Evolving a Programming Environment Into a Cooperative Software Engineering Environment

Walter R. Bischofberger, Kai-Uwe Mätzel
UBILAB, Union Bank of Switzerland
Bahnhofstr. 45, CH-8021 Zurich
{bischofberger, maetzel}@ubilab.ubs.ch

Christian F. Kleinferchner
TakeFive Software GesmbH, Austria
Jakob-Haringer-Str. 8, A-5020 Salzburg
kleinferchner@takefive.co.at

Teamwork is a prerequisite for the development of large complex software systems. In conventional software engineering coordination of teamwork is achieved by exchanging formal documents and by providing support for keeping these documents consistent while several developers are evolving them. In order to support teamwork more effectively it is either possible to provide better support for the conventional software engineering approach or to move the focus beyond coordination towards cooperation.

Each of the two follow-up projects of the Sniff project take one of these approaches. The commercial SNiFF+ project lays emphasis on the seamless integration of a programming environment and configuration management systems. The research oriented Beyond-Sniff project concentrates on the search for new approaches to supporting cooperative software engineering. [1]

1 Motivation and Overview

Software systems tend to have a more and more complex and comprehensive view of the application domains they model. The complexity of these systems goes hand in hand with the difficulties (the amount of efforts to be invested for their development) arising during their development. This is an important reason why the implementation of even modest software systems can only be carried out effectively as an iterative and cooperative process.

Communication and coordination are prerequisites to successful cooperation. Their quality strongly affects team productivity and the quality of the resulting products. Unfortunately, the costs of satisfying communication and coordination needs, quickly reach unacceptable dimensions. These costs naturally limit the size of closely cooperating teams and therefore, also the size and complexity of projects that can be carried out cooperatively.

Since we have (or want) to stretch the limits created by the negative side effects of cooperation, we need methods and tools that explicitly address communication and coordination. The provision of suitable methods and tools and their application are the core of an emerging discipline that is called cooperative software engineering.

The SNiFF+ and Beyond-Sniff projects are follow-up projects to the Sniff project carried out at UBILAB of Union Bank of Switzerland during 1991 and 1992. The result of the Sniff project was a working prototype of a full-featured comprehensive C++ development environment with a

1 Published in Procs. of CONSEG 95, New Delhi, February 1995.

special focus on openness, scalability, and browsing of large software systems [Bis92].

SNiFF+[2] is a commercial product which is a direct evolution of Sniff. Beyond-Sniff [Bis94b] is a research project that investigates concepts, architectures, and tools for cooperative software engineering. Both SNiFF+ and Beyond-Sniff intend to support large numbers of developers in cooperatively developing large software systems.

The SNiFF+ project takes an evolutionary path by refining known approaches to configuration management. The Beyond-Sniff project takes a revolutionary approach by building an innovative platform which makes it possible to closely integrate large numbers of tools applied by large numbers of developers and by writing tools on top of this platform which support cooperative software engineering in innovative ways.

The aim of this paper is to explain how the SNiFF+ and Beyond-Sniff development environments support cooperative software engineering. It starts with a short definition of our understanding of cooperative software engineering in Section 2. Sections 3 and 4 explain the aspects of SNiFF+ and Beyond-Sniff that relate to cooperative software engineering. Section 5 presents a summary.

2 Terminology

The terms "cooperative software engineering" and "computer supported cooperative software engineering" are used in different contexts with different meaning. From our point of view cooperative software engineering comprises all software engineering methods, norms and tools that support teamwork flexibly and effectively [Bis94b].
We assume an intuitive understanding of the term cooperation. Cooperation usually implies shared goals among different actors [Mal94]. Coordination is managing dependencies between activities. Coordination is an important part of cooperation.

We identify two forms of cooperation, policy-driven and informal cooperation. *Policy-driven cooperation* is done by exchange and correct handling of well-structured documents and concurrency control regarding the access to artifacts. *Informal cooperation* is characterized by the unrestricted exchange of structured or unstructured information.

Configuration management as defined by conventional software engineering is a subset of policy-driven cooperation. Examples for informal cooperation are the provision of textually annotated artifacts, e-mail messages, or the extraction of information from source code.

---

2 SNiFF+ runs on most Unix platforms and is free for universities. It can be downloaded by ftp from ftp.eunet.co.at (/pub/vendor/takefive) or from self.stanford.edu (/pub/sniff).

3 Cooperative Software Engineering with SNiFF+

One of the incentives for the evolution of SNiFF+ was to provide support for the cooperative development of large software systems. The integration of proven version control and configuration management technology was therefore a logical step. No efforts have been taken to provide specific support for informal cooperation yet.
This section discusses how SNiFF+'s configuration management system supports policy-driven cooperation and which aspects of SNiFF+ ease informal cooperation.

