

DINO - Distributed Interactive Network Objects The Java Approach

**Dieter Freismuth, Denis Helic, Georg Meszaros, Klaus Schmaranz,
Bernhard Zwantschko
Graz University of Technology, Austria
{dfreis, dhelic, gmes, kschmar, bzwan}@iicm.edu**

1. Motivation

Currently the World Wide Web is a static information providing system. The Internet community yearns for new, highly interactive systems for communication and information access or exchange. Today's Web browsers allow to view multimedia pages. Each user acts independently, there is no communication and collaboration between them. Existing solutions like chatrooms, news, or conference systems just satisfy one particular demand. In addition, they require different system environments, are often platform dependent and most important, they do not cooperate with each other.

What we need are small, cooperating tools which are dynamically loaded on demand. The tools should work inside the most usual Internet environment - the Web browser. Therefore the tools will be written in Java [JavaSoft], allowing them to run platform independently on any browser implementing the Java Virtual Machine. Thus the tools will be available to almost any Internet user. Some examples for tools are Chat, Whiteboard, Guided Tour, Courseware, and Authoring Tools.

To allow the tools (or modules) to communicate with each other we need a basic communication system. The main functionality of the communication system is to register and connect loaded modules as well as to provide a means of exchanging messages. Extended with a number of system modules providing facilities like network access or synchronous collaboration we have a flexible and powerful means for building dynamic, interactive applications.

In the following sections we will introduce DINO, a Java approach to such a Distributed Interactive Network Objects system.

2. Basic Concept

The heart of DINO is the Open Messaging Architecture. Open messaging means that all the modules use messages to communicate. Messages are like containers, freely expandable and capable of holding any kind of object. Messages are either routed through the system to distinct receivers or they are broadcasted. Each module connecting to the system gets its own messaging module. This messaging module serves as an interface for the module and provides an API to communicate and collaborate with other modules. Thus writing new modules is rather easy.

A DINO session is initialized by a Java applet embedded in any HTML page. The applet itself may for example appear as a clickable image. When activated the applet loads a distinct module, e.g. a chat module. This module then registers at the Communication Module, the kernel of DINO. This primary registration causes the whole system to establish itself, i. e. all basic system modules are loaded and connected. As the result of the registration, the chat module gets its messaging module and thus is able to send and receive messages. Any further module loaded either interactively by the user or by other modules will register and thus connect to the established DINO system.

A predefined set of system messages and system modules provides basic system mechanisms like killing modules, getting a list of registered modules, making protocol independent network access, opening a synchronous collaboration session or redirecting messages. These facilities will be dealt with in detail in the following sections.

3. Open Messaging Architecture

The kernel of the system is the Communication Module. It serves as an intelligent control and registration center. Each loaded module registers at the Communication Module and thus is transparently integrated into the system. Integration in this context means it has access to the API of a Messaging Module which takes care of all message exchange and routing functionality. Thus a very clear and clean separation is established. The system kernel is a number of interconnected Messaging Modules controlled by the Communication Module. The system periphery modules are connected to the system via individual Messaging Modules. The distributed internal system architecture is fully transparent to the single module.

A module sends and receives messages only via its Messaging Module. Therefore it creates a new message object and fills it with the desired information. The message is sent by delivering the message object to the Message Module. Each Messaging Module has a unique address and thus messages can be sent to distinct receivers. If no receiver is declared within the message it is broadcast. Message delivery within DINO is performed asynchronously. That means all incoming messages are queued within the receiver's Message Module and the receiver is notified. Thus, even if the receiver is busy the whole messaging system is never blocked.

Messages are like containers capable of taking any contents. Each message knows its creator, sender and receivers and is therefore very easy to route through the system. To improve message distribution performance, the Message Modules get a virtual point-to-point connection on demand controlled by the Communication Module. DINO predefines a number of special system messages that are used to control the communication, to kill modules or to access system modules. System modules are network access modules or synchronous collaboration modules. Another set of system messages is used to notify modules of environment changes like newly loaded or killed modules.

