Booster: A WWW-based Prototype of the Global Software Highway

Michael Baentsch, Georg Molter, Peter Sturm
Department of Computer Science, University of Kaiserslautern,
P.O. Box 3049, D-67653 Kaiserslautern, Germany
E-mail: {baentsch, molter, sturm}@informatik.uni-kl.de

Abstract:
In this paper, a framework for globally distributed software development and management environments, which we call Booster *is presented. Additionally, the first experiences with* WebMake*, an application developed to serve as an experimental platform for a software development environment based on the World Wide Web and the Booster framework is introduced. Booster encompasses the basic building blocks and mechanisms necessary to support a truly cooperative distributed software development from the very beginning to the last steps in a software life cycle. It is thus a precursor of the* Global Software Highway*, in which providers and users can meet for the development, management, exchange and usage of all kind of software.*

Keywords:
Software development environment; Computer supported cooperative work; World Wide Web; Structuring Approach; Global Software Highway

1. Introduction

Experiences gained by employing global information services foster the realization of a world-wide infrastructure in order to boost the interaction and cooperation among software developers and users. Such a *Global Software Highway*—in its final version—interconnects developer teams all over the world, organizes all artifacts needed during software development, and coordinates all tools and human beings involved in the process of software development, management, and usage. It enables all participants on the highway to put products including designs, sources, and documentations—complete or in parts—on a global information network (provided that sufficient protection is guaranteed). By these means participants are able to import foreign software directly (like including a single header file) regardless where the provider is located, and to develop and manage new software within this global environment. The highway also offers the opportunity to take advantage of additional computing power (e.g. for distributed compilation) and of special hard- and software environments available somewhere else, and to provide up-to-date documentation as well as remote on-line diagnosis. As a whole, the software highway offers the opportunity to globalize the trading of software products, capabilities and ideas on virtual market places.

A software highway must be well-structured and well-organized in order to reach the destinations in time (i.e., to find the required software solution with signs well-placed) and to avoid—or at least reduce—problems such as diversions, detours, various kinds of accidents, or traffic jams (e.g. not everybody all over the world should access GNU software directly from MIT). For these reasons, a far-reaching infrastructure must be defined to organize all objects, subjects, and activities on the highway and a flexible framework must be realized to incorporate all the various tools used by different groups for software development. The WWW is, for several good reasons, a promising candidate as a platform for the global software highway, although the complex structure of the highway cannot be mapped directly on the single-level graph structure of the Web. Therefore, the mechanisms available in the WWW have to be augmented in order to enable the structuring of complex systems by means of hierarchies and tailorable views with the goal to integrate the software highway seamlessly into the existing Web.

In the following section, we present the essential properties and some sample scenarios of the global software highway we advocate; section 3 consists of a discussion of the Booster system, a first version of the infrastructure and the framework needed to realize the global software highway; WebMake, an implemented set of tools which provides the basic functionality for software sharing and distributed compilation in the context of the Booster system, is presented in section 4. This paper concludes by comparing our ideas with related work and by discussing the next steps to be taken to complete the initial prototype of the Global Software Highway.

2. The vision of a Global Software Highway (GSH)

Developer teams and research groups nowadays carry out software projects in compliance with a chosen software development methodology and by using accompanying tools to organize and support the development process. In

Appeared in the Proceedings of the Second International Workshop on Services in Distributed and Networked Environments, June 1995, Whistler, Canada
ISBN 0-8186-7092-4, ©1995 IEEE

many cases, the software developed by such a group does not only form a ready-to-execute product, but it is also intended to be used directly by other teams as part of their software project (e.g. as libraries for given architectures). Although for this purpose some form of cooperation is needed, besides basic network services such as electronic mail, there is only little tool-mediated direct interaction between several groups during the different development phases, especially if distinct development methodologies are used. Instead, software sharing and re-use is facilitated only on the level of completed products such as sources, libraries, or server programs. Currently, in a networked environment this is quite often done in an off-line and somewhat archaic fashion by using file transfer services to obtain a complete package including executables, documentation, and possibly sources together with installation procedures. Assuming that the hard- and software requirements can be met, the foreign software then can be installed and (re)-used locally by the client group as a whole.

Global information systems such as the World Wide Web bear the potential to change dramatically the perception of software development on a global scale. Such a *Global Software Highway* (enriching the scope of the information superhighway proposed by Bill Clinton and Al Gore) might serve as a world-wide marketplace for developing, distributing, and locating software as well as all related documents on a potentially commercial basis. The goal of the GSH is to organize all artifacts and to coordinate all tools and human beings involved in the process of software development and management. Since no single development methodology is suitable for every software project, such a highway must serve primarily as a framework which provides all the required basic mechanisms. The methodologies are build on top of this infrastructure, with the corresponding tools fitting into the overall architecture. To this end, the software highway is not intended as—and cannot be—a mega environment combining existing methodologies and their tools into a complete and fixed whole. Instead, it must strive to provide the minimal glue to enable co-existence between distinct methodologies used in different groups, to achieve interoperability between the various tools used, and to augment the development processes with additional functionality concerning aspects of inter-group relationships.

