an applet or plug-in (applets run Java code, plug-ins for instance Tcl/Tk programs, or any other code for which a C++ implemented plug-in is provided). If connection with a server is needed, the HTTP protocol provides the possibility to execute CGI scripts, but this is often not sufficient. Instead, communication will be based on sockets or other higher level protocols, like CORBA's IIOP or Java's RMI.

3.2 SAT API

The system analysis tools application programming interface (SAT API) provides all the necessary code to make tools web-enabled. The SAT API is a framework of Java classes which take care of all aspects of communication, from the applet to the server-side executables. In addition, the SAT API provides performability-tools specific Java classes for user interface creation. The SAT

[1] Unless otherwise noted, in what follows GMTF denotes the software-view GMTF of Figure 2.

[2] We interchangeably use 'web-embedded,' 'web-based,' 'web-enabled,' to indicate tools that can be accessed and used over the world-wide web. That is, a front-end is started up in the user's browser, and, if necessary, a back-end server is available (typically for heavy-duty computation).

API is designed such that GUIs, menu options, and communication classes interface through abstract classes, not through their implementations. Hence, implementations of user interface, menu options and communication methods can be changed independently. We introduced an earlier version of the SAT API in a hand-out [18].

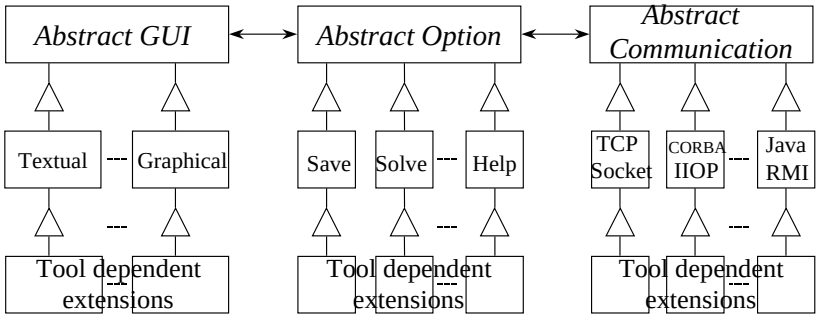

Fig. 3. Client side class hierarchy of System Analysis Tools API.

The SAT API contains classes for the client side and the server side of a web-embedded tool, and we first discuss how the client side is designed, using Figure 3. The main classes are the abstract classes 'GUI', 'Option' and 'Communication,' which interface with each other. Instantiatable subclasses implement (at least) the methods in the abstract classes; if necessary, tool-specific subclasses can be created to override and extend the implemented generic classes. Figure 3 depicts this set-up, using a triangle to indicate a class-subclass relation (as in [10]). So, for instance, there are GUI classes for textual as well as graphical interfaces, and communication classes for socket communication, Java RMI, etc. With this set-up, classes can easily be extended for specific tools, and can easily be replaced by specific implementations or by different technology (the communication can be TCP socket communication, Java RMI, etc., without impacting the GUI and menu options).

The GUI and Option classes are respectively the "view" classes (following the model-view-controller paradigm [7]), corresponding to the visual aspects of the GUI, and the "control" classes, corresponding to the options in the menu bar ('save', 'load', 'help', etc.). There has been made no advanced attempts to create extensive graphical user interfaces (Figure 7 shows a GUI based on the SAT API); the 'textual' classes provide xterm-like functionality to take over the role of the terminal window for textual tools. Of course, there is a need for reusable Java GUI classes for modeling tools, the same as AGNES provides in C++.

On the server side, the SAT API provides a server daemon that continuously listens for users that want to access the back-end, and there is a wrapper provided in which executables are being called. Figure 4 depicts the client and server

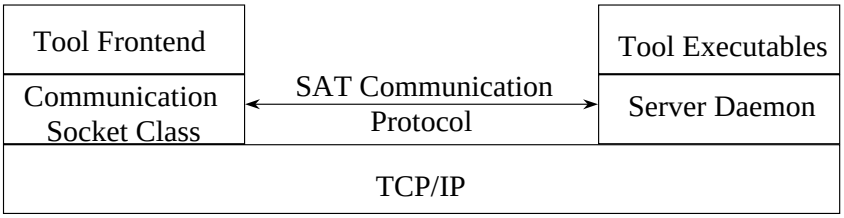

Fig. 4. Client-server functionality in System Analysis Tools API.

modules for the case that a specially developed communication protocol is used. In Figure 4 the 'Tool Frontend' includes the GUI and Option classes from Figure 3, while the 'Communication Socket Class' denotes a particular Communication class. Together they constitute the applet appearing in the user's web browser.

