CASE/CM Services Supporting Internet-Based Software Engineering

Robert P. Biuk-Aghai
Faculty of Science and Technology, University of Macau
fstrpba@umac.mo

Abstract

The widespread use of the Internet as a communication medium has opened the possibility to carry out software projects with a team of geographically distributed software engineers cooperating and communicating over the Internet. In the future, this development approach will likely become increasingly common. We argue that this necessitates the use of Internet-based Computer-Aided Software Engineering (CASE) and Configuration Management (CM) environments. In this paper a model of Internet-based CASE/CM services is presented, requirements of such systems and the underlying technology are investigated, and a prototype of such a system is introduced.

1 Introduction

The emerging widespread use of the Internet is transforming the way in which software development is carried out. Whereas in traditional software engineering a software project is carried out by a team of geographically co-located developers, Internet-based software engineering, variously referred to as distributed software engineering or virtual software engineering, is carried out by developers who can be located anywhere in the world, and may never have any face-to-face contact. In the non-commercial software world, this approach to software development has been successfully exercised for a number of years, as can be witnessed in such projects as GNU, Linux, XFree86, and others.

This new approach to engineering software systems brings with it a number of benefits. Firstly, software engineers need not be co-located in order to cooperate on a project. This has obvious cost benefits over the alternative of bringing a team of distributed members together in one location. Secondly, specialists can be shared between several concurrent projects. Finally, this approach enables any software engineer with suitable Internet access to participate in a project. This should be particularly attractive to those regions of the world that have otherwise been outside the mainstream of the software industry, such as many countries of the developing world. Many of these countries have a base of highly qualified engineers, but have difficulty in finding access to the international software market. Employing an Internet-based development approach, however, can ease the integration of software engineering competence from those countries in international software projects. We expect that in the future, Internet-based software engineering will have increasing importance for commercial software development.

For Internet-based software development to be effective, however, it must be supported by a suitable development environment. Because the face-to-face contact of traditional development approaches is absent, it becomes imperative, on the one hand to have efficient communication, and on the other hand to have access to all artifacts produced during the entire software development life cycle. To the latter end, Computer Aided Software Engineering (CASE) environments have proven to be useful in traditional software engineering [4]. For Internet-based software engineering, however, the existence of at least a CASE repository, if not a fully-fledged CASE environment, should be considered not only useful, but imperative. In addition, we agree with Brown et al. [1] that the underlying repository, and the system built on it, should integrate features of both CASE and Configuration Management (CM) systems.

Related work has been carried out by Eslick et al. [9], and van der Hoek et al. [12]. The former work presents a framework for software evolution that focuses primarily on various aspects of managing distributed program code. The latter work concentrates on configuration management specifically. The present work is distinguished firstly by integrating CASE and CM, and secondly by propos-
2 Repository Requirements

An important requirement for the CASE/CM repository is that it should use existing Internet server technology, instead of a new server built from scratch. Our motivations for this approach are firstly to utilize an existing network of installed servers, thus easing the introduction of Internet CASE/CM where a suitable Internet server already exists; and secondly to take advantage of suitable features that already exist in these servers.

The CASE/CM repository managed by the server contains all artifacts created during the entire software development life cycle, from project initiation until project termination. These artifacts may be project plans, various textual and graphical specifications created during analysis and design, source and object code, test suites and results, etc. In the remainder of this paper, we will refer to these artifacts as objects whenever they are used in the context of CASE/CM systems.

Suitable servers that are to be used as CASE/CM repositories should possess a number of features. We identify following three classes of features, in decreasing order of importance:

1. Indispensable features
   (a) Read and write access to the repository.
   (b) Restriction of access to identified users with appropriate access permissions.

2. Important features
   (a) Server-maintained attribute information associated with objects.
   (b) Support for storage of arbitrary types of objects.
   (c) Distributed storage of object collections across a network of participating servers.
   (d) Hierarchical structuring of objects in the repository.
   (e) Interoperability with other Internet servers.

3. Useful features
   (a) Migration and replication support.
   (b) Object caching.
   (c) Object annotation capability.

The requirement for feature 1a above already disqualifies existing Gopher [7] and Gopher+ [6] servers, as these only allow read access to their repositories.