This required basic functionality to be provided by the global software highway is centered around four distinct areas:

- “*Creating*” comprises all activities dealing with the development of software as well as the compilation and linkage of libraries or executables. It is the primary domain of software development environments. For this purpose, the software highway has to provide a global repository for secure storage, versioning, and retrieval of all documents. By utilizing the enormous distributed computing power available globally, the highway may also provide mechanisms for different kinds of remote compilation and execution services: distributed compilation of client software (possibly for a fee), remote compilation of a product on trustworthy machines to obtain a version for a hard- and software environment not available at the producer’s site, as well as on-demand assembly of a product’s current version or any earlier one some client depends upon.
- “*Offering*” deals with the appropriate announcement of available software products, of software requests issued by some group, and of design and programming capabilities some group wants to sell to interested consumers. The highway enables the dynamic establishment of various marketplaces where different subject areas are traded and different marketing strategies may apply (e.g. software department stores with fixed prices, software stock exchanges or software bazaars). It is also the place where independent traders might survey the accessible information as stored in the highway’s repository and produce attention-grabbing advertisements at high-yield markets.
- “*Finding*” deals with the formulation and specification what kind of software product is needed, which software requests a developer group wants to accept, and which group (resp. its capabilities) suits best a client’s requirements. Whereas “Offering” covers all subjects that have something to sell at a marketplace, “Finding” represents consumers walking around and looking for products to buy, requests to accept, and capabilitites to employ. Instead of the consumer himself, authorized autonomous agents and independent brokers might examine the market to compare and evaluate competing offers.
- “*Using”* covers all aspects of how to obtain, install, integrate, execute and maintain software (e.g. libraries, programs and accompanied documents) a group is interested in and willing to pay for. Therefore, the software highway has to provide mechanisms for obtaining the required package for a certain hard- and software environment. Consumers also need to be notifyied about new versions of a used package and they are interested in remote on-line diagnosis and debugging facilities. The whole area of “Using” requires licensing agreements between provider and consumer and counterfeit-proof accounting once-in-a-lifetime, once-for-each-version, once-for-each-execution, or on some other individually negotiable base.

Two very important qualities of a global software highway effect all four functional areas: *security* and *accounting*. The highway’s distributed repository, responsible for global document storage and retrieval, forms the integrating component for all the tools and users. There are good reasons for users to put complete software products including sources on a global information network (e.g. to take advantage of additional computing power, on-line documentation and diagnosis, automatic versioning and notification, globalized marketing of products, capabilities, and ideas). This global access to sensible information bears the potential of misuse. Therefore, security and accounting are crucial issues to guarantee the copyrights of individuals, groups and organizations, to establish fine-granular accounting schemes, and to carry out payment for services provided by third-parties such as traders, brokers, and market organizers.

3. Booster—A framework for the Global Software Highway

A software development environment designed to be extensible up to a truly global system cannot be realized as a single program or by a single set of tools. Instead, it must be designed as a general framework able to incorporate a large variety of tools needed by the different persons and groups involved in the software life cycle. It must be adaptable enough to be used with completely different development methodologies, such as structured program development or object-oriented design methods. With these requirements in mind we have designed *Booster*, our framework for the Global Software Highway. It consists of a **repository** which offers a persistent storage facility for all kinds of design artifacts, and a **set of tools** which are either specific solutions for certain tasks, such as editing special kinds of design documents, or coordinate other tools in order to enforce a specific policy (such as the use of a certain development method).

In this section, first the abstractions are introduced which we use to model the overall structure of a generic software development environment. In section 3.2, we present an abstract view to the application-specific classes which were necessary for implementing a first version of Booster.

3.1 Structural abstractions

In order to describe the components within the Booster framework, a general structural model was developed which contains abstractions for all objects, subjects and activities appearing in a general software life cycle.

These basic abstractions form a class hierarchy, which was designed using an object-oriented methodology [4], starting with the four basic functionalities and the scenario presented above. In this section, the most fundamental classes are described. The architecture of all components within the Booster framework is represented by graphs consisting of *typed nodes* and *typed edges*, which are modelled by the abstract `Node` and `Edge` superclasses in the class hierarchy. All entities in an actual structure description are instances of specialized `Node` and `Edge` subclasses, which have a concrete semantics in the context of a certain application.

Nodes are interconnected by directed edges, which describe the static structure of the modelled system as well as the communication channels for the interconnected entities. Only the originating node of an edge can initiate an action by passing a message to its peer. This results in a request-reply interaction scheme consisting of a query and an arbitrary number of response messages. Edges in the structural model can moreover show a polymorphistic behavior: depending on the types of the interconnected nodes, the information and its representation transported across an edge may be transformed according to the requesting client's needs.

`Description` nodes are a first specialization of the abstract `Node` superclass introduced above. They represent all artifacts in the modelled system which are atomic in the structural model, i.e., entities not composed of subobjects. Application-specific `Description` node subclasses represent the semantics of a given node in the modelled system.

The `Structural` node class is the abstract superclass for all structuring elements in our model. As its instances are `Nodes` themselves, they can be grouped by other `Structural` nodes describing a higher level of abstraction in the graph model. This allows the recursive construction of arbitrary hypergraphs, which is the basic technique we apply to model complex structures. An example description is shown in figure 1.

The first kind of structural Nodes, termed `Cluster`, encapsulates other nodes, thus allowing the construction of arbitrary node hierarchies. A `Cluster` node hides the subgraph it contains, thus abstracting from the details described inside. By expanding a cluster node, the contained subgraph which models the next level of abstraction within the system under development is exposed for traversal and manipulation. A `Cluster` node is