For each individual tool, the server daemon must be configured by setting some parameters, while the wrapper must be adjusted to execute the correct programs depending on the user input. This requires minor effort; we have implementations for SHARPE [27], AMPL [9] and TALISMAN [24], and they only differ through a few lines. The SAT communication protocol mentioned in Figure 4, specifies the message format for communication between front-end and back-end. If this protocol is being used, the construction of messages to be sent, and the parsing of receiving messages can be provided as a reusable class. Note that this communication protocol is based on message passing; if techniques based on remote method invocations (RMI, CORBA) are being used one will not use the SAT communication protocol.

4 FREUD

In this section we discuss the FREUD architecture and implementation. FREUD provides users with a single point of access to web-embedded system analysis tools of the type described in the previous section, and allows users to configure their own software support out of components registered at the FREUD site. We first discuss the basics of FREUD, without the configuration aspects. Then we discuss how FREUD supports configurable tools.

4.1 Registration Service

The provision of a "one-stop-shopping" facility is the original motivation behind FREUD, see Figure 5. Imagine people are interested in analyzing their system for performance or reliability, and want to find the tool that is most suitable. Then it would be attractive if there is a central, well-publicized, site with links to usable tools. This central site, or gateway, does not need to offer much other functionality, since that is left to the individual tools. Effectively, FREUD is thus a regular web site with a database for registered tools (For a more sophisticated view point, we refer to the discussion in [19], where it is argued that authentication, fault tolerance and load balancing may be done centrally. In this paper

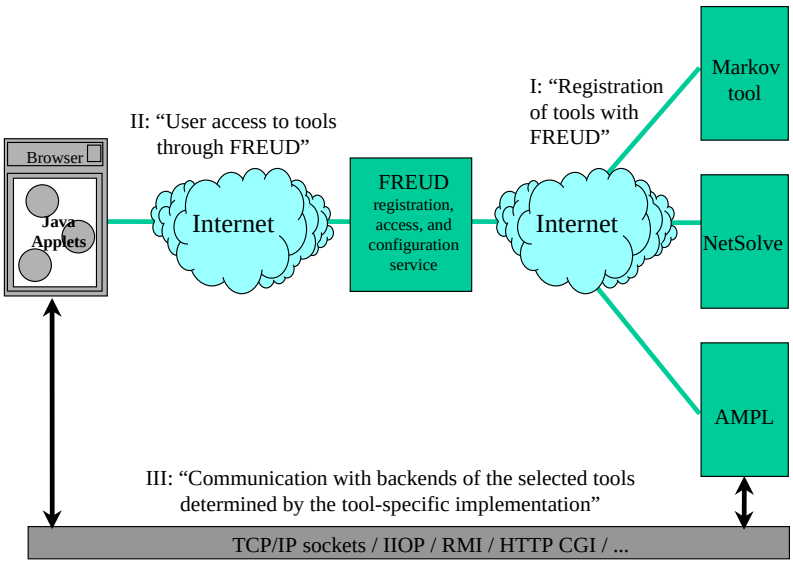

Fig. 5. FREUD.

we do not concern ourself with these issues.) In Figure 5, the FREUD site is in the center, and various tools have registered with FREUD (step I); these tools can be accessed by the user from a page with appropriate links dynamically created by FREUD (step II), after which the tool will be downloaded, and future communication is established in any desired way (step III).

Space limitations prohibit us from showing all web pages a user of FREUD may download, but the pages include user authentication displays, tool listings, tool registration forms, etc.

4.2 User View

When the user who accesses FREUD decides to build up a 'new' tool out of registered tool components, the applet in Figure 6 pops up. This configuration applet contains three panels. The left-most panel shows the tool components registered (the shown list contains all tools we have currently available over the web). The user selects desired components in this panel; for instance, a GUI to specify a model, a solution engine to solve the model, and a data display tool to show results. The selected components will then be displayed in the second panel. In Figure 6 Markovtool (a locally developed tool for drawing a hybrid Markov model/Petri net) and NetSolve (a solution engine from the University of Tennessee [3]) are being chosen.