3.1 Policy-Driven Cooperation with SNiFF+

Software configuration management is beneficial for any large software system that, due to its complexity, cannot be made perfect for all the uses to which it will be put. Such a system will be subject to numerous and sometimes conflicting changes during its lifetime, giving rise not to a single system, but to a set of related systems, called a “system family”. A system family consists of a number of components that can be configured to form individual family members. A substantial number of the components must be shared among members to make the family economically viable. Maintaining order in large and expanding system families is the goal of configuration management. [Tic88]

Effective software configuration management coordinates programmers working in teams. It improves productivity by reducing or eliminating some of the confusion caused by interaction among team members. [Tic88]

Coordination of concurrent development is therefore one of the major task of configuration management. Pessimistic and optimistic approaches to coordination are distinguished today [Sch93].

Pessimistic coordination means that all developers work on the same artifacts. The concurrent editing of the same files is prevented by locking. The advantage of pessimistic coordination is that consistency of the system is guaranteed and contradictory concurrent changes are prevented. Pessimistic coordination is the most widely used form for a closely cooperating team. However, there are cases when the overhead for pessimistic coordination is too expensive because it needs synchronous work of the developers and therefore a continuous dialogue. Pessimistic coordination excludes:

- modification of the same artifact (file) at the same time
- cooperation over a long distance which makes closely coordinated work impossible
- development steps that take more than a day before a consistent state is reached (e.g., a large architectural reorganisation)

In these cases *optimistic coordination* has to be used. During optimistic coordination each developer works on his personal copy of the source code. From time to time, the copies can be merged into a new shared version. Conflicts between changes have to be resolved during the merge process. The advantage of optimistic coordination is that developers can be decoupled almost completely for some time. The price for decoupling is the need for merging and bigger integration steps. In the merging phase, communication that has been postponed takes place in a more concentrated manner.

Pessimistic Coordination with SNiFF+
SNiFF+ supports locking-based pessimistic coordination by seamlessly integrating different version control systems (VCSs) with an adapter architecture. SNiFF+'s project editor is a user interface to the common functionality that the underlying VCSs offer.

The project editor, depicted in Figure 1, lets the user select sets of files and check them in or out. Furthermore, the user obtains information about the purpose and modification history of a file, and its actual locking state. Differences between versions of a file can be browsed with the DiffMerge tool.

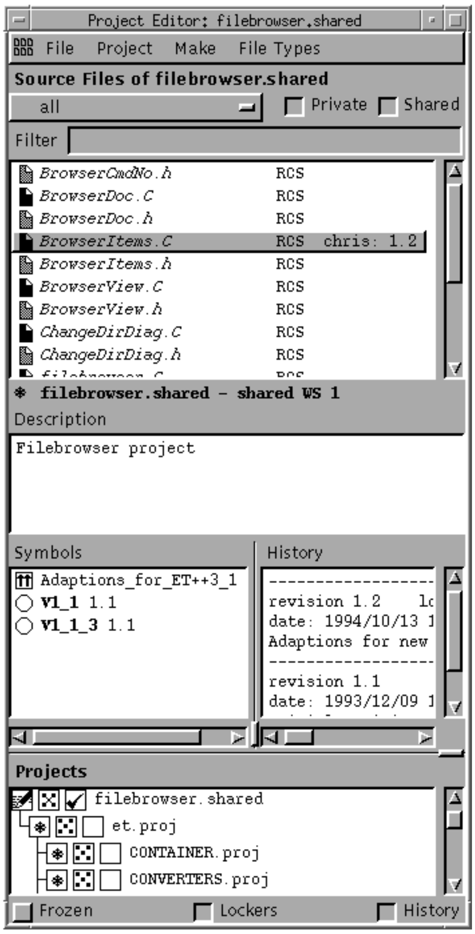

Figure 1. SNiFF+'s Project Editor

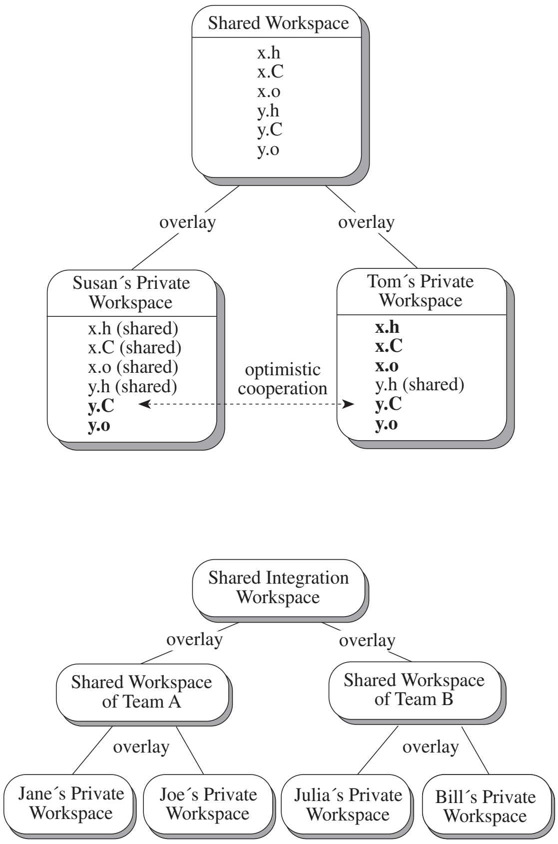

Figure 2. Application of workspaces

Optimistic Coordination with SNiFF+

During optimistic cooperation two contrary requirements need to be supported:

- decoupling the changes of a single team member from the other team members, while
- sharing as much information as possible.