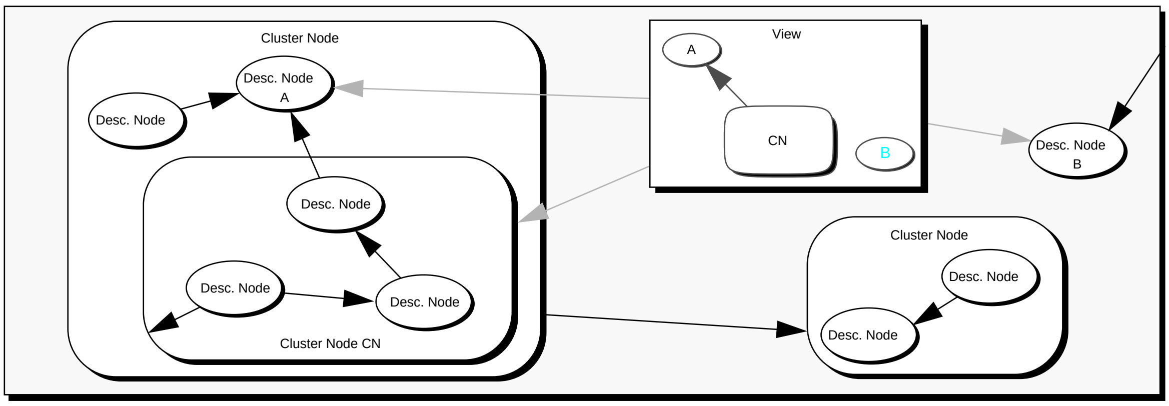


Fig. 1: Example of a structure description

not only a structuring element, but also introduces a certain semantics for the set of contained nodes, which is made explicit by appropriate `Cluster` subclasses. As an example, consider a cluster representing a library: it combines the functionality of all of its objects to offer a complete service.

The second kind of structural nodes, called `View`, provides a projection on the set of nodes and edges; as opposed to the `Cluster`, it does not contain other nodes, but only references them. While a node can belong to only one directly encapsulating `Cluster`, it may be referenced by an arbitrary number of `Views`.

By allowing the customized projection of details only relevant in a certain context, `Views` are powerful structuring instruments for managing the complex architectures of large-scale distributed systems. As an example, consider the development of a large software system: After identifying the main components and their dependencies, designer groups are assigned to subsystem development. When the requirements specification for a subsystem is sufficiently concise, there is no need to notify its implementors about internals of another subsystem. This can be achieved by defining appropriately configured structural views.

A simple example of a structure description is given in figure 1. Several general description nodes are grouped by three `Cluster` instances, and a `View` references some of the nodes and clusters, thus abstracting from details not relevant in a certain context.

In the abstract modelling level described in this section, there are two additional edge types derived from the general `Edge` class: `StructureEdge` instances connect `Structural` nodes and the nodes they contain; this applies especially to `Cluster` nodes and their contents, but is not shown in figure 1, as the `Clusters`' internal structures are exposed in this illustration. The `StructureEdge` subclass `ViewEdge` models the relationship between a `View` and its referenced nodes; in this figure, `ViewEdges` are shown in light gray. All other edges have a specific meaning in the context of the application; in the structural model, they are represented by general `Edge` instances and get their semantics by appropriate subclasses.

3.2 Specialized classes in the context of Booster

While the abstractions introduced in the previous section offer the basic building blocks to model the structure of a software system, the classes described here are representations of entities required for the design or implementation of a software tool or a service needed in a distributed cooperative development environment. The resulting class hierarchy is depicted in figure 2.

The basic artifacts for all software development activities are different kinds of *design documents*. They contain all the information collected during the design process about the system under development. In our structural model, all design documents are represented by `Description` node subclasses implementing the specific semantics of the given node type, such as `Source`, `Definition`, `ObjectCode` or `Executable` nodes. These classes also contain information about the way in which nodes of a certain type may be manipulated (e.g., a `Source` node or a `Definition` node is editable, while an `ObjectCode` node is not, at least not with a standard text editor).

Besides these classes describing concrete design artifacts, the structural model contains more abstract representations of entities appearing in the development process, such as `Tool`, a `Description` node subclass which models the properties and the behavior of each program used in a software development process. `Tool` subclasses model more specific kinds of tools, for instance compilers, source code or notational editors.

The `Person` class is the Booster representation of all people involved in the software life cycle. Instances of `Person` or one of its subclasses are used to store location information about the corresponding real-world people and serve as anchors for edges connecting the node to preferred `View` nodes; moreover, they are the place where authorization information is stored.

Specialized structuring mechanisms necessary for modeling a certain service in the Booster framework are represented by `Structural` node subclasses. Besides the structure abstractions for grouping basic software development artifacts (such as `Source Cluster` nodes containing `Source`, `Definition`, `ObjectCode` and `Documentation` nodes), the framework contains classes used to represent more abstract entities or to perform configuration or customization tasks. The `Group` class, a `Cluster` node subclass, contains a set of persons, e.g., users, developers or brokers. A specialization of this class is `Organization`, which describes both a group and the specific properties, requirements and interaction schemes of its members.

A `Person` can have an own `Workspace` node, a special-purpose `View` describing his preferences and the set of nodes currently selected to be displayed for manipulation. In order to support CSCW techniques, a `Group` subclass termed `CSCWorkers` has been designed. The persons contained in a `CSCWorkers` instance share a common view on the system under development, a so-called `Cooperative` view which can enforce a certain access policy when presenting the set of underlying nodes and edges to its users. All references to the encapsulated nodes are then dispatched using the view, which can take adequate measures to ensure consistency and integrity. Additionally, in an interactive CSCW environment all participants can be informed about relevant events.

Finally, several new edge types have been introduced to model the application-specific relationships between the `Node` subclasses, such as the `Depends Upon` edge class with the two subclasses `Includes` and `Needs Lib`, and the `Documents` edge class.

Accounting and protection are orthogonal aspects in the context of the structural model which had to be taken into consideration when designing all Booster classes. They are modelled as Mixin classes (cf. [4]) containing the necessary mechanisms to ensure uniform accounting and protection schemes.