The user then needs to specify how the tool components should interact; that is, which component's output should be forwarded to which component's input. In the example, Markovtool output (a model) will be forwarded as Net-

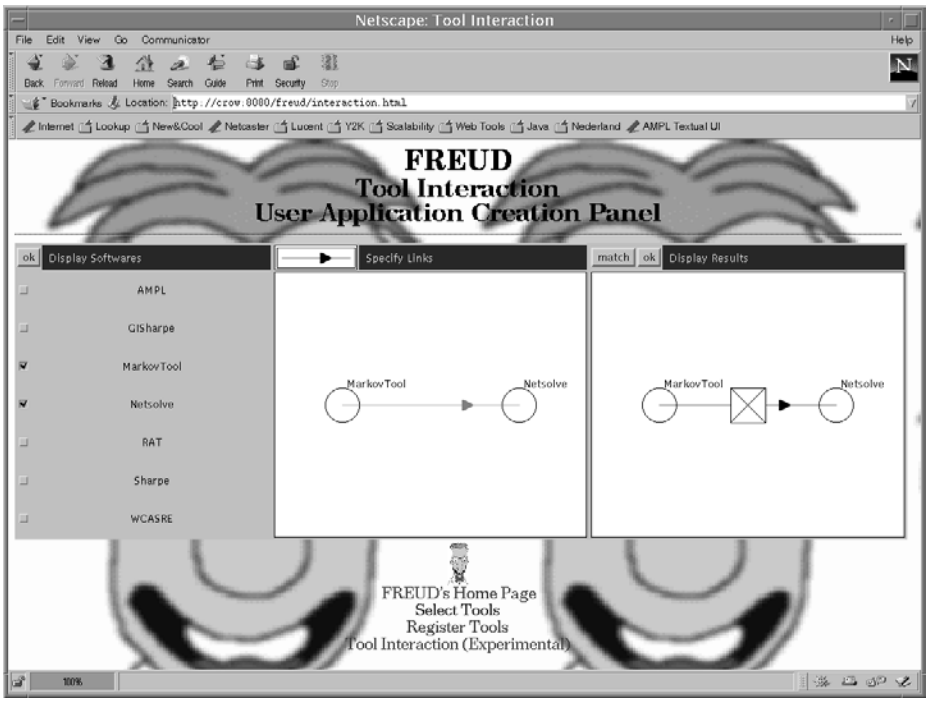

Fig. 6. Configuration facility FREUD.

Solve input (for solution). Since the formats of Markovtool and NetSolve do not match, a translator must exist to convert. The user therefore clicks 'match' to see whether a translator between Markovtool and NetSolve is available. The FREUD gateway will search its database; if the translator exists, it shows up as a marked square between the selected tools. In Figure 6 it can be seen that a translator from Markovtool to NetSolve is available. Finally, the FREUD gateway then dynamically creates an HTML document with the desired collection of tool components and translators (after the user clicks 'OK'). The page in Figure 7 appears (Section 4.3 explains how the page is constructed).

To put this example in context, we illustrate how Markovtool, NetSolve, and the translator fit the GMTF. Markovtool is a level i modeling formalism (for some i; if no further tools are considered one can take $i = 1$), with as output a description of the Markov model, and an initial distribution. The translator then generates a level $i-1$ model in terms of a system of linear equations, which NetSolve then solves. The translator thus functions as a 'generator' component in this example, while NetSolve is a solver at level $i-1$.