SNiFF+ supports these requirements by providing the concept of overlaying workspaces. A workspace is the location in the file system where a specific incarnation of a software system is located. Workspaces of the same system can be organized into a hierarchy. Artifacts (files) present in a lower level of the hierarchy hide (override) artifacts in upper levels (see Figure 2). A hierarchy of workspaces makes it possible to decouple the work of developers (or even teams of developers).

SNiFF+ distinguishes between private and shared workspaces. A private workspace can be modified only by one team member, its owner; all check-in and check-out operations and modifications are applied to the private workspace. A shared workspace is read only for all team members and contains the configuration the team is working on. Files can only get to the shared workspace via a check-out operation from the project repository. A shared workspace has one or more administrators.

When a team member has finished a development task after successfully building and testing it in his private workspace, it is guaranteed that the private workspace and the shared workspace are consistent. The changes can then be checked-in to the repository of the shared workspace. After the next update of the shared workspace, the changes will be seen and used by the other team members.

SNiFF+'s workspace mechanism results in a high degree of flexibility for decoupling and sharing, because the scope of the decoupling and time of synchronizing changes (updating the shared workspace) can be fully controlled.

We at TakeFive, for example, are generally synchronizing and building the shared workspace over night, so that on the next day, each team member can work with the newest changes of all team members. This allows for incremental evolution and avoids the need for big integration steps and tests. The regular updates of the shared workspace are done by cron scripts. During complex project reorganizations, the shared workspace is kept at a stable stage for more than one day, until the reorganization is carried out and tested in a private workspace.

It makes sense to have more than one shared workspace if more than one team is developing a system or using the evolving system as a library for their work. Then one shared workspace would contain part or all of the latest version of the system modified by the owner team while the other shared workspace is seen also by the second team and would contain less often updated more stable and mature versions of the system.

In most of the cases when work done in a private workspace is checked into a shared workspace this has no effect on the co-workers at all. Conflicts can occur for two reasons:

- A file of the shared workspace was updated while it was checked out into a private workspace. In this case SNiFF+ issues a warning. To resolve this conflict a developer can merge the two versions with the DiffMerge tool. In the Configuration Manager it is possible to get an overview of all such conflicts.
- An update to a file in the shared workspace leads to a conflict with dependent files in private workspaces. In this case the Configuration Manager, depicted in Figure 3, can be used to investigate the nature of the changes applied to the shared workspace. The Configuration Manager shows the available configurations of a project, its structure, and what specific file-versions are part of the configuration. Differences between configurations on change set and file-level can be shown and differing versions of files can be loaded into the DiffMerge tool.

Policy Enforcement and Process Management

Process-centered software engineering tries to establish a comprehensive theoretical basis for understanding, describing, and enacting specific software processes [Mad91, Ost87].

The basic idea is to describe a specific software process with all the activities and information flows it comprises. The resulting process model is represented as a set of rules that define in which sequence under which preconditions which documents may be modified with which tools. With the same mechanism invariants for the usage of tools are defined [Kai93]. A process model is enacted by executing it with a process engine, which is the hub of every process-centered development environment. The process engine controls the application of all tools.

