A Java Extension Framework for Web-based Simulation*

Tao Tang and Chu R. Wie+

State University of New York at Buffalo, Dept. of Electrical Engineering, Bonner Hall,

Buffalo, NY 14260

Keywords framework, Java, web-based, simulation, legacy, native

Abstract

We have designed a framework for the web-based simulation applications with a Java front-end. In this paper, we discuss the architectural design and related considerations of this framework. We present details of a multi-layer architecture, key components and their functions, and the abstraction of the common features in the web-based simulation into this framework. This framework is implemented in Java technology with object-oriented design. This framework can incorporate an arbitrary legacy simulation application in various domains by using configurable and extensible interfaces. The usage of our framework can be in the area of developing Java web-based simulation services and simulation problems.

* Supported by National Science Foundation

+ Author to whom correspondence should be addressed, wie@eng.buffalo.edu

1 Introduction

Simulation is an effective tool in learning, research and development. Access to various simulation resources is often limited due to the cost involved in deploying and maintaining the software packages. With the growth of the World Wide Web (WWW) and commercial information distribution systems over the last few years, there has been effort by various groups to extend the simulation resources to the World Wide Web [Kapadia 2000; Chen 2000; Romberg 1999; Haupt 1999]. The resource is then accessible via the web through a uniform user interface, the web browser. A web-based simulation has advantages over native application-based simulations due to its portability and location transparency -- being able to deliver computing and simulation services to end-users anytime anywhere. In addition, it does not need to expose a large amount of resources and control to the end-user while serving sufficient utilities. A well-designed framework is very useful for developing such systems with a considerably less effort, yet yielding a high quality design and performance for the developed product. This framework can possess such attributes as platform independence, extensibility, and ease of use and maintenance. This paper presents the design and implementation of a framework for web-based simulation applications. This framework was developed with an objective to extend the legacy simulation applications to the web using Sun Microsystems' Java technology [Sun Microsystems Inc. 1995]. This framework was applied to implement a web-based microelectronic device simulation package which is named WebSimMicro [Tang 2001; WebSimMicro website 2001]. WebSimMicro project is an online simulation service accessible via a web browser, implemented using the framework which is presented in this paper.

This framework takes into consideration the management issues of remote and local resources for the simulation packages, the request queuing, security models, database and disk storage issues. It supports a multi-layered architecture and an object-oriented composition design, which is important for extension and maintenance. Common tasks of a middle-layer of web-based applications are integrated into this framework. The common tasks include user authentication, session control, request pool, input/output interfaces, engine container management, abstract engine, activity audit and administration suite. Therefore, our framework accommodates many common tasks of the web-based simulations and provides a complete middle-layer support. This framework is readily reusable for extending any native simulation packages to the WWW with little additional design or implementation effort.

Java technology provides the ability to distribute computing services over the Internet in a platform independent manner. The following components are often included in the web-extension: the Java Applets embedded in a web browser as the front-end user interface; the servlets connecting the applet to the server-side Java; JNI (Java Native Interface) connecting the server-side Java to the native simulator; and the Java Database Connectivity (JDBC) for accessing a relational database. Java brings an advantage of a pure object oriented design, extensibility and platform independence. In addition, it helps separate the presentation logic from the computation logic. The user interface is completely replaced by the Java applet presentation. By using Java applets as the graphical user interface (GUI), the users do not need to know the details of usage syntax of the specific simulator. The underlying computational work is done remotely by the application engine residing in a server computer.

This framework and the WebSimMicro project, which is presented in our separate paper [Tang, 2001], provide two kinds of facilities: the tools for building a web-based simulation application, to be used by *software developers*, and the services for performing online microelectronic device simulations, to be used by *the end users of the simulator*. In this paper we focus on the framework architecture and design.

2 Related Work

There are various reported web-simulation projects. These works can be roughly categorized according to their relative emphasis on the framework research (software engineering) or on the implementation effort (development of the web-simulation software product). The frameworks proposed include Purdue University Network-Computing Hubs (PUNCH) [Kapadia 2000; PUNCH website 1998]; Web-based ELectronic Design (WELD) at the University of California Berkeley [WELD website 2001]; ARCADE [Chen 2000; ARCADE technical report 2000]; UNiform Interface to COmputing REsources (UNICORE Plus) [Almond 1999; Romberg 1999; Romberg 2000; UNICORE Plus website 2001]; TheGateway at Syracuse University [Haupt 1999; WebFlow website 2001]; and the Framework for Educational Java Applets [Yuan 2001; Yuan and Wie 2001]. Projects aimed at providing web-based simulation tools include Advanced Web-based Simulation (AWS) [AWS website 2001]; Java based Digital Simulation Automation System (JSAS) [Hur 1999]; The WEB-accessible Petri Net Tool (WebSPN); HAmburg DEsign System on digital circuits (HADES) [HADES website 2000]; and the Java Applet Service on Semiconductor Simulation [Wie 1999; Wie 1998].