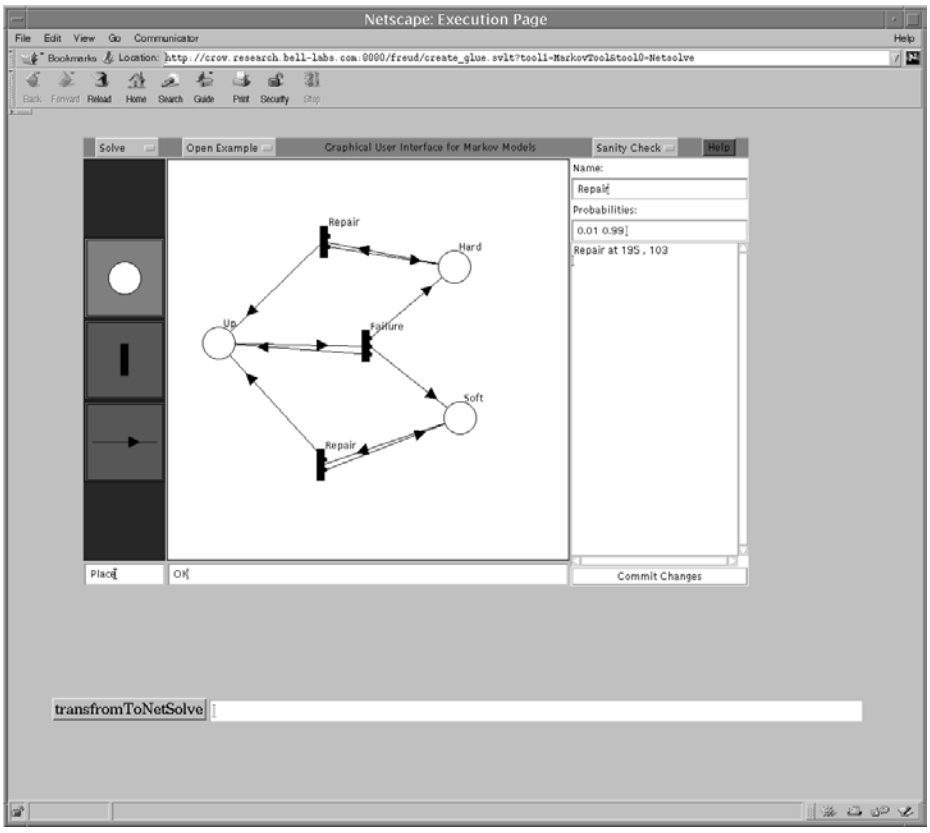

Fig. 7. Configured tool, showing Markovtool and a translator to NetSolve.

4.3 Coordination Mechanisms

In FREUD it is assumed that all registered tools are client-server applications themselves, and there is therefore communication between different components possible on the client side as well as at the server side. We implemented two types of communication, one on the server side based on file I/O, and one on the client side based on scripting using JavaScript. We will concentrate on the client-side coordination through scripting.

The nature of performability tools is such that the pattern of usage of components is typically predictable, as manifested by the GMTF. When we carry out an evaluation, we start with a GUI to create the model, then generate the mathematical model that lends itself for solution, and then display the results. In the GMTF as depicted in Figure 2, this typical user pattern corresponds to a path from left to right. Coordination between components therefore is 'sequential': output of one tool becomes input of the next, possibly after conversion between formalisms and formats. The coordination mechanisms do therefore not

need to be very advanced to lead to useful results; in fact, we need a variation of the 'pipe' construct.

JavaScript is a scripting language for controlling browsers, and can be included in HTML documents. Not all browsers know how to interpret JavaScript, but for instance the Netscape browser does. When configuring tools, JavaScript allows for exchanging information between different applets running in the same HTML document. This is even possible if the two applets come from different machines. JavaScript interfaces with Java applets by calling its methods; for instance, to read the model from Markovtool, JavaScript calls Markovtool's public method `model2String()`.

Using JavaScript it is relatively straightforward to construct a translator that functions as a 'pipe' between applets. The remaining task of the translator then is to convert the format. This can be done in JavaScript as well, or one can choose to create a conversion applet, and it depends on the complexity of the operation what method is preferred. In any case, the JavaScript code, including possibly a reference to an applet, is registered with FREUD, and can from then on be used to glue together tool components.

Below we show the JavaScript code used in the Markovtool-NetSolve translator.

```
function transfMtoN() {
        var aPanel = document.Gui.draw_panel;
        var aString = aPanel.model2String();
        document.HighLevel.display.value = aString;
        document.Markov2Netsolve.transform(aString);
}
```

The JavaScript variable `aPanel` takes as 'value' the instantiated object `draw_panel` in the Gui, which is the tag used in the HTML code to identify the Markovtool applet. Then JavaScript calls the method `model2String()` in Markovtool to obtain a string representation of the model. In the last line the string is input to the applet that is part of the translator.

Note that there is no call to a NetSolve method; instead we used the option of file input and output. The reason for this is symptomatic for the current state of Internet software. The NetSolve Java interface (publicly available) uses a later version of the SUN Java development kit than the Netscape browser accepts. The SUN HotJava browser runs the NetSolve interface, but does not yet know how to interpret Netscape's JavaScript. As a consequence, we have to split up the use of the tools (we run NetSolve in HotJava or appletviewer using URL upload of the translator's output). When Netscape is able to use a newer JDK version (hopefully in the near future), or when HotJava starts supporting JavaScript (supposedly in the near future) we may decide that scripting is preferred over file transfer and change the implementation accordingly. It should also be noted that for the in-house developed tools we have combinations of tools that do coordinate through scripting only.