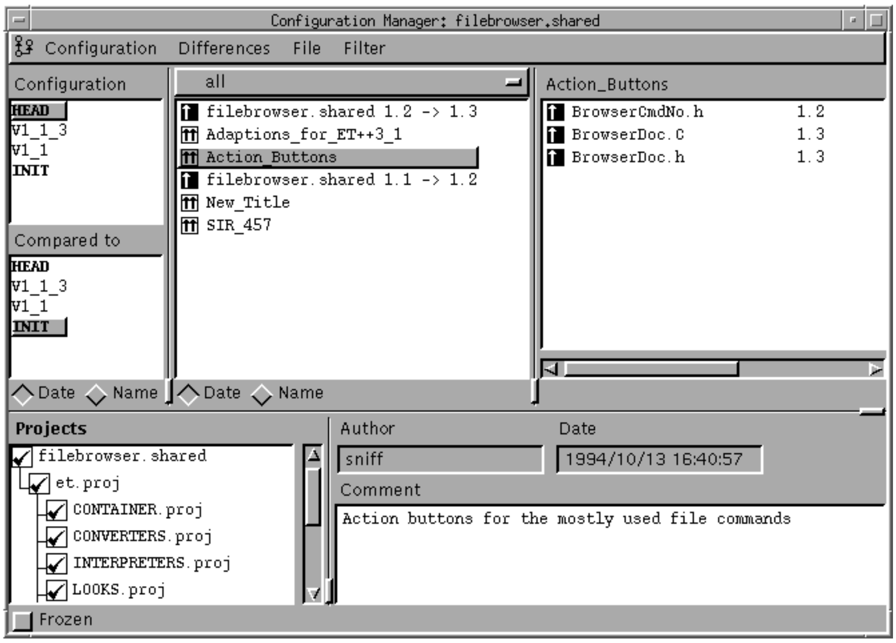

Figure 3. SNiFF+'s Configuration Manager

SNiFF+ does not provide a process engine. Rather, SniffAccess, its access interface, allows the implementation of policy enforcement and therefore, its integration with a process engine. SniffAccess makes it possible to externally drive and control SNiFF+ with two interaction mechanisms:

- requests to SNiFF+
- notifications (or triggers) from SNiFF+

Requests specify actions like opening a project or loading a file into the Editor. Notifications are sent to SniffAccess on events for which a trigger was installed. Notifications have associated parameters supplying further information on notification and can be of two types: pre-action or post-action. On a notification SniffAccess can itself start an action (either call another program or send a request to SNiFF+). Additionally, on pre-action notification (pre-trigger) SniffAccess can either allow the action in SNiFF+ to be completed or not, depending on the result returned to SNiFF+. SniffAccess can be used via an API and via shell commands.

3.2 Informal Cooperation with SNiFF+

SNiFF+ does not provide any specific support for informal cooperation. Nonetheless it eases informal cooperation considerably, by speeding up the process of reading and understanding source code with its browsing facilities. These aspects were discussed in depth in [Bis94a] and will not be covered any further in this paper.

3.3 Realization

Open Adapter Interface to Version Control Systems

SNiFF+ provides abstractions for most of the functionality of classical VCS implementations. For these VCS functions SNiFF+ provides a consistent and easy-to-understand user interface. For example, the SNiFF+ check-in-out operations can be easily mapped to any VCS. The interface consists of about 40 commands that have to be mapped to the commands of a specific VCS. Mappings can be defined and customized in the preferences dialog. Typically they consist of inline awk scripts but they can be implemented with any kind of shell commands. Currently adapters for ClearCase [Atr94], RCS [Tic85], and SCCS [Roc75] are provided.

Configuration Management
Unfortunately CMSs are too different to be integrated with an adapter interface. For this reason a CMS was implemented in SNiFF+, to provide seamless integration with the programming environment. The implementation uses the VCS adapter architecture and works with all VCSs that support symbolic names for versions.

3.4 Related Work

The idea of isolating the work of developers based on workspaces and of supporting the handling of conflicting changes with merge tools is not new. It was for example implemented in SUN's NSE [NSE89]. SNiFF+ distinguishes itself from other such approaches by providing a seamless integration of the workspace support into a programming environment.

Several available programming environments provide some kind of integration with CMSs. This integration usually provides only the possibility to call some commands of the CMS from a menu of the programming environment (e.g., integration between ClearCase and Softbench [Atr94]).

Most programming environments for large scale software development do not provide an explicit project concept, as discussed in [Bis94a]. By definition a CMS has a project model. If a programming environment does not have an explicit project model it can not be tightly integrated with a CMS because that requires the programming environment to transparently control the project model of the CMS.

3.5 State and Further Proceeding

SNiFF+2 is currently in beta test and has been successfully deployed in large software development projects. So far RCS, SCCS, and ClearCase have been successfully integrated via SNiFF+2's open VCS interface and other VCSs are soon to follow.

We intend to take two approaches to evolving SNiFF+, first by improving the support it currently provides and second by studying the commercial feasibility of the approach taken in the Beyond-Sniff project. This second approach would result in splitting SNiFF+ into a set of services and tools running on the Beyond-Sniff platform.

4 Cooperative Software Engineering with Beyond-Sniff

Considerable research for methods and tools supporting cooperation is currently being carried out in the areas of process-centered software engineering and CSCW. We believe that results from both areas should be practically applied as soon as they are available. Unfortunately, the support of informal cooperation on software development is neglected in both software engineering and CSCW.

It is therefore important to do research in the area where CSCW and CSE intersect. We have taken first steps in this direction in developing Beyond-Sniff. This section describes the innovative aspects of Beyond-Sniff that support cooperation. A more detailed discussion can be found in [Bis94b].