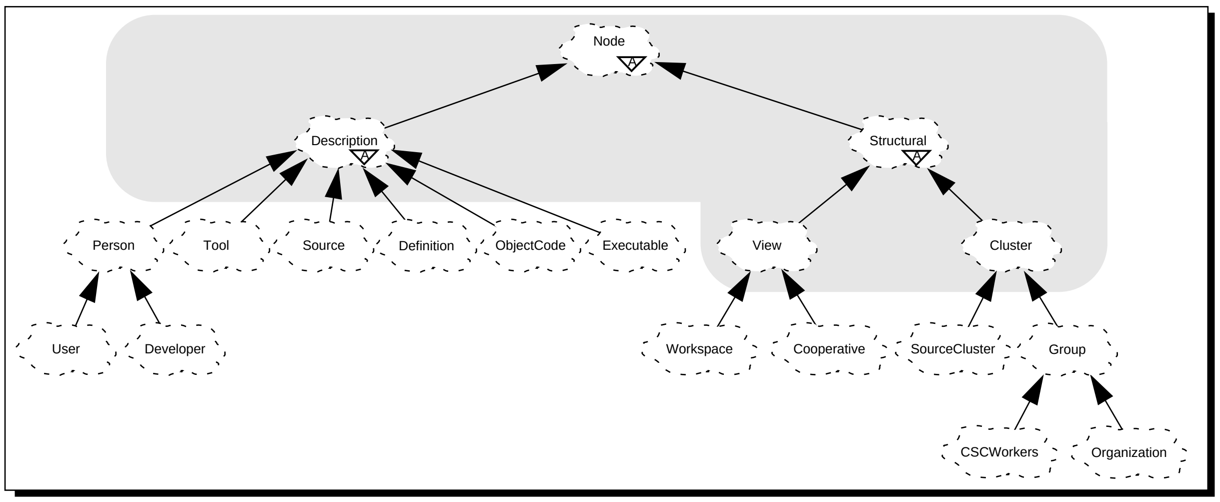


Fig. 2: Overview of the node class hierarchy needed to implement Booster. The classes on grey background are those already presented in section 3.1.

4. WebMake—a set of tools in the Booster framework

Applying the structural model on top of the World Wide Web [2], we implemented *WebMake*, a relatively simple, yet full-featured software development environment encompassing management facilities for all source and object code, executables and design documents of the system under development, each of them possibly versioned. Relations between the different artifacts, like textual inclusion or link-time, i.e., library dependencies are captured by the structures of this software development environment. The users and developers of software as well as all tools needed for distributed compilation, documentation and execution of the developed system are also represented within the set of applications making up WebMake.

4.1 Mapping the abstract classes onto the Web

The notion of full fledged classes and objects as known in an object oriented programming language like C++ is not directly available as such within the WWW. Therefore, a translation between the documents stored and returned by a WWW server and their representation in the WebMake implementation has to be performed. This happens transparently within our framework of C++ classes by reading the document level type information and instantiating an object of the appropriate class.

The mapping between the abstract classes presented in section 3 and the concrete classes needed to implement WebMake is relatively straightforward: `Nodes` are MIME [5] typed documents, e.g. HTML [3] pages, executable programs or simple source code. All specific functionality has to be realized in the subclasses of `Node`, since it is an abstract superclass providing merely the generic interface and some basic services for all nodes. To implement typed nodes, some MIME types like '*/x-c++-source' or '*/x-sparc-executable' have been introduced in addition to the already available types like 'text/html' or '*/x-gzip'. Typed edges are mapped to *Context-sensitive Links (CLinks)*, general hyperlinks augmented by mechanisms providing dynamic configurability. They have been realized by a generic cgi-bin and configuration files. Since CLinks can react to all information retrieval requests issued to a WWW server (httpd) in a context dependent way, they can provide each client, resp. server with the necessary type information needed in the extensible architecture of Booster.

All kinds of `Structural` nodes, like `Views` or `Clusters`, are mapped to specially formatted HTML pages with lists of typed edges to logically contained nodes. Thus, any WebMake-specific structural node contains nothing else than textual MIME type specifiers and hyperlinks. Cluster documents in particular additionally contain the aforementioned configuration information for CLinks, thus providing the desired access polymorphism: Any client accessing a cluster-specific context-sensitive link can now be returned exactly the—potentially dynamically built—information required.

The last basic class to be mentioned here, `Description` node, is on the one hand a superclass for all WebMake classes containing no structural node elements. On the other hand, it is the Booster representation for all 'normal' WWW pages not created particularly for the Booster framework.

The `User` and `Developer` classes are both derived from `Person`, which contain information about the real-world people represented by instances of these classes. This encompasses information like the person's whereabouts (email or snail mail address and phone number) but also about his login status, privileges, or project assignments.

All nodes representing the mere software under development are derived from `Description` node. These are Source, ObjectCode, Developed Executable, Definition, Error and Bug Report nodes modeling the respective software development artifacts. The last type of description node to be presented is the class Tool representing a specialized form of Executable node, a direct subclass of Description node modeling all client or server side programs that might be triggered by some client-induced action. It has been implemented by documents specifying the type of required input, the produced output and possibly other relevant information, for instance the description of known problems with a certain version of a given compiler.

The relations between all the software description nodes mentioned above are implemented and maintained within Source Cluster instances managing them. It is their task to provide the correct data upon demand (e.g. via the aforementioned access polymorphism provided by CLinks), to handle concurrent access to the data, and to deal with all problems of version control and consistency. For example, if some users want to edit the same source document at the same time, the first traversing the respective CLink initiates a lock on the source node, whose representation, i.e., the file, is transferred to him. The second one might now be denied access to the source code completely, or a more sophisticated negotiation process between the first and the second user can take place in order to achieve a fine-grained locking policy.

Another type of node derived from the Cluster superclass is the Group. Instances of this class primarily contain references to Person nodes, and thus represent the project teams. Derived from Group, the Organization node additionally contains interaction schemes specifying which group member has what responsibilities. These mechanisms are all realized via CLinks, which can react in a context-dependent way, e.g. who traversed an edge to an organization: If a bug report is submitted from an external user, this information can automatically be routed to the appropriate development group or person within the organization.