Some of these projects (PUNCH, JSAS) provide a basic input extension to the web-based simulation without graphical design assistance, and expose the original input syntax of the wrapped software applications to the end user. Therefore, these products depend on the users' knowledge in the specific simulation package. Some of them present the framework based on a procedural approach, i.e., lack the platform independence [Kapadia 2000]. These characteristics limit their portability. However, the strength of this approach is that the full capability of the simulator is available to the end user. Others (UNICORE, ARCADE, THEGATEWAY, WELD) use the Java technology, and provide a uniform design, but with a specific emphasis on certain implementations. UNICORE's security architecture requires local user ID and the application is accepted only in a batch mode [Romberg 1999]. ARCADE [Chen 2000] and THEGATEWAY [Haupt 1999] targeted on the collaborative features of distributed applications. WELD [WELD website 2001] targets on the data management tools for distributed environment. ARCADE proposed multi-layer architecture and emphasized the collaboration in developing distributed sub-modules for a given application [Chen 2000]. Based on our past experience in the web-based micro-electronic device simulation, we wanted to move the end-user's attention away from the syntax of the specific simulation packages and toward the physical design aspect of the device. In addition, a pure object-oriented and multi-layer framework was preferred for extensibility. Platform independence was favored for portability. The framework must also be easy to use. We designed a framework that meets these requirements.

3 Framework Architecture

The architecture of our proposed framework was designed to meet the static structural requirements of the web-based simulation, the efficiency of application integration, and the

overall performance of the web-based simulation system. By adopting a multi-layer architecture and dividing the distinct functions into different components distributed across various layers, we aimed to loosen the coupling between components, to increase reusability, and to improve development efficiency. Components could be flexibly grouped to meet the performance requirements and to form a thin client and thin server model.

The proposed framework was based on the decomposition of a web-based simulation system into components and tiers. A tier consists of a collection of components. The overall framework architecture is divided into *presentation tier*, *logic tier* and *application tier*.

3.1 Logic view

The overall logic of the system is shown as in Figure 1. Components sit on three major tiers according to their key role within the framework. The coupling stream among components generally follows the flow of a simulation task. Login servlet provides the global access management to the entire system. It also controls the creation of user profiles and the user administration.

Presentation tier: The Input/Output Java applets work as the front end of the system. In the eyes of the end users it is the whole system to deal with while from the view of a simulation application, the input applet plus the logic layer delegate every action of the user to the application engine.

Logic tier: Gateway servlet is the component that the input applet talks to. This servlet is responsible for gathering and pre-processing the user inputs and prepare the simulation

environment. The simulation controller controls the flow of multiple job requests and sets a proper context for each job. It manages the simulation requests on different topics using a scheduling scheme.

Application tier: Engine Container identifies the environment and requirement for each simulation job and will actually call the simulator (or the *engine* which wraps the native simulator). An engine container regulates the flow of execution in a given simulation job.

There are different job management roles for the engine container and for the simulation controller. The engine container handles multiple user requests on the *same* simulation topic from different users. The simulation controller controls the jobs for all topics and users, which then *assigns* each job to an appropriate container based on the job topic, as illustrated in Figure 2. In this way, the engine container actually maintains a *sub-queue* of the jobs that the simulation controller manages.

Let us review a simple example of a web-based microelectronic simulation on bipolar junction transistor (BJT) I-V characteristics. The user logs into her simulation session and views the BJT device structure through a design interface (input applet). Then the user inputs her own design parameters for the device to specify a BJT transistor structure and submit the whole design to the server. Later the user will check the simulation status. Should the simulation job be completed, she would retrieve the information regarding the simulation jobs done previously -- here the Current-Voltage (I-V) data as well as the I-V graph are available for preview. After viewing the output, she may revise the parameters to do the simulation again. If she is satisfied

with the results, she wraps up her work by packing all the useful data and downloads the pack to her local workstation. Then this simulation task is completed.

The sequence for this scenario can be classified as login, simulation and status checking. The logic for login and status checking is common for most web-based applications. The simulation sequence diagram is illustrated in **Figure 3**.

3.2 Component view

Based on the example scenario from the previous section, we further specified the components with operations like:

- User login/logout
- The device design interface
- Simulation job control
- Simulator wrapping
- Simulation status and history
- Simulation data presentation, and
- Utilities for packing the simulation output and downloading

These components spread over the three tiers which we discussed earlier.

3.2.1 Components in the Presentation Tier

Presentation deals with the most basic input/output functionality of the application. Some other components that span over more than one tier, such as login authentication, are also classified into this group. This tier can achieve a high reusability by encapsulating the

presentation requirements and presentation modes that are common to various application engines. The actual implementation of this tier is strongly related to the web components such as Java Server Pages (JSP) or Servlet and JavaBean [Sun Microsystems Inc. website 2000].

- UI Programming Models

The user interface (UI) update can be made the responsibility of either the client or the web server, which corresponds to two programming models: *client-driven UI* or *server-driven UI*. For the client-driven UI model, the client gets data from the server, buffers the data, and changes the user interface according to the logic built into the client side [Wie 1999; Wie 1998]. On the other hand, in the server-driven UI model, whenever the user interface needs an update, the client sends the HTTP request to the web server, then the server generates a new page according to the logic built into the server side, and sends the page back to the client [IBM Inc. 2000]. The client-driven UI model is suitable for small components due to the fact that the logic components have to be either downloaded or pre-installed for each client. When dealing with large UI components, however, the server-driven UI has the advantage of reducing the network traffic.