4.1 Informal Cooperation with Beyond-Sniff

Motivation for Informal Cooperation

Cooperative software development requires a lot of communication between developers. The increasing popularity of object technology tends even to increase the communication needs. Due to closer cooperation many small pieces of information have to be shared. This is frequently neglected because the conventional approach of putting them into documents with fixed structure does either not make sense or is too expensive. The problem is typically most annoying for information that can not be formalized, such as ideas, short term plans, or information about classes and methods. This kind of information is difficult to store in documents and it is even more difficult to keep it up to date and find it once it is stored.

Developers are often interrupted by requests for some specific information that cannot be provided by someone else. This kind of interruption affects ones concentration and may be counter-productive. Facilities for asynchronous communication may remove this source of productivity-decreasing interruptions.

Developers working in the same building can reduce the information deficit to a certain degree by informally keeping each other up-to-date. This is not the case for teams separated by large distances. In this case there exists either a constant information deficit or a huge communication overhead, both of which reduce overall productivity.

Informal Cooperation with Annotations

We experienced this phenomenon first hand when Sniff was commercialized and certain parts were finished in Zurich while work was already going on in Salzburg. This was a strong motivation to develop an annotation mechanism as part of the Beyond-Sniff platform. This annotation mechanism makes it possible to connect structured information with any kind of artifacts, be it fine-grained artifacts such as classes and instance variables or coarse-grained artifacts such as projects and files.

A Beyond-Sniff annotation has a type that defines which information fields it comprises. This makes it possible to store different kinds of structured information. Frequently used annotation types are, for example, error, documentation, idea, and to-do annotations. Annotated artifacts are visually marked in all Beyond-Sniff tools. One mouse click suffices to display all annotations connected with an artifact. Annotation types can be extended by inheritance and they are defined with a graphical schema editor.

Annotations are centrally stored for every project per site. A developer has either the possibility to access an annotation via artifacts, or he can formulate an OQL [Cat94] query with a query tool to obtain all annotations matching certain conditions. For example, it is possible to obtain all *idea* or *to do* annotations that have been connected to a certain project since a given date. Figure 4 shows the query tool with an evaluated query. Figure 5 shows the screen after selection of a matching annotation: The user sees that the class SymtabItem has annotations. One of them is shown in a separate window.

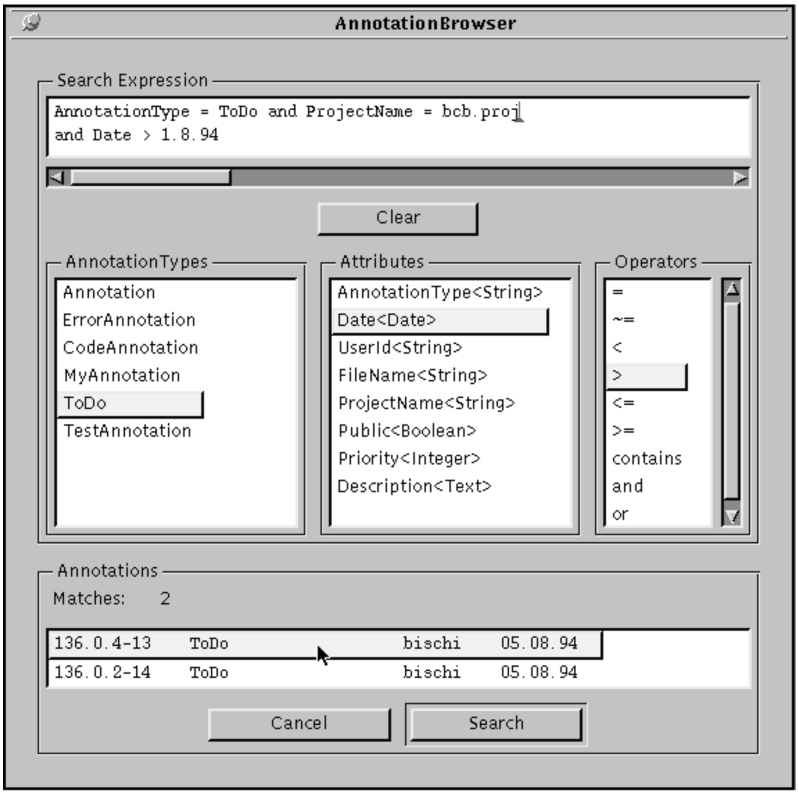

Figure 4. AnnotationBrowser.

There are many situations in which a developer wants to be notified automatically when an annotation is created. For these situations he can define OQL trigger queries that are executed for every newly created or modified annotation. Upon matches the developer is notified either with the Beyond-Sniff notification tool or by e-mail. A typical trigger query selects, for example, all error annotations of projects belonging to a certain developer.

The central storage of annotations together with the query and trigger query mechanisms makes it easy to share information. For instance, there is no need to bother about who might be interested. This reduces the communication overhead by decoupling developers the same way as the Smalltalk change propagation mechanism decouples cooperating objects [Gol89].