The last major group of structuring nodes developed for WebMake are the View subclasses. Its direct successor Workspace contains project-specific information about persons and the tools they use. Cooperative Views have to keep several users with the same view onto a common project up-to-date. To realize this functionality, the notion of an *interactive CLink (ICLink)* describing a context-sensitive link that is capable of influencing the *client* side has been developed. The process of using a cooperative view thus takes place in two stages: At first, a user connects his client to a cooperative view via a simple CLink; this interest of the user is stored with the cooperative view. A client side program which can influence the user's WWW browser is started by the MIME return type evaluation mechanism of each WWW client. In the second stage, a cooperative view's ICLink can send messages to this client side program influencing the user's current view. For example, when a new bug report has been submitted to an organization, the Cooperative View 'Project Status' must display this fact to all maintenance personnel of the appropriate development group.

On the server side, a basic security infrastructure has already been built into the respective WWW applications, but the different httpd implementations offer a varying degree of security: Almost all servers log each HTTP access with time and originating host; some also try to identify the user on whose behalf some access was initiated. This authentication is based on the RFC 931 [22] user identification proposal which every client host is free to comply to. Therefore, the client side must be trusted to faithfully return the correct per-user information to the service provider. Since malign access is still possible, if only this method is used, the following kind of authentication has been implemented for WebMake: Only RFC 931 verified accesses to server side data are permitted, if and only if the originating host is on a list of trusted hosts. Because this widely used approach does not scale well and since a general security and accounting scheme is needed for the WWW, several approaches to providing client and server site authentication are currently under development by different institutions working on the Web. The three most promising approaches are the *Secure Socket Layer* (SSL) of Netscape Communications [14], *Secure HTTP* (S-HTTP) of Enterprise Integration Technologies (EIT) [21] and *Shen* [12] of CERN. They all propose using public key systems for authentication in combination with a bulk data cypher method like RSA once a secured connection has been established. As soon as this (de-facto) standard emerges, it will be used in Booster to provide secured GET, PUT and POST access to all data under its control. However, since the authentication based on RFC 931 in combination with an access control list is encapsulated in independent C++ classes, any replacement of these classes with more portable methods of authentication requires no further modifications to the framework.

A generally accepted accounting scheme as needed for billing users of software in the context of WebMake is not yet available within the WWW. Although the possibility to tag all documents with a certain 'price' is provided in the HTTP specification, as long as a trusted secure authentication and data transfer scheme for clients and servers is not widely available, accounting cannot gain broad acceptance.

4.2 Client and server side extensions

One major development effort was concerned with the design and implementation of a new kind of WWW client. The graphical user interface (GUI) of WebMake features a direct manipulation approach to handling all displayed elements like icons representing the different types of Nodes as well as graphical edges depicting the typed Edges of the Booster framework. The client is capable of handling multiple threads of control in the sense that it can process user interface actions as well as interleaved server side induced data updates. This is necessary to seamlessly integrate the notions of views onto a graph-like structure and that of immediate response to client side manipulation of user interface elements, such as dynamically configurable, context-sensitive popup menus. The GUI as shown in figure 3 is currently not yet capable of displaying 'normal' HTML documents, because its major purpose is to serve as the interface to the world of Nodes and Edges within the Booster framework. The ability to display standard HTML data as marked up hypertext had to be realized by another client side program.

This program, called *CommandConverter*, uses the remote control facility of Mosaic [18] to cause an unmodified XMosaic 2.4 client to display any desired document. The *CommandConverter* offers this functionality to any server or client side program like our special-purpose GUI by providing a secured TCP/IP [6] port. To ensure that no unauthorized access to any local WWW browser via this port takes place, RFC 931 authentication and an access control list are used. This facility has furthermore been extended to only accept commands from any server side program presenting a valid magic number as a capability. For example, Emacs Lisp commands influencing a local editor via the gnuserv extension to the GNU Emacs can be issued via the *CommandConverter*. This application offering a kind of remote control is the client side program spawned by an unmodified Mosaic browser after having registered with a `Cooperative View` (as a reaction to the returned MIME type 'application/x-spawnRC'). The command converter then processes all further requests from the `Cooperative View`'s ICLink by causing the local Mosaic to (re)load a certain URL, thus realizing a limited degree of asynchrony to the otherwise strictly synchronous processing of client side induced HTTP requests.

On the server side, the following programs had to be developed to realize the desired functionality of WebMake. The first was the cgi-bin implementing CLinks. It can be used in one of two ways: One possibility is to process the data encoded in the HTML 3.0 forms format sent to the appropriately installed cgi-bin, facilitating a gateway access for standard browsers. Alternatively, it can process a specialized protocol developed to realize and control connections between long-running server side programs and clients as opposed to the short stateless HTTP requests. Further processing continues identically in both cases: If the context-sensitive link was used to access a cluster, the respective configuration file for the CLink is read, and according to the context, the appropriate subroutine or external program is activated. A problem that occurred at this point was how to achieve persistence: Since the respective cgi-bin is activated anew on each link traversal, a state-preserving approach had to be developed. This is currently done by server side memory mapped files containing all relevant information for subsequent invocations of the CLink.

As an additional benefit and proof of concept, we have also created a generic interface to filter programs. For instance, an on-line C++2HTML converter is now accessible at the URL http://www.uni-kl.de/AG-Nehmer/GeneSys/c++2html.html.

When a user requests an up-to-date executable from a `Cluster` node, the process as depicted in figure 4 is triggered: Considering the client host's architecture and the timestamp of the request, the cluster determines whether a new executable has to be built. If this is the case, all nodes upon which the cluster depends are recursively caused to do an update of their respective sources returning the object code for the correct architecture. If any host serving some source data is of the wrong machine architecture, it may off-load the compilation to an adjacent machine with a gateway to a compiler for the correct host architecture. Alternatively, if no such remote compilation site could be found, the mere source code can be returned. In case no error occurred, the object code is linked and the resulting executable is transmitted to the originating client which may be appropriately billed for this complete operation. If something went wrong, this fact as well as the error message is transmitted to the originating client, which collects these messages and presents them to the user. In our GUI, the icon representing an erroneous node is highlighted and thus made visible to any interested user, who can then take corrective actions.