Our framework supports both models. Specifically, JSP/Servlets implement the server-driven UI model and are mostly used in the simulation management tasks; and the Java applets implement the client-driven UI model and are mostly used in the simulation input/output UI. This makes sense because most management tasks, such as status query or the simulation history display, do not need much intervention from the users. This is suitable for the server to generate and show the UI all at once. On the other hand, the input/output tasks such as the device parameter design and a graphical view of the output may involve a substantial amount of user

interactions and updating of the UI. If implemented in a server-driven UI, there will be a considerable traffic between the client and the server, which will decrease the efficiency. Therefore, by using a client-driven UI, i.e., moving the UI logic to the client side, the overall performance of the web application has been increased in our framework.

- Input and Output Interfaces

The input and output user interfaces are implemented as configurable Java applets. They can also serve as the base classes for more problem-specific input and output requirements. It interacts with the end user for design issues of the simulation problems, and delegates the user for the consequential communication with other components in the framework. It allows the user to modify, submit, reset the design, and retrieve the application output in a uniform manner. Class InputApplet constructs an input interface for the application and provides the default values for the input. Class OutputApplet is responsible for displaying the graphics to the user. It uses most codes in building the graphical user interface and responds to the choose diagram action and refresh request from the user. Both of them also keep the identity of the simulation jobs.

3.2.2 Components in The Simulation Logic Tier

The simulation logic part hides details of the web technology. A brief consideration of the interface between the UI and the web simulation logic is useful here.

Logic part of a web simulation is the unit that is ultimately responsible for coordinating the client requests and the simulator responses. The logic part must address a wide range of potential

requirements which include ensuring transactional integrity of the application components, maintaining and quickly accessing the application data, supporting the coordination of workflow processes, and integrating new application components into the existing structure. To address these requirements and help facilitate the development of logic on the web simulation server, the framework provides the following core functions:

- Database interaction,
- Collaboration between jobs ,
- Application integration for possible extension, and
- Reusable utilities

Java platform provides a wide range of programming and data access services that facilitates the development of the logic part for our web-based simulation framework.

Gateway Servlet (GS): The gateway servlet acts as a glue between the presentation tier UI and the logic tier. It links the middleware (i.e. logic tier) and the front-end UI controls, collects the user inputs, and operates independently for different simulation tasks. It connects to the front-end at the client side and accepts the client requests, collects the design parameters, and delegates the end-user for the interaction with the logic layer components.

Simulation Controller (SC): Simulation jobs may be submitted from multiple users at the same time and many of them may request the same simulator (i.e., the same application engine). Generally, for an application running in a web environment, its resources may need to be shared among multiple jobs at the same time. The SC acts between the gateway servlet and the engine

container, and therefore acts as an execution agent. Here, the engine container contains all the necessary simulation engines for a complete simulation job.

Engine Container (EC): The last major component in the logic layer of the framework is the engine container. The engine container is responsible for holding separate engines and combining them as a whole working unit for a specific simulation topic, as well as dealing with the queuing and the calling of different jobs on a shared simulation engine. In addition, it uses an instance object of DBManager to interact with the database on the job and queue status. To achieve a higher degree of reusability, the container is designed to hold any engine.

Utility Components: Several utility components are developed in our framework. These components represent a collection of standard methods designed for various purposes. They can be designed in a highly reusable way. Many of the utility components used in our framework are implemented with the techniques provided in the Java technology.

3.2.3 Components in the Application Tier

For a real simulation to take place on the web, the core driving part is the underlying simulator which is usually a well-developed program for certain application area. One of the goals for integrating these programs into the a web environment is to make the smallest possible changes to the original package.

Java Native Interface (JNI) [Java Native Interface Specification 1999] allows a Java code, that runs inside a Java Virtual Machine (JVM), to inter-operate with functions that are written in

native programming languages, i.e., the platform-dependent functions. JNI adds flexibility in integrating legacy native applications into Java, but it also adds complexity for the development. By using JNI, the Java code may lose its platform independence, valued most by Java. Caution must be taken when developing JNI applications because it may not be pointer safe any more. For our framework, this is not a problem because for any actual use of the framework, the application engine must reside on certain specific platform. The application tier of the framework is where the local package joins in. One important task of this framework is to provide a generic interface to these native applications. The implementation details mostly depend on the framework user of an individual simulation topic.

Abstract Engine: It is desirable that most or all of the common features of various simulation packages be extracted into a higher level abstraction for a generic simulation package. The abstract engine was created for this purpose. It is an abstract component which cooperates with other components in the framework. Most common features of various application engines are abstracted and used here.