One important element remains to be discussed, namely the posting of interfaces. To create a translator, one must know the methods available to interface

with the respective tools. Therefore, at registration with FREUD, a tool posts the public methods available, and gives a description of its semantics. This is done using the same registration page submitted when registering a tool. For the Markovtool, the method `model2String` is posted, as well as the way to access it through `draw_panel`. In the current setting the interfaces are posted in natural language, listing the relevant method calls and their functioning. This could be substituted by a more formal way of describing the interface definition.

4.4 Discussion of the FREUD Implementation

FREUD establishes coordination between software components with minimal requirements on these components. We started from the premise that tools should not have to be adjusted before they can fit in the FREUD configuration service. That is a very important decision with far-reaching consequences, since it limits the level of sophistication one can establish in the coordination between tools (a 'for' loop over a global variable, or interactive display as for the transient solvers in TimeNet [12] are some examples of more complex interaction patterns). We think, however, that performability tools are used in relatively predictable ways, and we therefore establish useful interaction by simple means.

If we are willing to impose further rules on the tool implementations registered with the FREUD configuration service, we can use more advanced coordination mechanisms [1, 4, 17, 21]. These mechanisms have properties like event sharing and state persistence, enabling much more intricate forms of coordination. Then GUIs may interoperate, and front ends and back ends can be registered independently (instead of only full-blown client-server tools as in the FREUD architecture). If developers are willing to code within component architectures more advanced forms of cooperation can be achieved, but it is not beforehand clear whether upgrade to complex existing coordination platforms (Java Beans and the like) is required and advisable for our purposes.

5 Conclusion

In this paper we have presented the FREUD architecture, which offers users a single point of access to web-based performability modeling tools. More importantly, it provides mechanisms to let users configure 'new' tools out of registered components. As a consequence, users are able to leverage of existing tools, like GUIs, numerical solution libraries [3] or linear programming executables [18].

We have embedded the discussion of the FREUD configurable tool facility into the larger issue of programming for reuse, since we expect that performance and reliability modeling tools can benefit from this greatly. We therefore also paid considerable attention to the system analysis tools API we developed. This Java API allows legacy tools as well as new tools to be made web-enabled with minor effort. In addition, we proposed a software-view GMTF (as variation of the functional-view general modeling tool framework in [14]) to help structure software and identify requirements for performability tools.

References

[1] R. M. Adler, "Distributed coordination models for client/server computing," *IEEE Computer*, vol. 28, no. 4, pp. 14–22, April 1995.

[2] M. Bouissou, "The FIGARO dependability evaluation workbench in use: Case studies for fault-tolerant computer systems," in *23th Annual international Symposium on Fault-Tolerant Computing (FTCS-23)*, pp. 680–685, Toulouse, France, June 1993, IEEE, IEEE Computer Society Press.

[3] H. Casanova and J. Dongarra, "NetSolve: A network server for solving computational science problems," *International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, no. 3, pp. 212–223, Fall 1997.

[4] P. Ciancarini, A. Knoche, R. Tolksdorf, and F. Vitali, "PageSpace: An architecture to coordinate distributed applications on the web," in *Fifth International World Wide Web Conference*, Paris, France, May 1996.

[5] G. Ciardo and A. S. Miner, "SMART: Simulation and Markovian analyzer for reliability and timing," in *Tool Descriptions, Supplement to Proceedings 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 41–43, Saint-Malo, France, June 1997.

[6] D. D. Deavours and W. H. Sanders, ""On-the-fly" solution techniques for stochastic Petri nets and extensions," in *7th International Workshop on Petri Nets and Performance Models*, pp. 132–141, Saint Malo, France, June 1997, IEEE, IEEE Computer Society Press.

[7] A. Eliëns, *Principles of Object-Oriented Software Development*, Addison-Wesley, Reading, MA, USA, 1995.

[8] M. E. Fayad and D. C. Schmidt, "Object-oriented application frameworks," *Communications of the ACM*, vol. 40, no. 10, pp. 33–38, October 1997.

[9] R. Fourer, D. M. Gay, and B. W. Kernighan, *AMPL: A Modeling Language for Mathematical Programming*, Duxbury Press, Belmont, CA, 1993.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, USA, 1995.

[11] D. Gelernter and N. Carriero, "Coordination languages and their significance," *Communications of the ACM*, vol. 35, no. 2, pp. 97–107, February 1992.