Further, World Wide Web [3] servers currently do not possess feature 1b in the general way that we require, and are therefore also not suitable CASE/CM repositories. However, work on the http protocol is continuing [11, 13], and may produce a protocol suitable for our purposes in the future.

Utilizing a distributed file system, such as AFS, is a possible option for handling distribution, but would require the construction of a suitable server to implement the other mentioned features, which makes it less well suited for implementing a repository.

The only currently existing Internet server that more or less fully meets all the above requirements is HyperWave1 [2]. Section 4 will introduce relevant features of HyperWave in more detail.

3 Internet CASE/CM Service Model

The concept of CM services has already been discussed elsewhere [5]. Such services represent well-defined, implementable CM functionalities. In the present paper, we propose a layered model of services for CASE and CM that are essential in an Internet-based software development environment.

In our Internet CASE/CM service model, we define four service layers. These contain the fundamental services on which CASE/CM systems can be constructed. In general, Internet-based CASE/CM environments will be client/server systems. Servers

---

1Formerly Hyper-G; as of 6/96, HyperWave Software Inc. is continuing development of Hyper-G under the name HyperWave.
host the CASE/CM repositories, and clients access these repositories and perform certain functions on them. Figure 1 shows the relationship of the CASE/CM service layers with client and server. Note that while the services are shown as belonging to the client, some of them may actually be offered by the server, as will be explained below.

3.1 Service Layers

1. Communication service layer

Clients utilize services offered at this layer to communicate with CASE/CM servers, in order to perform certain functions on their repositories. The implementation of services at this layer is specific to the type of server accessed. Services included are: opening and closing connections, sending commands and receiving feedback, receiving and transmitting streams of data, etc.

2. Repository service layer

The repository service layer provides the primitives for reading from and writing to the repository that resides on the server. It utilizes functions of the communication service layer in order to access the repository, and like the lower layer utilizes server-specific features. Services at this layer include: reading, creating, modifying, deleting, moving, and locking/unlocking files and/or directories (or their equivalents) stored in the repository.

3. Access service layer

Access services are built exclusively on top of the lower-level repository services and provide server-independent functions for accessing the repository. Services include: retrieving a particular version of a versioned object, encrypting or decrypting an object, consolidating a sequence of version differences into a new version, importing a hierarchy of objects into the repository, exporting a subtree of the repository to the client, etc.

4. Policy service layer

Services at this layer are intended to implement certain policies that are required for a particular CASE or CM system. Policy services are implemented using services provided by the access service layer, and like that layer are server-independent. They include, for example, a checkout/checkin service, which is built on access services that retrieve or create versions of a versioned object, and perform the corresponding locking and unlocking functions. Other services include managing change-sets, consistency checking between various CASE artifacts, etc.

3.2 Implementing the Services

As mentioned in the previous section, our Internet CASE/CM model assumes pre-existing Internet servers. At the moment, only HyperWave can be considered a suitable candidate for a CASE/CM repository, but as other systems, such as WWW and Gopher, evolve, these may become viable alternatives in the future. This gives rise to the possibility of a number of different server systems with possibly very different features being utilized as repositories. In the case of a very primitive server, the burden would be on the client-side service layers to implement missing features, while a sophisticated server would allow the client to be much simpler. For instance, if the server maintains object attributes, the client will be able to manipulate these attributes simply by sending appropriate commands to the server. On the other hand, if the server has no support for object attributes, the client itself must maintain these attributes, for example by storing them in a separate file in the server, and implement all attribute manipulation operations itself. Thus there is a spectrum of cases, depending on the sophistication of the server utilized, ranging from the very simple, distributed file system approach,
The choice of a particular Internet server affects the implementation of the lower two layers. Secondly, the two upper layers remain unaffected when the server is changed. In the latter case, all that is required is to replace the implementation of the lower two layers for one server with that for another server. This modular approach makes it possible to create plug-in modules for each type of server, implementing the lower two layers. When the type of server changes, the current server-specific module is plugged out, and the appropriate new module is plugged in, without affecting existing clients or impairing client functionality.