Abstraction included most essential attributes that an engine has to have. For example, an engine does certain computing or logic operations. This feature was abstracted as *do simulation*. An engine also manipulates the user inputs and outputs and does error handling. This feature is abstracted as *do IO*. Another benefit of abstract engine is that some additional useful functions can be added during the implementation. A concrete engine (i.e., a subclass of the AbstractEngine class) will inherit these functions automatically. For example, the ability to

control the total number of instances of a single engine at any given moment can be added as a class variable.

Batch Engine: Our framework has a *BatchEngine* class as the base component for all engines that runs in batch mode. *BatchEngine* is an abstraction of these applications within the general *black-box* model, i.e., take one input and give one output, but hide all internal processes. It is the base wrapper for a non-interrupted application and presents the basic controlling interfaces. The input command scripts are prepared by the application developer. When calling a native application, we only need to specify this script file as the input, which then significantly simplifies the integration of new batch applications into the framework.

Interactive Engine: Our framework has designed a base *InteractiveEngine* class for the purpose of wrapping interactive applications. It is required that such applications must provide a native Application Programming Interface (API), for example the C libraries, for manipulating the command interpreter such as to initialize and close the interpreter, send commands and retrieve responses. The *InteractiveEngine* class provides methods to initiate the engine and talk to the interpreter from time to time by emulating the real user's behavior, for example, giving a command to the application engine. The implementation is based on the API mapping and slightly more complex than that of batch engines. It should contain at least the following operations: initialize or close a command interpreter, send commands, and retrieve responses [Tang 2001].

4 Discussion

Our framework was designed for extending legacy applications to the Web. The application of the framework in developing the web-based simulation can be illustrated as in

Figure **4**. The development efforts are directed to the two end of the architecture: the presentation tier development involving the input applet HTML configuration or InputApplet class extension for advanced UI requirements; the application tier application integration by extending the appropriate engine classes (BatchEngine or InteractiveEngine). The common middleware tasks for the web-based application are encapsulated in the framework. The consideration of middleware function and the ease-of-use are achieved in the framework, compared with the non-object-oriented designs. Our framework and other frameworks presented [Kapadia, 2000; Chen 2000; Hur 1998] can be mutual complementary in the architecture, the user interface models, the security model, and the legacy application integration methods. By adopting three-layer architecture and object-oriented design, we attempted to realize the function modulization and isolation, to increase the design flexibility and scalability, and most importantly, to improve the productivity and efficiency in the web-based application development. Implemented in Java, our framework supports three design steps with clear and consistent tasks: input design, engine configuration, and output design. The presentation layer provides components with data-driven implementations, which separates the simulation-generic and problem-specific requirements therefore lifts the reusability and extensibility. The logic layer contains the simulation controller and the engine container which are responsible for queuing, transaction control, monitoring and task collaboration. They provide implementations for the middle-layer tasks and provide a faster implementation. By abstracting the simulation engines

into two models (*batch engine* and *interactive engine*), the application layer makes it possible to achieve the *engine independence*. In other words, different engines can be treated and integrated in a uniform way. This is an essential benefit when leveraging and extending the legacy simulation programs to the web environment. In addition, the framework provides a collection of utilities with an easy-to-use interface.

The limitation of our framework may result from the abstraction itself. Due to the abstraction, some problem-specific designs were pushed out of the framework. In order to fit into the framework, they sacrifice their problem-specific requirements to a more generic form. For example, the user interface component in our framework, which contains the design image of the simulation problem and parameter panel component, is an agreement between general requirement and the more specific requirement in accommodating legacy applications. This eases the learning curve for an entry level user but may limit the full control over the application for an experienced user. Therefore, for the experienced users who want the maximum use of the application, the traditional simulation method such as *telnet* and *rlogin* is still a preferred way.

The future improvement to our framework is in incorporating distributed simulation models into the framework. The back-end simulation jobs can be fulfilled in parallel, by several servers in collaboration. This will reduce the turn-around time for the simulation jobs. Appropriate communication protocols that fit our framework are Java RMI-IIOP and CORBA. CGI-to-CORBA prototype has been proposed in ARCADE [Chen 2000]. The Java RMI-IIOP may fit better into our Java-based framework. The integration of a *distributed simulation model* will make it possible for the simulation to run across organizations and platforms jointly.

Another improvement can be made by using Extensible Markup Language (XML) to describe the data in the process of generating input files from the templates provided by the framework. XML may become an the industry-wide standard, adopted by an increasing number of institutions [Weiss 1999]. The framework users may use multiple frameworks in order to integrate a variety of class libraries and components in their development. Making use of XML for the data exchange among various frameworks eliminates the problems that may arise from incompatible data definition standards. This shall increase the *interoperability* between different frameworks.

5 Summary

We have designed a framework for Java-based web extension of native applications. Our framework has been designed to facilitate the web extension of native packages. It was intended to save the development time and to improve the quality of design and implementation of web-extension of native programs. By incorporating the Java technology, WWW, and object-oriented design, our framework consists of a set of reusable components and the reusable architectural design. We have implemented a web-based simulation package of microelectronic devices by applying this framework, which is presented in a separate paper [Tang 2001]. It allows a remote access to such powerful microelectronic engines such as FLOODS [Liang 1994] and MINIMOS [Selberherr, 1980]. Finally, a website that provides the web-based simulation in our microelectronic device is available for public access [WebSimMicro website 2001].