[12] R. German, C. Kelling, A. Zimmermann, and G. Hommel, "TimeNET–a toolkit for evaluating non-Markovian stochastic Petri nets," *Performance Evaluation*, vol. 24, pp. 69–87, 1995.

[13] O. Günther, R. Müller, P. Schmidt, H. K. Bhargava, and R. Krishnan, "MMM: A web-based system for sharing statistical computing modules," *IEEE Internet Computing*, vol. 1, no. 3, pp. 59–68, May-June 1997.

[14] B. R. Haverkort, *Performability Modelling Tools, Evaluation Techniques, and Applications*, PhD thesis, University of Twente, The Netherlands, 1990.

[15] B. R. Haverkort, "Performability evaluation of fault-tolerant computer systems using DyQN-tool$^+$," *International Journal of Reliability, Quality and Safety Engineering*, vol. 2, no. 4, pp. 383–404, 1995.

[16] B. R. Haverkort and A. Ost, "Steady-state analysis of infinite stochastic Petri nets: Comparing the spectral expansion and the matrix-geometric method," in *Seventh International Workshop on Petri Nets and Performance Models*, Saint Malo, France, June 1997, IEEE Computer Society Press.

[17] D. Kiely, "Are components the future of software," *IEEE Computer*, vol. 31, no. 2, pp. 10–11, February 1998.

[18] R. Klemm, S. Rangarajan, N. Singh, and A. P. A. van Moorsel, "A suite of internet-accessible analysis tools," in *Tool Descriptions, Supplement to Proceedings 9th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pp. 34–36, Saint-Malo, France, June 1997.

[19] R. Klemm and A. P. A. van Moorsel, "Offering computing services on the world wide web," Submitted for publication, February 1998.

[20] K. Koischwitz, *Entwurf und Implementierung einer parametrisierbaren Benutzungsoberfläche für hierarchische Netzmodelle (agnes–ein Generische Netz-Editor-System)*, Master's thesis, Technische Universität Berlin, Berlin, Germany, October 1996. In German.

[21] D. Krieger and R. M. Adler, "The emergence of distributed component platforms," *IEEE Computer*, vol. 31, no. 3, pp. 43–53, March 1998.

[22] T. W. Malone and K. Crowston, "The interdisciplinary study of coordination," *ACM Computing Surveys*, vol. 26, no. 1, pp. 87–119, March 1994.

[23] R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, *Computer Performance Evaluation Modelling Techniques and Tools*, volume 1245 of *Lecture Notes in Computer Science*, Springer Verlag, Berlin, Germany, 1997.

[24] D. Mitra, J. A. Morrison, and K. G. Ramakrishnan, "ATM network design and optimization: A mutlirate loss network framework," *IEEE/ACM Transactions on Networking*, vol. 4, no. 4, pp. 531–543, August 1996.

[25] J. K. Ousterhout, "Scripting: Higher-level programming for the 21st century," *IEEE Computer*, vol. 31, no. 3, pp. 23–30, March 1998.

[26] A. Puliafito, O. Tomarchio, and L. Vita, "Porting SHARPE on the web: Design and implementation of a network computing platform using Java," in *Lecture Notes in Computer Science, Vol. 1245, Computer Performance Evaluation Modelling Techniques and Tools*, R. Marie, B. Plateau, M. Calzarossa, and G. Rubino, editors, pp. 32–43, Springer Verlag, Berlin, Germany, 1997.

[27] R. A. Sahner, K. S. Trivedi, and A. Puliafito, *Performance and Reliability Analysis of Computer Systems, An Example-Based Approach Using the SHARPE Software Package*, Kluwer, Boston, MA, 1996.

[28] J. Sametinger, *Software Engineering with Reusable Components*, Springer Verlag, Berlin, Germany, 1997.

[29] W. H. Sanders and D. D. Deavours, "Initial specification of the Modius modeling tool," Internal report, University of Illinois, Urbana-Champaign, IL, Fall 1997.

[30] W. H. Sanders, W. D. Obal, M. A. Qureshi, and F. K. Widjanarko, "The *UltraSAN* modeling environment," *Performance Evaluation*, vol. 24, no. 1, pp. 89–115, 1995.

[31] O. Sims, *Business Objects: Delivering Cooperative Objects for Client-Server*, IBM McGraw-Hill Series, McGraw-Hill, Berkshire, UK, 1994.

[32] A. Zimmermann, *Modellierung und Bewertung von Fertigungssystemen mit speziellen Petri-Netzen*, Dissertation, Technische Universität Berlin, Berlin, Germany, 1997. In German.