### 3.3 Typical CASE/CM Configuration

Using the CASE/CM Service Model defined above, it is possible to construct an Internet software development environment. In a typical setting, every development site has its own local server, maintaining a local repository, mainly for objects created at that site, but possibly also containing migrated or replicated objects from other sites. Besides the local server, there are also remote servers, maintaining remote repositories. Objects belonging to a particular project or software configuration can be distributed across these local and remote repositories. Each of the different servers can be of a different type, e.g., HyperWave, WWW, etc. The union of all repositories, local and remote, forms a virtual, global repository. However, clients always access the global repository through their local server. If a desired object is stored in the local repository, it can be retrieved from there immediately; otherwise, the local server will retrieve it from the remote repository at which it is stored. In either case, the location of the object is transparent to the client. This implies that the local server needs to be capable of interoperating with the remote servers. Given that the types of servers mentioned in section 2 are for the most part interoperable, this is not an unrealistic assumption. It also implies that clients only ever need to be able to work with one particular type of server, i.e., they only need to use the plug-in module implementing the communication and repository services for the local server. Figure 3 shows a sample configuration of client, local repository server, and three remote repository servers. Client and local server are connected on the same local area network, which in turn is connected to the Internet.

### 4 The HyperWave Repository

In section 2 we investigated basic requirements for a CASE/CM server, and concluded that HyperWave is currently the only kind of server that meets those

---

2 In fact, they only can use one module, as otherwise service name conflicts between the upper and lower layers would arise.
requirements. Here we present pertinent features of HyperWave that prove to be useful for implementing a CASE/CM repository.

1. Client/server architecture

HyperWave is a client/server system, and, in addition to existing clients, new clients can be implemented, utilizing HyperWave's Client/Server Protocol [10].

2. Object database managed by the system

In HyperWave, all objects are kept in a HyperWave database. Access to these objects is only possible through the system, which greatly facilitates the maintenance of security and data integrity. Further, clients may both read objects from, and write objects to the database.

3. A variety of server-supported object types

The HyperWave server distinguishes more than a dozen pre-defined object types, such as hypertext, image, postscript, etc. In addition, any object with a user-defined type can be inserted into the HyperWave database.

4. Object attributes

The system maintains a set of attributes for each object. Depending on the object type, a number of mandatory and optional system-defined attributes exist, for example title, keyword, type, author, etc. Users can also add user-defined attributes.

5. Information structuring

HyperWave supports hierarchical structuring of objects in collections. Collections are conceptually similar to directories in a file system, in that they contain a set of objects, which may in turn be collections, or ordinary objects.

All objects stored in a HyperWave database are at some level under the server's root collection, i.e. the top-level collection in the database, and all root collections of all HyperWave servers in the world are under a single, virtual, global HyperRoot. Thus, any object of any (online) HyperWave database in the world is part of a single object hierarchy (which forms a directed acyclic graph).

6. Object linking

All HyperWave objects world-wide are identified through a unique global object ID. When objects of a collection are accessed, this global object ID indicates to the server where the object can be found, in the local database, or on a remote server. This provides an object linking facility, which makes it possible to have collections that contain both local and remote objects. In either case, the object location is transparent to the client, and it is the responsibility of the server to access any remote objects (i.e. the client only ever communicates with the server it initially connected to).

7. Built-in search facilities

HyperWave supports searching for objects through queries on the object ID, the attribute information, or, in the case of hypertext documents, on the full-text content. Searches are not restricted to the local server, but can span multiple servers.

8. User accounts, user groups, and access rights

The HyperWave server maintains user accounts and hierarchies of user groups, akin to the Unix system. Users may identify with user ID and password, or may access the server as the anonymous user. Access to HyperWave objects can be restricted to specific users and/or user groups, and different access rights can be defined for the different access modes (read, write, delete). By default, all objects are readable by anonymous users, and writable and deletable only by the owner.

9. Interoperability with WWW and Gopher

HyperWave servers are two-way interoperable with both Gopher and WWW, i.e. Gopher and WWW clients can access HyperWave as if it were a Gopher or WWW server, respectively, and HyperWave clients can access Gopher and WWW servers as if they were HyperWave servers (except that HyperWave-specific features not found in those servers are obviously not available).