6 Acknowledgements

We would like to acknowledge the financial support by National Science Foundations through the grant numbers DUE9752316, DUE9950794 and a partial support from the University of Virginia through a subcontract number 526007.

REFERENCES

ALMOND, J., SNELLING, D. 1999. Uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems (Special Issue on Metacomputing, Volume 15, 5-6, Oct.)*, 539-548.

ARCADE technical report. 2000. http://www.icase.edu/Dienst/UI/2.0/Describe/ncstrl.icase/TR-2000-39.

AWS website. 2001. http://aws.ctc.com.

CHEN, Z., MALY K., MEHROTRA, P. AND ZUBAIR, M. 2000. Arcade: A web-Java based framework for distributed computing. NASA/CR-2000-210545 ICASE Report No. 2000-39 (Oct.), 14.

HADES website. 2001. http://tech-www.informatik.uni-hamburg.de/applets/hades/html/hades.html.

HAUPT, T., AKARSU, E., FOX, G., KALINICHENKO, A., KIM, K. S., SHEETHALNATH, P., YOUN, C. H. 1999. The gateway system: uniform web based access to remote resources. *ACM 1999 Java Grande Conference*. San Francisco, CA, Jun.

HUR, Y., KACPRZAK, D., SZYGENDA, S. A. 1998. Java based digital simulation automation system. *International Conference on Web-Based Modeling & Simulation*.

IBM application framework for e-business. 2001. Web application client programming model. http://www-4.ibm.com/software/ebusiness/clientwp.html.

Java Native Interface Specification. 1999. http://java.sun.com/products/jdk/1.2/docs/guide/jni/spec/jniTOC.doc.html.

KAPADIA, N. H., FORTES, J. A. B., AND LUNDSTROM, M. S. 2000. The Purdue University network-computing hubs: running unmodified simulation tools via the WWW. *ACM Transactions on Modeling and Computer Simulation 10*, 1 (Jan.), 39-57.

KAPADIA, N. H., FIGUEIREDO, R. J., FORTES, J. A. B. 2000. PUNCH: web portal for running tools. *IEEE Micro. 20*, 3. 38-47.

LIANG, M. AND LAW, M. 1994. An object-oriented approach to device simulation - FLOODS. *IEEE Transactions on CAD, 13*, 10. 1235-1240.

PUNCH website. 1998. http://www.ece.purdue.edu/punch.

ROMBERG, M. 1999. The UNICORE architecture: seamless access to distributed resources. *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing* (HPDC-8, Aug.). IEEE Computer Society, Los Alamitos, CA, 287-293.

ROMBERG, M. 2000. UNICORE: Beyond web-based job-submission. *Proceedings of the 42nd Cray User Group Conference* (May 22-26). Noordwijk.

SELBERHERR, S., SCHUTZ, A., AND POTZL, H. W. 1980. MINIMOS - a two-dimensional MOS transistor analyzer. *IEEE Transactions on Electron Devices, 27*, 8 (Aug.). 1540-1550.

SUN MICROSYSTEMS INC. 1995. http://java.sun.com.

Sun Microsystems Inc. product page. 2000, http://java.sun.com/products/jsp, http://java.sun.com/products/servlet, http://java.sun.com/products/javabeans.

TANG T., WIE, C. R. 2001. WebSimMicro: web-based simulation of microelectronic devices. Submitted to *ACM Transactions on Modeling and Computer Simulation*.

TANG, T. 2001. WebSimMicro: framework and implementation of web-based simulation on microelectronic devices. Masters Thesis, State University of New York at Buffalo (May).

UNICORE Plus website. 2001. http://www.fz-juelich.de/zam/RD/coop/unicoreplus.

WebSimMicro website. 2001. http://jas7.eng.buffalo.edu.
WEISS, A. 1999. XML gets down to business. *Networker 3*, 3 (Sep.). 36.
WELD website. 2001. http://www-cad.eecs.berkeley.edu/Respep/Research/weld.
WebFlow website. 2001. http://www.npac.syr.edu/users/haupt/WebFlow.
WIE, C. R. 1999. Educational java applet service. http://jas.eng.buffalo.edu.
WIE, C. R. 1998. Educational java applets in solid state materials (invited paper). *IEEE Trans. on Education 41*, 4 (Nov.). 354.
YUAN, Z. 2001. Design, implementation and application of framework in Java educational applets. State University of New York at Buffalo Masters Thesis.
YUAN, Z, WIE, C. R. 2001. Framework design and implementation for Java educational applet. Submitted to *IEEE Transations on Education.*

Figure Captions

Figure 1. Logic diagram

Figure 2. Job management in simulation controller and engine container

Figure 3. Simulation sequence diagram

Figure 4. Use the framework.