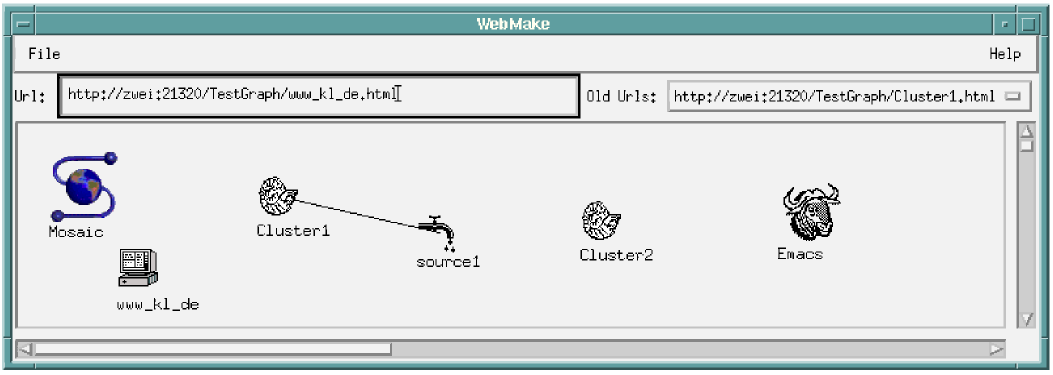

WebMake
File
Help
Url:
http://zwei:21320/TestGraph/www_kl_de.html
Old Urls:
http://zwei:21320/TestGraph/Cluster1.html
Mosaic
Cluster1
source1
Cluster2
Emacs
www_kl_de

Fig. 3: Screendump of the WebMake GUI showing several icons giving direct manipulation access to the various nodes and tools within the framework

4.3 Experiences

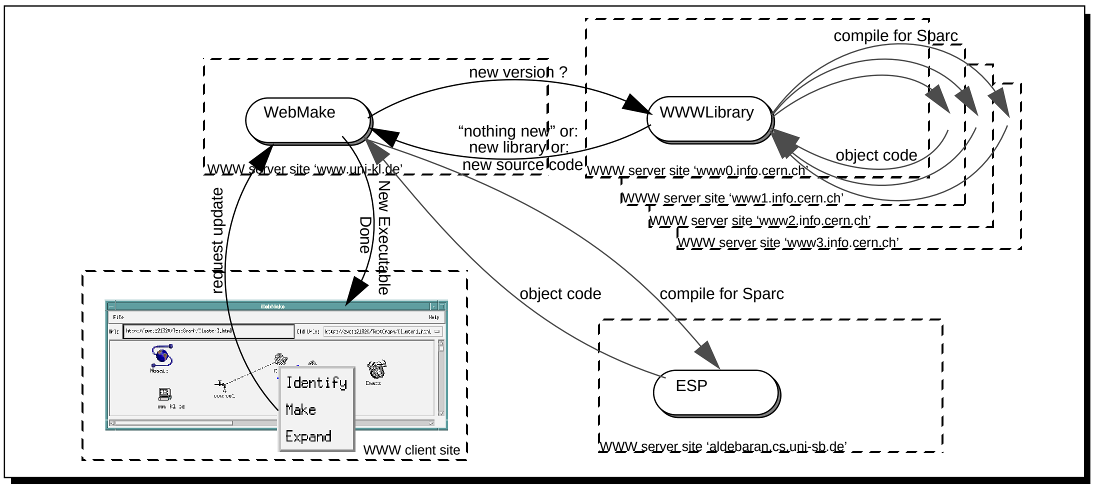
A first positive experience with WebMake has been made after putting the development of its framework under control of the WebMake mechanism itself. Thus, the development of WebMake took place on the WWW as soon as the first client and server side programs had been created. To this end, several httpds have been installed in our cluster of UNIX workstations serving the necessary software development artifacts, like source code, declarations and documentation. This installation is of course not yet really representative for the GSH, but it provided us with some interesting results. For example, when storing the data local to the host running the WWW server, a nearly linear speedup in compilation times could be achieved with each additional host. Comparing the WebMake approach to distributed software update with a non-scalable NFS based solution, another advantage showed up: The load caused by the NFS daemon programs seemed to have a more negative impact on the respective host than the httpd-based solution of WebMake. We could not reach linear speedup because of the process creation overhead incurred by each WWW document retrieval when using unmodified httpds.

One of the problems concerning the WWW mechanisms is the inflexibility of the currently cused addressing scheme via URLs: As long as a fixed internet host is encoded into the address of some document, it is not possible to transparently realize data migration or mask a host failure. Since this problem has already been spotted, a solution via the more general concept of *Uniform Resource Name*s (URNs) has been designed by the WWW community. With this, a naming scheme will be available facilitating transparent reorganization of data, which is necessary for a rapidly changing and evolving environment like the envisioned GSH.

As briefly mentioned above, the current specification of the Hypertext Transfer Protocol is another problem: Each document transfer from a server currently requires the establishment of a full-fledged TCP connection, the transmission of acceptable MIME types from the client to the server, and the server side processing of the request—in many server implementations by a new heavyweight process. Since this overhead occurs for each and every document requested, HTTP is a severe performance bottleneck for all applications requesting many documents in short periods of time. Because this is especially a problem in the context of WebMake where many documents have to be retrieved recursively, we are waiting for the distribution of HTTP-NG, the next generation Hypertext Transfer Protocol addressing these problems. An important technique to improve the overall performance is the ability to bundle requests from a given set of clients directed towards one server. This is achieved by a session layer semantics specifying a control channel over which commands and type information are sent, and logically separated data channels returning the requested information—possibly interleaved for multiple documents in parallel—to the client(s).