Annotations are conceptually a mechanism for undirected communication. Sometimes it is useful to make sure that coworkers read a particular annotation. Beyond-Sniff has two features for that purpose. First, an annotation can be specifically addressed to developers. Second, the creator can specify that he wants to be automatically notified whenever an annotation is opened.

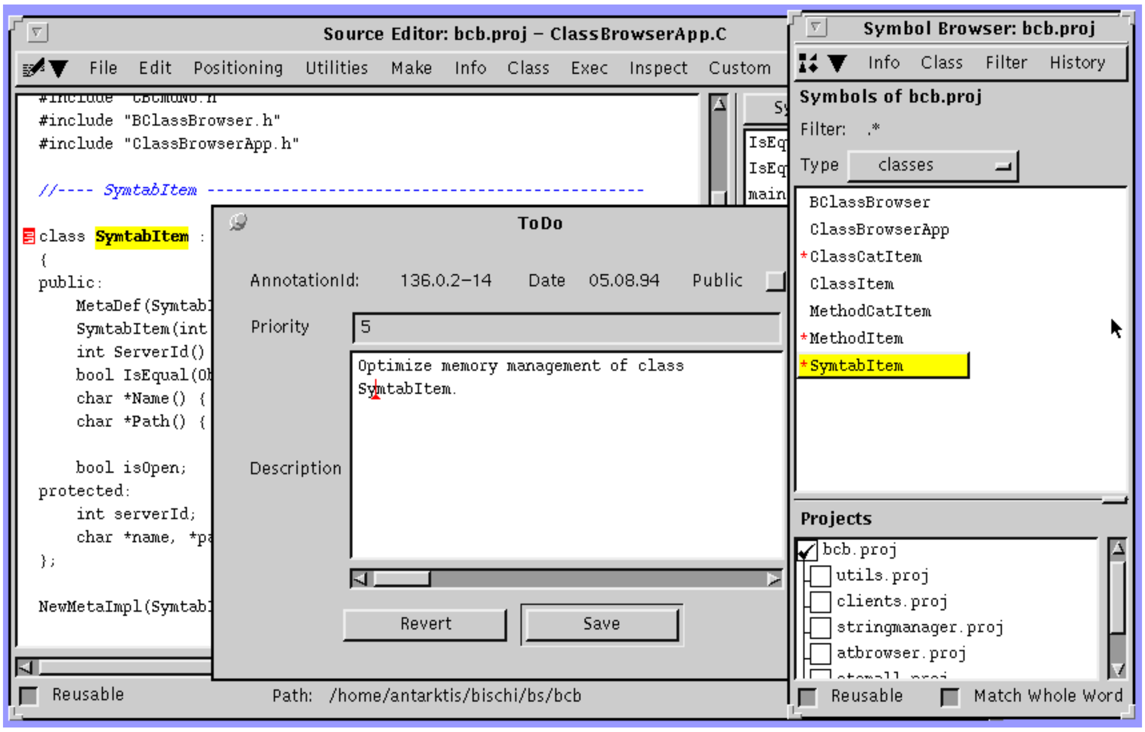

Figure 5. Annotation aware Editor and SymbolBrowser with an annotation.

Annotations can not only be inserted manually but may also be generated and inserted by tools which make certain events visible in the annotation universe. Typical of such cases are the check-in of a modified file into the version control system and the modification of a project structure.

Beyond-Sniff's annotations are a hybrid approach to information management. On the one hand they can be used together with links to organize information as a hypertext. On the other hand they are typed, they are centrally stored and they can be retrieved with a query mechanism. This integration of hypertext and database approach makes it easy store structured information, to connect it with any kind of artifact and to find it in different ways.

In this paper annotations and links were only discussed in the context of cooperative software development but they can fulfill a large number of information management and communication needs.

4.2 Policy-Driven Cooperation with Beyond-Sniff

Beyond-Sniff supports the conventional configuration management approach, i.e., optimistic and pessimistic coordination, but it goes further by supporting optimistic coordination over low bandwidths and by providing extensive support for merging versions of entire projects. A detailed discussion of the first topic goes beyond the scope of this paper. The rest of this section gives a brief overview of Beyond-Sniff's merging support.

Projects define the level of granularity on which developers are cooperating with Beyond-Sniff. A project consists of all artifacts which are relevant for the development of a certain software system. Projects are explicitly defined and can be structured in a tree of subprojects.