10. Caching and replication

Objects retrieved from remote servers are cached in the local server for faster subsequent access. It is also possible to replicate remote objects in the local server, but at the moment, replicas in HyperWave are only snapshots, i.e. updates on the original objects are not propagated to the replicas. Nevertheless, replicas are useful for objects that are seldom or never modified: access to a remote object is automatically redirected to the local replica, if it exists, which can significantly speed up access time.
All the features just outlined prove to be very useful for constructing an Internet CASE/CM repository. The following sections describes a prototype of such a repository that is based on HyperWave.

5 ICCS Prototype

At the University of Macau, the ICCS (Internet CASE/CM Systems) project is currently underway, whose aim is to investigate requirements and issues related to future Internet-based CASE and CM systems. As part of this project, a prototype of an Internet CASE/CM system based on HyperWave is under development.

5.1 Prototype Design

To date, a rudimentary version of the four service layers described in section 3 has been defined, which will be extended as the project carries on. This is implemented as two shared libraries of C functions, one for the two server-dependent layers (library libcsrc), and the other one for the two server-independent layers (library libiccs_ap). These libraries are dynamically loaded by the CASE/CM client, which is implemented in Tcl/Tk and runs in a Unix environment.

The advantage of this design is that the implementation of the CASE/CM services can be changed at any time by replacing the services libraries, without affecting existing clients. Libraries may need to be replaced, for example, when changing the type of server, in which case only the server-dependent library libcsrc is replaced; or generally when upgrading the libraries to newer versions.

As mentioned earlier, our ICCS prototype uses a HyperWave server as its repository, consequently the server-dependent library libcsrc is designed specifically for HyperWave, employing the Hyper-G Client/Server Protocol [HG-CSP], which is an asynchronous connection-oriented application layer protocol based on TCP/IP.

5.2 ICCS Service Layers

To illustrate the use of the CASE/CM service layers, we have chosen to implement a small set of services required for supporting one particular configuration management model, namely the familiar checkout/checkout model [8]. As our project progresses, other services, supporting other CASE/CM models, will be added. Relevant services in the four layers are outlined below (some services are not shown). Functions implementing services of a specific service layer are prefixed by two characters indicating the layer they belong to: ic for ICCS communication service functions, ir, ia, and ip for repository, access, and policy service functions, respectively. Some of the functions depend on certain configuration information, such as the host name and port number of the server, the default user ID, etc. Certain of these settings are stored in a system-wide configuration file, others in a configuration file in the user's home directory.

1. Communication services

   icinitialize Reads configuration settings from the local configuration files, opens a connection to the local server, and logs the user in as the user specified in the user's personal configuration file.
   icterminate Closes the connection to the server.
   icsend_command Sends an HG-CSP command to the server.
   icsend_object Transmits object data over a separate connection to the server.
   icreceive_object Receives object data over a separate connection from the server.

2. Repository services

   irread_object Retrieves the specified object from the database.
   irinsert_object Inserts a new object into the specified collection of the local server.
   irreplace_object Replaces the specified object with the one supplied.
   iremove_object Deletes the specified object from the database.
   ilock_object Locks the specified object, either as non-writable or non-read-writable.
   iunlock_object Releases the lock on the specified object.

3. Access services

   iaget_version Retrieves the specified version of the object, or the current version if no version is specified, and optionally locks it.
   iaset_version Inserts a new version of a versioned object into the specified branch of the version graph, and optionally unlocks it.
4. Policy services

\textbf{ip\_checkout\_object} Checks out the specified object either in read-only or in read-write mode. In the latter case, the object is also locked.

\textbf{ip\_checkin\_object} Checks in a new or previously checked out object. If it was locked, the lock is released.

\textbf{ip\_branch\_object} Checks out the specified object and creates a new branch in the version graph. The new branch is locked non-read-writable.

\textbf{ip\_merge\_object} Merges an object in an offspring branch with an object in its ancestor branch, creating a new version in the latter branch.