One of the most important features of the Open Messaging Architecture is its ability to redirect messages. This means, modules are able to grab messages sent and/or received by distinct other modules before they are delivered. As an example, one special module can control one or more other modules by grabbing some of their incoming and outgoing messages. The special module is then able to look inside each message and decide to manipulate it, throw it away or just let it pass its normal way. This facility can be used to increase the system's intelligence by adding intelligent daemons. It can also be used to coordinate or synchronize modules that were originally not designed for cooperative work. This approach extremely increases modularity and simplicity of modules. To demonstrate the power of this capability here are two examples:

- **Bandwidth Monitor:** A local module monitoring the network bandwidth grabs all event messages from a whiteboard application before they are sent over the Net. This module knows that certain messages like mouse moves can be packed together into a single one. In this case only one out of n messages need to be sent over the net. In addition the module could probably reduce the resolution of the whiteboard if the Net connection bandwidth is low. Thus interactivity is granted by reducing the quality of the output even for low bandwidth connections. The original whiteboard module does not even have to know about this feature.
- **Cache:** A number of modules fetch documents via the Net. If these modules are implicitly related to each other they will probably request the same documents. In addition, some modules will provide functionality like "replay" or "back", causing the same document to be fetched repeatedly. Therefore a cache module could be implemented, grabbing all network access messages. All incoming documents are cached by the module and then passed on. If a module requests a document that is already residing in this cache, the cache module intercepts the request and immediately delivers the local document to the requester.

4. System Modules

As we have seen by now the concept of the Open Messaging Architecture provides the ability to write small modules and makes extension of the system capabilities easy through special mechanisms. The minimum set of modules any developer can rely on is called the System Module Set. In this section we will introduce a number of important system modules and the facilities they provide.

- The Communication Module: As described above, the Communication Module is the heart of DINO. It pumps the blood of DINO, the messages through the system and connects all organs, the modules, together. In addition it takes care of the overall module management like loading and visualizing available modules as well as destroying active modules that are no longer needed.
- The Manager Module: The brain of DINO is the Manager Module. DINO is not only designed to run within a Java capable browser, but also as a standalone application. Therefore different modules are needed to adapt the system to the environment. The Manager Module loads the appropriate base modules and coordinates them. Different browsers offer varying facilities to Java applets and therefore tailored Browser Control modules are available. The Browser Control module provides a transparent means to control and use browser functionality like displaying HTML pages. Thus DINO modules can use browser capabilities with simple, browser independent messages.
- The General Internet Connection Module: If DINO needs to communicate with other DINOs or servers, it uses its sense organ, the General Internet Connection Module (GICM). This system module serves as a link between the DINO and the Net. It provides highly abstracted, protocol independent, flexible and asynchronous access to the Web. Modules use abstract command messages to use the GICM. As an example think of an "ls" command: if the requested "list" concerns an FTP server, the GICM opens an FTP connection and delivers a list-object to the requesting module. If the server is not an FTP but a Hyper-G [Maurer 96] server, a "getChildren" command will be sent through a connection using Hyper-G Client Server Protocol. But the requesting module will be delivered a well defined list-object in both cases. Thus different protocols are transparent to DINO modules. To provide this protocol independent feature, the GICM uses dynamically loaded protocol handlers. If a new protocol is needed, the GICM searches the local file system as well as the Web to locate an appropriate handler. It is simple to add new protocols to the system by writing a new protocol handler. With this level of abstraction it is easy to implement HTTP-NG or adding security mechanisms to existing protocols. Since modules may use the GICM concurrently, the GICM is designed to support asynchronous protocols to increase network performance.
- The Synchronous Collaboration Module: To let DINO communicate with other DINO systems we need a means of connecting a number of distributed clients together. Therefore we require servers with a session oriented client server protocol. Hyper-G meets this requirement and in addition offers the functionality of message passing between users. Therefore the first implementation of a distributed DINO system is Hyper-G based. If a module requests a connection to another user, a Synchronous Collaboration Module is loaded. This module then connects to the nearest Hyper-G server via the GICM and asks for the user. If the requested DINO user is connected to that Hyper-G server, the messaging facility of Hyper-G is used to establish a virtual point-to-point connection. The DINO system of the collaboration partner automatically loads a Synchronous Collaboration Module, which then holds the logical connection. The Synchronous Collaboration Module provides transparent message passing. That means, that