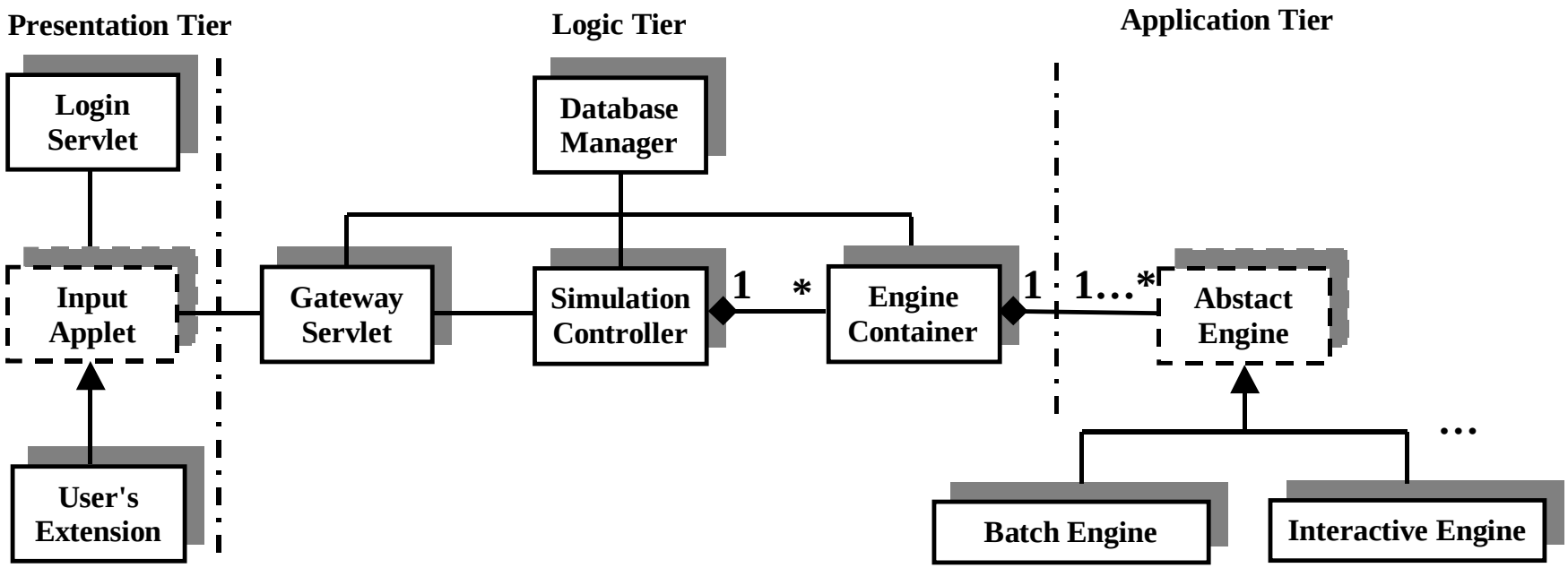
Presentation Tier
Logic Tier
Application Tier
Login Servlet
Database Manager
Input Applet
Gateway Servlet
Simulation Controller
1
*
Engine Container
1
1…*
Abstact Engine
User's Extension
…
Batch Engine
Interactive Engine

Figure 1.

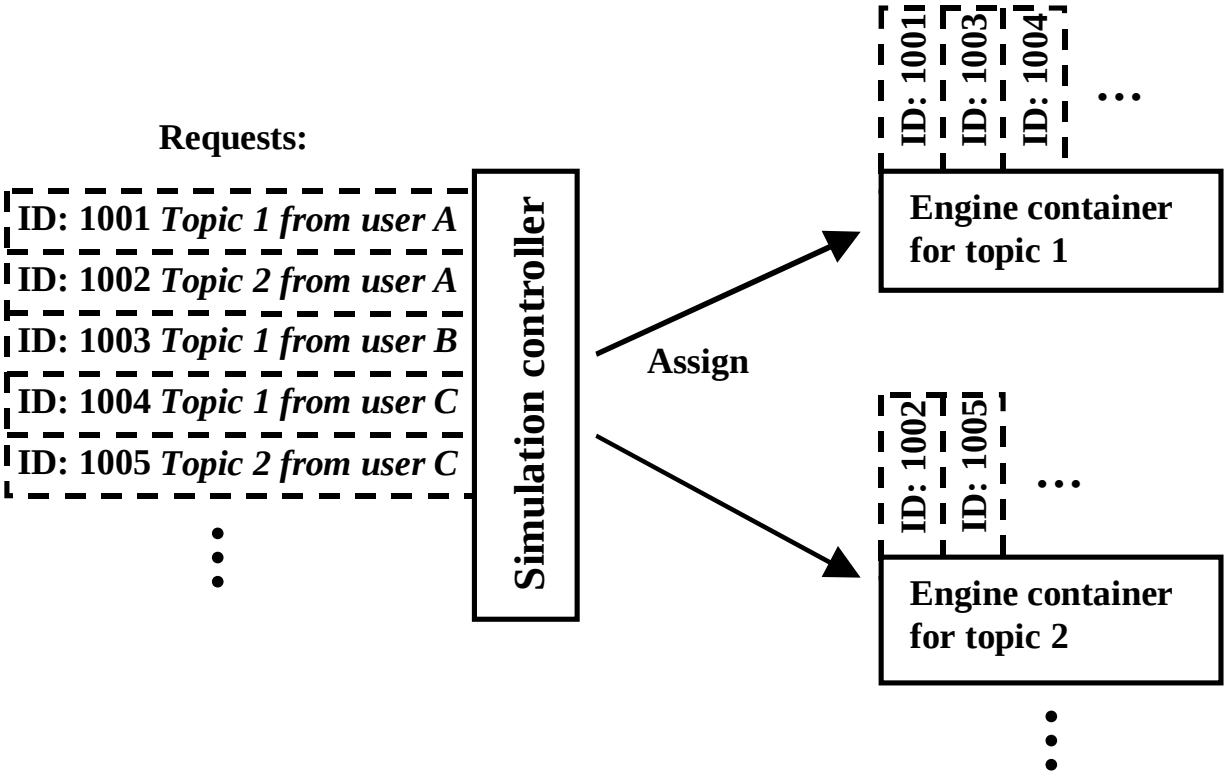
Requests:
ID: 1001 Topic 1 from user A
ID: 1002 Topic 2 from user A
ID: 1003 Topic 1 from user B
ID: 1004 Topic 1 from user C
ID: 1005 Topic 2 from user C
Simulation controller
Assign
ID: 1001
ID: 1003
ID: 1004
Engine container for topic 1
ID: 1002
ID: 1005
Engine container for topic 2