4.4 Further work

The already mentioned security and authentication problem will be tackled by adhering to an emerging WWW standard in this field in order to realize the portable accounting and data access scheme absolutely necessary for the envisioned Global Software Highway. The proprietary protocol currently in use to maintain connections between long-running applications will be replaced by HTTP-NG as soon as it becomes available. Furthermore, we aim to remove the process creation overhead on the server side of WebMake by developing a portable, multithreaded gateway specialized to serve the entities of



compile for Sparc
new version ?
WebMake
WWWLibrary
"nothing new" or:
new library or:
new source code
object code
WWW server site 'www.uni-kl.de'
WWW server site 'www0.info.cern.ch'
WWW server site 'www1.info.cern.ch'
WWW server site 'www2.info.cern.ch'
WWW server site 'www3.info.cern.ch'
request update
New Executable
Done
object code
compile for Sparc
Identify
Make
Expand
WWW client site
ESP
WWW server site 'aldebaran.cs.uni-sb.de'

Fig. 4: Sample of a possible collaboration between different sites participating in the WebMake approach to globally distributed software development

the Booster framework. The WebMake client for presentation of Booster documents will be extended to handle interpretable script commands. This way, the GUI can be dynamically and incrementally extended without having to be rebuilt. To accomplish this, we are currently evaluating the object oriented interpreted language Python [20]. Another extension to the framework will be the realization of (semi-)automatic integration of somehow archived source code that might be in the public domain. A tool for automatically obtaining the code, making it accessible to the WebMake tools and using it from within our framework will be developed.

The structural model underlying WebMake can also be used to organize the information chaos of the World Wide Web. The notion of Views can be employed to present data either ordered by the information provider or by the user community, e.g. by evaluating each user's rating of WWW sites and pages. As soon as we have finished the most urgent implementation issues of the Booster framework on top of the already operational WebMake, we will pursue this concept useful for anybody who finds himself lost in hyperspace too often.

Because we want to do an early evaluation of our ideas to foster discussion about Booster, the software will be made freely available. To be able to continuously evolve, improve, and validate the concept and the implementation, we are now beginning to search for alpha test sites, which would be willing to install the software in order to evaluate our concept of the Global Software Highway.

5. Related work

Since we know of no single system realizing all the features of Booster in combination, this section contains an overview about work in the related areas of Computer Supported Cooperative Work (CSCW), Software Development Environments (SDEs), and Hypertext in general.

In the area of CSCW systems, Oval [19] is a radically tailorable tool for developing cooperative applications by transforming and extending existing systems. It provides four key building blocks (objects, views, agents, and links) which are shown to be sufficient to implement complex cooperative systems. However, the structural abstractions used in Oval do not offer the possibility to form aggregations of objects, as opposed to the hypergraph structures which we explicitly designed for this purpose. The basic elements of Oval and of its predecessor ObjectLens [17] as well as those of SEPIA [13] are similar to the ones defined for Booster, confirming the presumption that our structural model is general and flexible enough for the realization of the GSH. Another CSCW system, KMS [1], features many of the capabilities we strive for with Booster, like direct manipulation of icons representing documents in a graphical browser or wide-area collaboration. However, it lacks the flexibility of the Booster approach, since it prescribes a certain, unmodifiable user interface policy, uses a too optimistic concurrency control mechanism, is founded on a database approach and does not regard links as full-fledged objects. However, Booster is not intended to provide a general-purpose CSCW environment like Conversation Builder (CB) [16]. Some aspects, like a maximum degree of flexibility and active support facilities that have been found to be advantageous in projects like CB or SEPIA, are incorporated into our model as well.

When considering challenges of large-scale CSCW system development as presented in [9], it becomes obvious that many of them are addressed by the WWW. However, some, like the cooperation and coordination problems, are only—at least partially—solved by the Booster framework on top of the World Wide Web, thus closing the gap to the more ideal CSCW environment aspired. Another indication of the validity of our model in this context results from the reflection about issues for modern hypertext systems (as opposed to first generation systems like Neptune [8]) presented by [11] in the context of the NoteCard system: Just two of the seven issues identified are—partially—solved by the WWW mechanisms, while Booster handles six of them appropriately.

In the area of software development environments, Hypercode [7] is—like our system—based on the mechanisms of the World Wide Web. But unlike Booster, which aims to use all available servers as a globally distributed storage and processing facility, it focuses on hyperlink-enriched source code contained in a central database. In [15], the members of the Arcadia project report about the most important design criteria of a modern software development environment and the tensions between them. For instance, the appropriate use of abstraction for describing functional and non-functional aspects of the system under development, as well as flexibility, i.e., the possibility to extend or replace dynamically almost all components within the environment are discussed. Because our approach is based on a general framework and an open structural model for a uniform description of all interacting tools, we consider it general enough to cope with all the problems the Arcadia implementors encountered. A system combining the advantages of hypertext with an SDE is the Documents Integration Facility (DIF) [10], which foremost aims at providing assistance while documenting software. It relies primarily on a centralized database and can thus be not as scalable as the completely distributed approach to data storage and management taken by WebMake. Additionally, it cannot be as flexible and extensible as our system, because it is tailored for a certain documentation method. In overall functionality, it can be compared to the mere documentation/source storage facilities provided by WebMake in its current state of implementation.