DINO modules can address distributed modules, that are modules of remote DINO the same way they address local modules. Thus implementing synchronous collaboration tools like Chat, Whiteboard or TALE [Freismuth et al. 96] are as easy as writing a simple applet.

If the DINO systems of two users are connected to different Hyper-G servers, we take advantage of the Hyper-G server interconnection architecture. Thus all DINO can be connected using fast server-server connections. If a DINO user joins more than one session with remote users, the system just has to run an adequate number of Synchronous Collaboration Modules. Think of two simultaneously used Chat Rooms as an example.

A number of additional system modules provide access to local system resources and allow the Graphical User Interface of modules to be arranged and merged. DINO allows building of applications at run-time. It is an open, flexible and system independent approach. Designing and adding new modules to the system is very easy through the Open Messaging Architecture. System modules provide a transparent access to the Web and support synchronous collaboration with other Web users. DINO runs within the most common Web applications, the browsers, which should guarantee a high user acceptance. Because of the distributed approach, the whole system is highly scalable.

5. Example Applications using DINO

One of the most important features of DINO is its support of synchronous collaboration. Therefore we want to introduce some of these tools that use DINO. All of these modules may be used separately or in any combination. If a number of users join a collaboration session, using for instance a Whiteboard, and one user wants to chat, this user only needs to add a Chat module to the session. All other users, joining the same session will automatically get the Chat module too. Thus the character of a session can be dynamically changed on demand during run-time. If a user joins a session in progress, the user's Synchronous Collaboration Module loads the currently used modules and sets their state. Thus the new user gets information about the history of the session. In the following we will provide some example applications using DINO to make the benefits clear.

- **Whiteboard:** In principle a Whiteboard is a drawable area. Several Whiteboard modules communicate by sending messages containing drawn objects like lines or circles or even text. Incoming objects from other whiteboards are interpreted and the object is drawn. But in addition DINO allows to merge the GUI of different modules. Merging a Whiteboard and an image viewer, users may discuss the viewed image by overlaying marks to parts of the image.
- **Guided Tour:** Imagine two or more users, one of them in the role of a teacher, the others in the role of a student. The teacher knows about locations of relevant material on the Web. Using the Guided Tour, students' browsers follow the teacher who is navigating the Web. The students browser will always display the

- documents the teacher is currently accessing. Therefore the teacher guides the students through specific documents. In addition with other synchronous tools like a Chat Module, this tool allows users to discuss interesting material. With the ability to change roles each user can contribute to a fascinating session.
- TALE - Teaching Adaptive Lessons Electronically: TALE, a new courseware system, is currently being developed at the IICM. It is completely built on DINO, which in conjunction with Hyper-G makes it possible to deliver the first usable platform independent student/teacher environment. Not only recording a reusable courseware session is possible but also synchronous communication between students and teachers. If desired also parts of the communication are recorded. Such a recorded session can then also be used asynchronously by students to work through a lesson at the desired speed, depending on the students level of skill and interest.

6. References

[JavaSoft] JavaSoft Inc.: "Java Home Page", <http://www.javasoft.com>.

[Maurer 96] Maurer H. ed.: "HyperWave - The Next Generation Web Solution", Addison-Wesley, (1996).

[Freismuth et al. 96] Freismuth D., Helic D., Meszaros G., Schmaranz K., Zwantschko B.: "TALE - Teaching Adaptive Lessons Electronically - Professional Distance Education Using Hyper-G Technology", submitted to ED-Media '97.