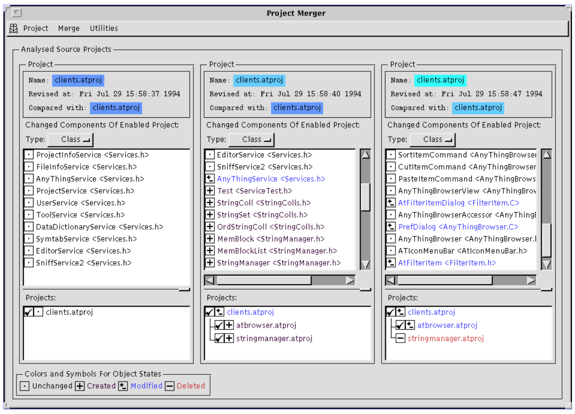

Figure 6. TurboMixer visualizing differences in structure and classes

The merging of projects is a central task of any kind of optimistic cooperation. Beyond-Sniff's TurboMixer provides support for comparing and merging projects. Differences can be browsed at the project level and on varying levels of symbolic and textual detail. It visualizes differences with colors and pictograms.

Figure 6 shows the TurboMixer in comparing three consecutive versions of a project, ordered by increasing age from left to right. The structure of every working project is visualized as a tree and the classes are listed above them. New, changed, and deleted elements are visualized the same way in the tree and in the list. The semantics of the pictograms and colors is described in the lower left corner.

4.3 Realization

Beyond-Sniff consists of cooperating tools running on various workstations. The biggest challenge in implementing such a distributed system is tool integration, i.e., the integration of different kinds of tools and services in a way that they can cooperate as seamlessly as possible (from the user's point of view).

Beyond-Sniff consists of a platform for tool integration and an extensible number of services and tools. Figure 7 provides an architecture overview that also shows a number of important services and tools. It is beyond the scope of this paper to give a comprehensive overview of the approaches taken in realizing this architecture. Further information can be found in [Bis94b].

4.4 Related Work

In the area of CSCW there is a large number of published synchronous approaches such as synchronous editing of documents and video conferencing but there are no approaches that have achieved a relevant level of practical application besides video conferencing systems. Dewan proposes in [Dew93] applying different synchronous approaches such as synchronous editing and debugging to cooperative software engineering. This approach, however, does not address the relevant problems of cooperative software engineering discussed in Section 2.

Some programming environments such as Cadillac [Gab90] and Field [Rei90] already incorporate annotations. In contrast to our approach their annotation concepts are simplistic means for connecting some information with source code. They are tool-specific and cover only a small part of the artifacts. Moreover, these environments are aimed at single developers.

We do not know about tools similar to TurboMixer. Only Grass describes in [Gra92] similar concepts and ideas.

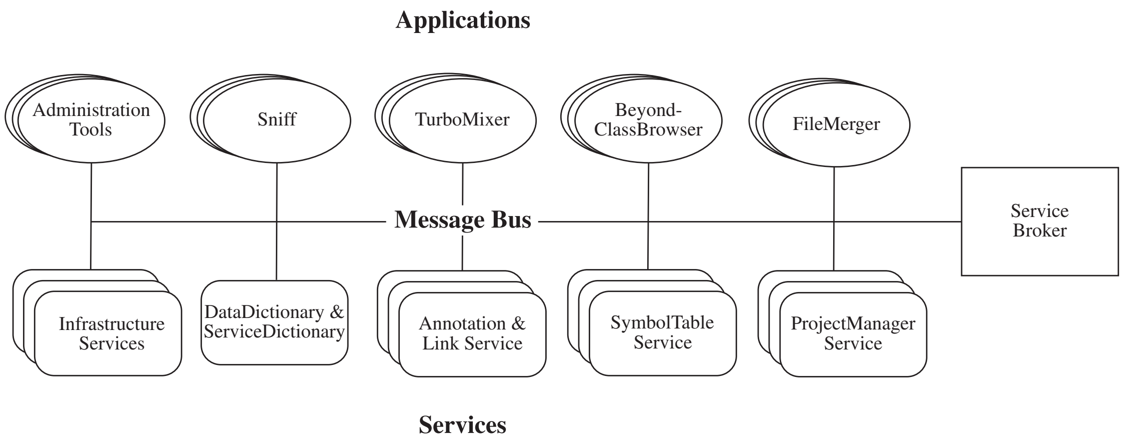

Figure 7. Architecture overview.

4.5 State and Further Proceeding

We implemented the first Beyond-Sniff prototype in 1993 to validate the feasibility of our approach. Based on this experience we rewrote large parts of the infrastructure, which is now mature enough that Beyond-Sniff is used on one host by multiple developers cooperatively. It is used for its own evolution.

Work is going on in different areas. By working with Beyond-Sniff on Beyond-Sniff we are currently evaluating how useful the support for cooperative software engineering is in practice. This will probably trigger several iterative refinement processes. The message bus will be replaced to make it possible to use Beyond-Sniff cooperatively on several hosts. The tools running on Beyond-Sniff will be evolved.