Figure 2

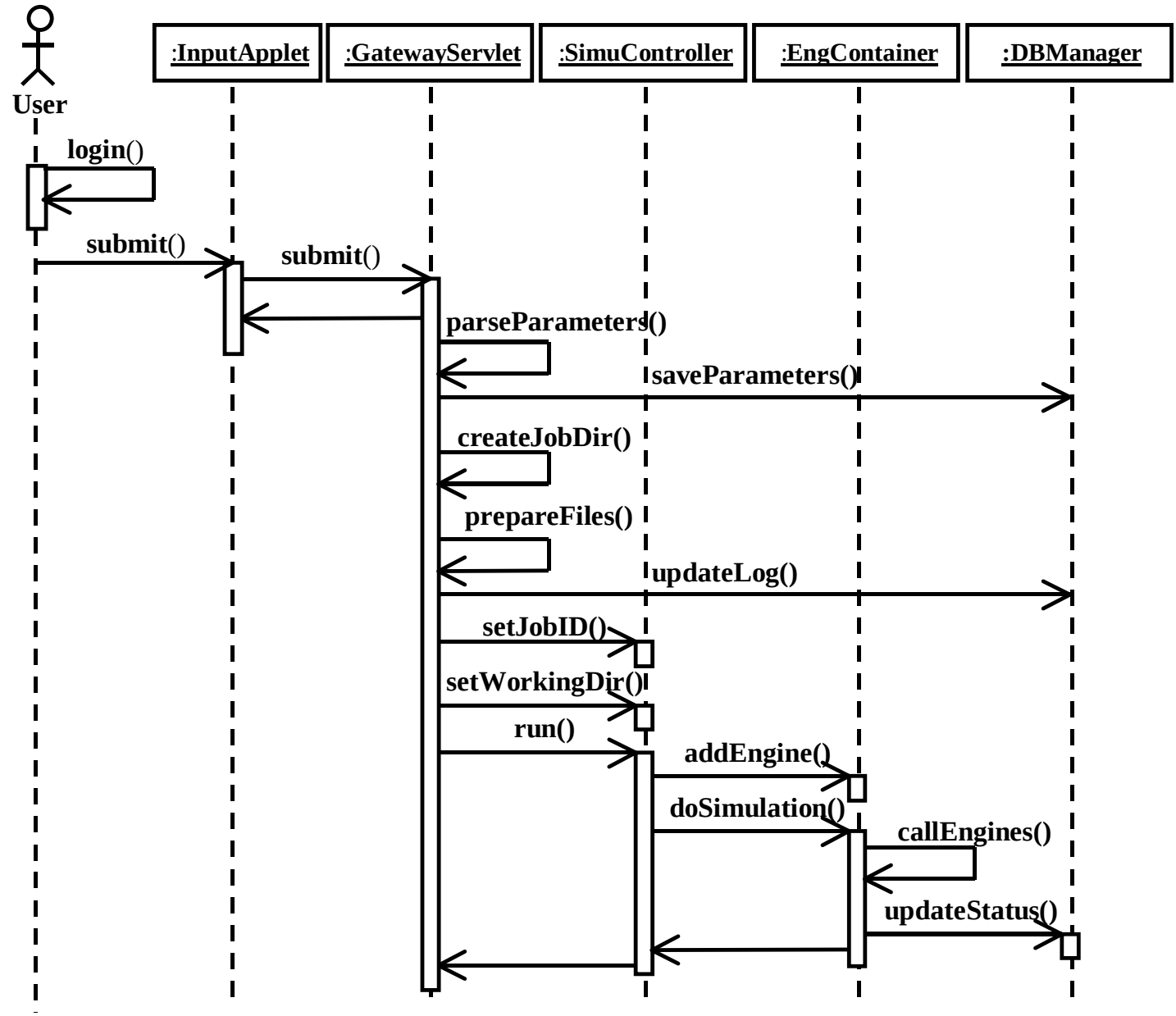
User
:InputApplet
:GatewayServlet
:SimuController
:EngContainer
:DBManager
login()
submit()
submit()
parseParameters()
saveParameters()
createJobDir()
prepareFiles()
updateLog()
setJobID()
setWorkingDir()
run()
addEngine()
doSimulation()
callEngines()
updateStatus()