5.3 Information Structuring

Software projects generally tend to generate a large number of artifacts. It is important that these are managed in an efficient way, so as to facilitate access to them. This is especially the case in Internet-based projects, that is, assuming that there is no means outside the Internet through which these artifacts can be accessed. To this end, the Internet-based CASE/CM repository is used to store these artifacts, i.e., objects in the repository. The structure of the collection of objects can generally follow the project’s structure, as defined in the project plan.

The project plan is defined at the outset of the project, and identifies project phases, their deliverables, and the responsible staff. For the ICCS system, the project plan is used to create the information hierarchy, i.e. the hierarchy of HyperWave collections. This hierarchy has a global entry point into the ICCS part of the database, which is usually the collection \texttt{iccsroot}, and which contains collections for individual projects. Each project collection in turn has collections for the project’s major phases, which contain collections for sub-phases, etc., until a task breakdown structure of the desired degree of granularity is reached. Development artifacts are stored as objects in collections of the appropriate level, which can be a collection at any level, but which will mostly be lower-level collections. For example, at the top level of the project’s collection hierarchy, the project plan and a number of other general project-related documents may be stored. However, analysis specifications, program source code, etc., will mostly be stored at the lowest level of the collection hierarchy.

Typically, a project’s manager will define the complete collection hierarchy of the project in his/her own local repository, and either maintain replicas of the various objects, or links to their location. Individual developers, however, need not do so, particularly if they are only involved in a small part of a large project. In this case, they will maintain the subtree of the project that is relevant to their work.

The CASE/CM client, utilizing the services implemented by the service libraries, presents this hierarchy to the user (i.e. the developer), to allow the selection of a particular object that the user wishes to work on. In our case, having services that implement the checkout/checkin CM model, the user can choose an object from the hierarchy and request it to be checked out, branched off from the ancestor branch, merged with the ancestor branch, or checked back in. Other services work in a similar manner.

5.4 Future Plans

The ICCS prototype is still in early development and lacks many necessary features. We plan to improve it in a number of ways:

1. More services

   So far we only have services for the checkout/checkin CM model. We need to greatly increase the number of services to cover other CM models, different semantics of the various models, and CASE services.

2. Tool integration

   CASE tools are currently not integrated in ICCS. Instead, objects need to be retrieved from the repository and worked on externally. We plan to integrate the tools into ICCS, in order to make it a complete development environment. Tool integration should be user-configureable, to allow users to add or remove their own tools.

3. Improved security

   Since artifacts of commercial projects may represent significant assets to their owners, access to objects needs to be tightly controlled. Currently, security is achieved through limiting access to authenticated users only. However, once retrieved, the objects pass in plain form across the network to the client. We plan to encrypt and decrypt all objects in the client using PGP, and to transmit and store only encrypted objects.
4. Annotation capability

Developers need to be able to annotate objects under development by others. Such annotations form a part of the project's deliverables and are to be stored in the repository. Annotations in turn may need to be annotated by others in a threaded manner. This can be easily achieved through HyperWave’s hyperlinking facilities.

5. Automatic hierarchy administration

Currently, the collection hierarchy has to be maintained manually. In the future, we plan to propagate a master hierarchy defined by the project manager to other team members’ repositories. Only the part of the hierarchy required by the team member should be propagated, including links to other members’ repositories for jointly developed parts of a project.

6 Conclusion

This paper has presented a model for Internet-based CASE/CM services, and a prototype of a system that implements several of these. These services are intended to be built on top of existing Internet servers, which maintain the repositories, and we conclude that HyperWave is currently the most suitable server for this purpose. The service model distinguishes services that are dependent on the particular server used, and those that are server-independent. This separates CASE/CM clients from the server they access, and makes it possible to replace servers without affecting existing clients.

We believe that Internet-based software development will attain increasing importance in the future, as the Internet becomes increasingly widely used not only as a means for communication, but also for cooperative work. CASE and CM environments will have an important role to play in aiding this emerging software engineering paradigm. Besides technical issues, however, others, related to the software process and management of Internet-based projects, need to be explored to be able to tap the potential benefits that this paradigm offers.

References