Implementation on the support for cooperation over low bandwidths has not started yet. Its technical foundation, however, has been laid by implementing GTS [Maf94], a transport layer independent, reliable multicast system.[3]

5 Summary

Software systems are most often developed in teams. Teamwork implies cooperation and therefore also coordination needs. We identified two forms of coordination, policy-driven and informal coordination. Conventional configuration management addresses the more obvious need for policy-driven coordination. Pessimistic or optimistic approaches can be distinguished. Informal coordination is neither addressed in theory nor supported by tools in practice.

Based on the results of the Sniff project two follow-up projects were started. For both of them it was a central goal to implement considerable support for cooperative software engineering. In the commercial SNiFF+ project an evolutionary approach was taken which results in a seamless support for configuration management and version control and therefore for policy-driven coordination.

In the research-oriented Beyond-Sniff project a search for new approaches is going on. The current results are the annotation mechanism which makes it possible to connect structured information with any kind of artifacts, be it fine-grained artifacts such as classes and instance variables or coarse-grained artifacts such as projects and files. This information can be retrieved flexibly via queries, triggers, and all annotation-sensitive tools. The second result in the area of cooperative software engineering is extensive support for structural, symbolic, and textual comparing and merging of entire projects.

Our search for novel approaches to supporting cooperative software engineering is not yet finished, but we made some important steps.

Acknowledgements

We would like to thank our colleagues at TakeFive and in the Beyond-Sniff project at UBILAB for their involvement in the development of SNiFF+ and Beyond-Sniff. Special thanks go to Thomas Kofler and Bruno Schäffer who gave us a lot of comments on how to improve this paper. Finally we would like to thank Union Bank of Switzerland for funding a laboratory where research projects such as Beyond-Sniff can be carried out.

---

[3] GTS was developed in a cooperation between University of Zurich, UBS/UBILAB, and Siemens Munich. The project was sponsored by the Swiss Federal Commission for the Advancement of Scientific Research (KWF).

References

[Atr94] Atria Software, Inc.: ClearCase User's Manual, 4000-011-A, May 1994

[Bis92] Bischofberger WR: Sniff - A Pragmatic Approach to a C++ Programming Environment. in Proceedings of the USENIX C++ Conference, Portland, Oregon, Aug. 1992

[Bis94a] Bischofberger WR, Kofler T, Schäffer B: Object-Oriented Programming Environments: Requirements and Approaches. in Software – Concepts and Tools , Vol. 15 No. 2, Springer-Verlag, 1994

[Bis94b] Bischofberger WR, Kofler T, Mätzel K-U, Schäffer B: Computer Supported Cooperative Software Engineering with Beyond-Sniff. UBILAB Technical Report 94.9.1, 1994

[Cat94] Cattell RGG (ed.): The Object Database Standard: ODMG-93; Morgan Kaufman Publishers, 1994

[Dew93] Dewan P, Riedl J: Toward Computer Supported Concurrent Software Engineering. IEEE Computer, January 1993

[Gab90] Gabriel R. P. et al.: Foundation for a C++ Programming Environment. In Proceedings of C++ at Work-90, Secaucus, New Jersey, 1990

[Gol89] Goldberg A, Robson D: Smalltalk-80–The Language; Addison-Wesley 1989

[Gra92] Grass JE: Cdiff: a Syntax Directed Differencer for C++ Programs. in Procs. of the USENIX C++ Conference, Portland, Oregon, Aug. 1992

[Kai93] Kaiser GE, Popovich SS, Ben-Shaul IZ: A Bi-Level Language for Software Process Modeling. in Procs. of the 15th ICSE, 1993

[Mad91] Madhavji NH: The Process Cycle. in Software Engineering Journal, September 1991

[Maf94] Maffeis S, Bischofberger WR, Maetzel KU: GTS: A Generic Multicast Transport Service. UBILAB Technical Report 94.6.1, 1994

[Mal94] MaloneTW, Crowston K: The Interdisciplinary Study of Coordination. in ACM Computing Surveys, Vol. 26, No. 1, March 1994

[NSE89] The Network Software Environment. Sun Technical Report, Part No. 800-3295-10, Sun Microsystems, 1989

[Ost87] Osterweil L: Software Processes are Software too. in Procs. of the 9th ICSE, 19987

[Rei90] Reiss SP: Interacting with the FIELD environment. Software – Practice and Experience, Vol. 20, S1, 1990

[Roc75] Rochkind M: The Source Code Control System (SCCS). IEEE Transactions on Software Engineering, Vol. 1, No. 4, 1975

[Sch93] Schefstöm D., van den Broek G.: Tool Integration–Environments and Frameworks; John Wiley & Sons, 1993

[Tic85] Tichy W: RCS – A System for Version Control . Software – Practice and Experience, Vol. 15, No.7, July 1985

[Tic88] Tichy W: Tools for Configuration Management. Proceedings of the International Workshop on Software Version and Configuration Control (Jan. 27 - 29, 1988 Grassau), Teubner, 1988