Figure 3

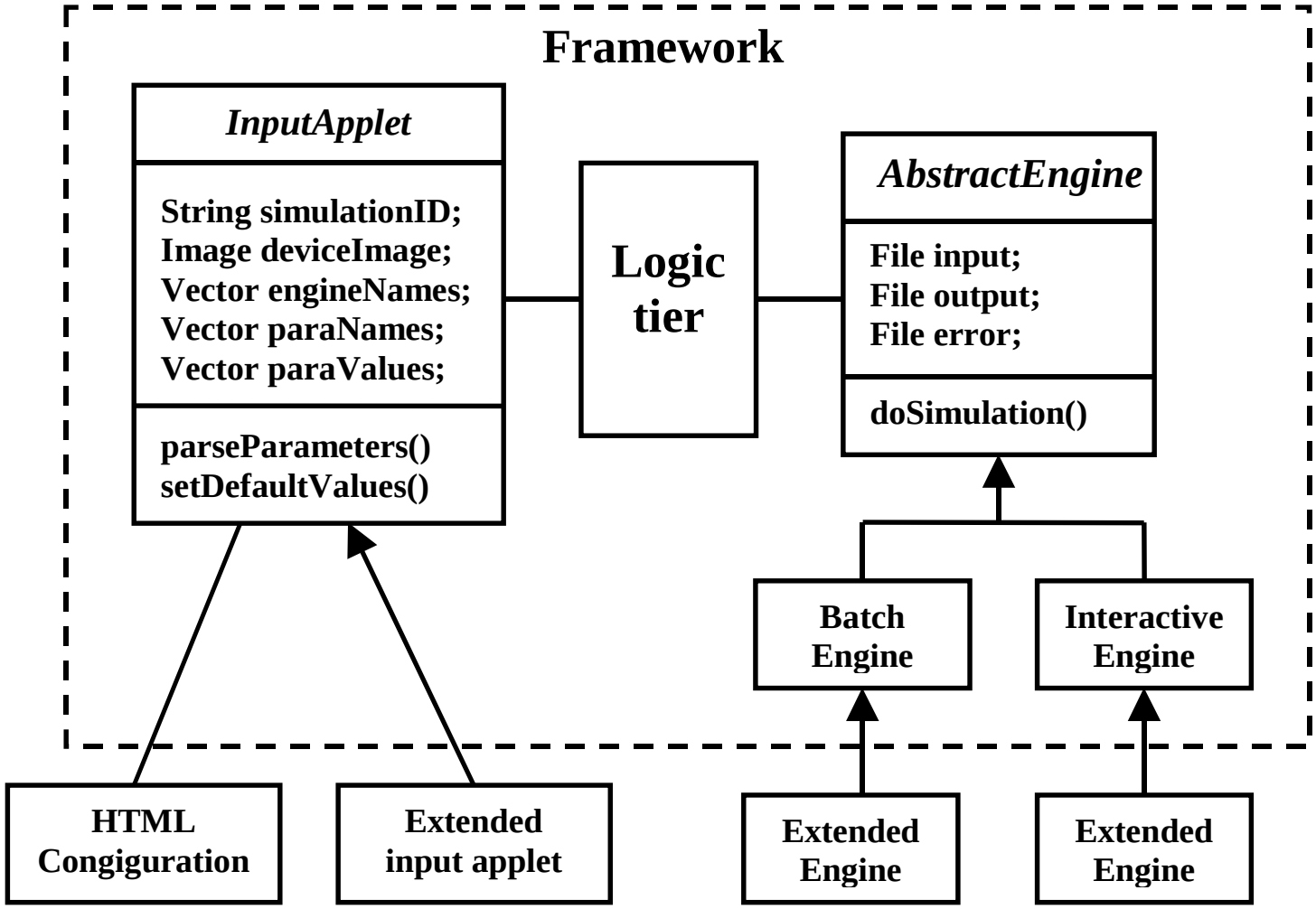
Framework
InputApplet
String simulationID;
Image deviceImage;
Vector engineNames;
Vector paraNames;
Vector paraValues;
parseParameters()
setDefaultValues()
Logic tier
AbstractEngine
File input;
File output;
File error;
doSimulation()
Batch Engine
Interactive Engine
HTML Congiguration
Extended input applet
Extended Engine
Extended Engine

Figure 4