However, the most important advantage of the Booster framework in comparison with other CSCW, SDE, and hypertext environments is its close integration into the readily available infrastructure of the World Wide Web. Since WebMake can use unmodified WWW clients and servers, no obstacles to a wide acceptance and installation of the respective service programs exist. Thus WebMake is given the chance to become more generally accepted than the proprietary approaches cited above.

To round up this presentation of related work, we refer the interested reader to the URL of the Used Software Exchange (http://www.hyperion.com/usx), a first realization of a marketplace for software. Although no technical paper about this pragmatic approach exists to date, its mere existence underlines our opinion about the feasibility of the Global Software Highway.

6. Conclusions

In this paper, a first vision of the Global Software Highway providing mechanisms for world-wide software development and management is presented. We have introduced Booster, a framework containing all components needed for representing a general software life cycle as well as a structural model offering abstractions appropriate for organizing large software systems. The first prototypical application within the Booster framework is WebMake, which we have used to validate our concept and gather further experiences. Its implementation, the problems that had to be solved and our ideas about the further evolution of WebMake have been discussed.

The World Wide Web has been preferred to a proprietary environment to provide the necessary infrastructure, since it already realizes many mechanisms necessary for the development of Booster. Besides, the opportunity to combine the implementation of a useful tool like WebMake and a structuring approach sensible for the WWW community as a whole is challenging.

The experiences made with WebMake encourage our positive opinion of this approach to software development. Nevertheless, much work remains to be done. Besides realizing the immediate extensions mentioned in section 4.5, other issues have to be resolved: On the one hand, we will evaluate this approach to software development and management by experimenting with remote installations of Booster, and on the other hand, we will pursue its evolution to finally provide the set of tools needed to realize the envisioned Global Software Highway.

7. References

Because the World Wide Web is a relatively young medium, some of the cited documents are not available via the WWW itself. Therefore, only some of the references are given with their URLs at which they can be accessed in the most up-to-date version.

[1] R.M. Akscyn, D.L. McCracken, E.A. Yoder: *KMS: A Distributed Hypermedia System for Managing Knowledge in Organizations;* Communications of the ACM, Vol. 31, No. 7, July 1988.

[2] T. Berners-Lee, et.al: The World-Wide Web; Communications of the ACM, August 1994, Vol. 37, No. 8, pp. 76-82.

[3] T. Berners-Lee (editor): *HyperText Markup Language;* URL=http://www10.w3.org/hypertext/WWW/MarkUp/MarkUp.html.

[4] G. Booch: *Object-Oriented Analysis and Design with Applications,* second edition, Benjamin/Cummings, 1994.

[5] N. Borenstein, N. Freed: *MIME (Multipurpose Internet Mail Extensions): Mechanisms for Specifying and Describing the Format of Internet Message Bodies;* Internet RFC 1521; URL = http://www.oac.uci.edu/indiv/ehood/MIME/1521/rfc1521ToC.html.

[6] D.E. Comer: *Internetworking with TCP/IP, Volume I: Principles, Protocols, and Architecture,* second edition, Prentice Hall, 1991.

[7] A. DeHon, J. Brown, I. Eslick, L. Karbiner, T.F. Knight, Jr.: *Hypercode;* 2nd International Conference on the World Wide Web, Chicago, October 1994; URL = http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/DDay/brown/hypercode/hypercode.html.

[8] N. Delisle, M. Schwartz: *Neptune: A hypertext system for CAD applications*; Proceedings of the ACM SIGMOD Annual Conference, Washington, D.C., May 1986.

[9] K. Gronbaek, M. Kyng, P. Mogensen: *CSCW challenges in large-scale technical projects - a case study;* Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work.

[10] P.K. Garg, W. Scacchi: *A Hypertext System to Manage Software Life Cycle Documents;* IEEE Software, May 1990.

[11] F.G. Halasz: *Reflections on NoteCards: Seven Issues for the Next Generation of Hypermedia Systems;* Communications of the ACM, Vol 31, No. 7, July 1988.

[12] P.M. Hallam-Baker: *Shen: A Security Scheme for the World Wide Web;* URL = http://info.cern.ch/hypertext/WWW/Shen/ref/shen.html.

[13] J.M. Haake, B. Wilson: *Supporting Collaborative Writing of Hyperdocuments in SEPIA;* Proceedings of the ACM 1992 Conference on Computer-Supported CooperativeWork.

[14] K.E.B. Hickman: *The SSL Protocol;* URL=http://home.mcom.com/info/SSL.html.

[15] R. Kadia: *Issues Encountered in Building a Flexible Software Development Environment - Lessons from the Arcadia Project*; SIGSOFT Software Engineering Notes, (Dec. 1992) vol.17, no.5.

[16] S.M. Kaplan, W.J. Tolone, D.P. Bogia, C. Bignoli: *Flexible, Active Support for Collaborative Work with Conversation Builder;* Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work.

[17] K.Y. Lai, T.W. Malone, K.C. Yu: *ObjectLens: A "spreadsheet" for cooperative work;* ACM Transactions on Office Information Systems 6, 4 (1988).

[18] National Center for Supercomputing Applications: *Using Mosaic by Remote Control;* URL=http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/remote-control.html.

[19] T.W. Malone, K. Lai, C. Fry: *Experiments with Oval: A Radically Tailorable Tool for Cooperative Work;* Proceedings of the ACM 1992 Conference on Computer-Supported Cooperative Work.

[20] G. van Rossum, J.de Boer: *Interactively Testing Remote Servers Using the Python Programming Language,* CWI Quarterly, Volume 4, Issue 4 (December 1991), Amsterdam.

[21] E. Rescorla, A. Schiffman: *The Secure HyperText Transfer Protocol;* Internet Draft; URL = http://www.commerce.net/information/standards/drafts/shttp.txt.

[22] Mike StJohns: *Authentication Server;* Internet Request for Comments 931; URL=gopher://ds.internic.net/00/rfc/rfc931.txt.