Perspectives on Software Development Environments

Vassilis Prevelakis
Dennis Tsichritzis

Centre Universitaire d'Informatique (CUI)
Universite de Geneve
24 rue du General-Dufour, Geneva 1211,
Switzerland.

Abstract. Using graph-like structures to store and organise ideas, concepts and programs in a Software Development Environment is not new. This approach, however, has two drawbacks: the rigidity and large size of the resulting graph. Users have difficulty managing change in the network and as the information piles up, they have trouble finding their way in the graph. In this paper we describe an organisation based on *perspectives* that attempts to alleviate these problems. Perspectives provide a uniform model for views, versions and contexts - *and* can be composed via *perspective operations*. After a brief introduction where we examine the problem, we give a more rigorous description of the model behind perspectives and the operations that can be performed on them. Finally, we outline a prototype implementation built to demonstrate the power and flexibility of our model.

1 Introduction

Over the past two decades, rapid advances in the capabilities of computer hardware have resulted in the construction of software systems of ever increasing complexity. This in turn has resulted in a need for sophisticated Software Development Environments (SDEs). Such environments support the design team as it progresses along the various stages of the product development process. This activity is characterized by the generation of a multitude of project products which include specifications, documentation, source code, messages between members of the team, discussions on various aspects of the design, tradeoffs, etc [1]. The great difficulty is in keeping all this information together in a single integrated structure while allowing for the flexible investigation of alternative designs. By having a SDE that manages the generation, maintenance and control of these products we reduce duplication of effort and lack of coordination [2]. The alternative is systems where the documentation does not keep up with the programming, eventually becoming irrelevant and leading to situations where the function of various modules or subsystems can only be determined by examining the code [3].

The heart of a SDE is the Object† Management System (OMS). This is responsible for the administration of all the artifacts stored in the system. The OMS can be simply a front-end to the standard file system (e.g the SCCS [4] and RCS [5]), or a DBMS. Most systems have their own custom-built DBMS (Adele [6], Cactis [7], etc.) although there are also systems based on commercially

† The term *object* is used in a generic sense; the precise definition depends on the OMS used and possibly the schema selected by the programmer.

available DMBSs (e.g. VBASE [8]). Finally, a third category consists of systems that alter the appearence of the file system according to the request issued by the user (e.g. the SUN NSE [9], the 3-D File System [10], Gypsy [11] etc.).

Regardless of the actual implementation, all OMSs must satisfy a number of requirements in order to be effective as part of a SDE. These are the management of composite objects, versioning, configuration management and support for parallel activities.

It is often necessary [12] to view objects in an OMS as composite, i.e. comprising a number of interrelated objects, or to decompose an object into its constituent parts. Composite objects allow the user of the OMS to maintain an uncluttered global view of the objects in a given application by grouping them in composite objects, thus reducing the total number of artifacts visible at one time. An example of such *aggregation hierarchies* may be found in Rigi [13].

Since, the primary activity in a SDE is editing and refining of ideas, the system must provide adequate support for versioning. Versions can be used to track change in the system; thus, in cases where modifications introduce bugs or interfere with other parts of the software it is important to be able to know exactly what has changed and at what stage in the development process. Users should be able to use the versioning facilities to reverse the effects of actions such as deleting or modifying an object.

Another requirement is configuration management, that is the ability to work on *variants* of a basic design. For example, when dealing with a large software product the system must be able to manage variants customized for different target environments (such as different operating system, hardware etc.). In this way we keep only one copy of the common code and related information, while the parts dealing with the customization and the platform specific details (i.e. the variants) are stored separately. Given a specific configuration, the OMS can extract the required information (both common and variant) thus constructing one complete system.

Current software development projects are more often than not relying on teams of people rather than a single individual [14]. Therefore, the OMS must provide the facilities that enable all members of the team to work without interfering with each other. This implies the ability to create private workspaces and work in isolation while retaining the option of merging this work with that of the other members of the team. Individuals should be able to create their own customized views of the system [15] and thus experiment with alternative designs.

Another aspect of the OMS is the schema used to represent the objects and their attributes. Here we will work within the framework of an entity-relationship (ER) model as it is quite popular (e.g. used in PCTE [16], Rigi [13], etc.) and because relationships can be used to depict inter-object references.

In this paper we present a model for organising ER networks that attempts to comply with the above mentioned requirements. The model is based on *perspectives* - graph structures which can be combined and operated on in various ways. In the next section we provide formal definitions for perspectives and perspective operations such as addition, selection, and projection. Section 3 discusses the role of perspectives in an OMS. Finally, section 4 outlines the architecture and implementation of an OMS based on perspectives.

2 Perspectives

Consider a universal name space $\{n\}$ of node identifiers. A *node* in our model consists of a pair $[n, C_n]$, where:

- n is a node identifier from $\{n\}$, and
- C_n denotes the contents of the node with identifier n.

We do not make any assumptions about the particular structure of the contents of a node, except that it can include pointers to other nodes (in the form of node identifiers). The rest of its contents can be strings, bitmaps, attribute values, etc.

A *perspective* P_i is a set of nodes over $\{n\}$ with the links attached to the nodes as presented by the node pointers. Within a given perspective node identifiers are unique (i.e. there can be no two nodes with the same node identifier). A perspective usually defines a semantic context, for instance, a version, view, etc. Note that the links are anchored inside the originating node but point to the outside of the destination node. Pointers can be dangling, pointing to nodes absent from the perspective; in this case we say that the link points to a *missing* node. A node may also exist but may be completely empty. We will denote by $[n, \varepsilon]$ a *missing* node and by $[n, \emptyset]$ an *empty* node. The difference between the two will become clear in the following section.

Perspectives can share node identifiers. For example $[n, C_n^1]$ can be a node in perspective P_1 and $[n, C_n^2]$ can be a node in perspective P_2. Nodes in different perspectives which share a common identifier will be called *compatible*. We expect, of course, that there should be a good reason for selecting the same identifier for the nodes. For example, $[n, C_n^1]$ can be the source code of a program and $[n, C_n^2]$ the documentation of the same program. In many cases compatible nodes with the same identifiers would refer to the same real world "entity". In terms of notation we will use C_n^i to denote the contents of the node n in perspective P_i. We will drop the superscript if there is no ambiguity about the perspective.

2.1 Operations on perspectives

What differentiates our model from existing ones (like the Intermedia *webs* [17]) is the ability to combine perspectives by means of operations. Consider any operation (unary, binary, etc.) which is defined in terms of node contents. For example:

$$u(C_n^i) \to C$$

$$r(C_n^i, C_n^j) \to C$$

We do not make any assumption about the operations except that they are well defined and give as a result something which can be interpreted as node contents. Unary operations are intended for modeling updates on the contents of a node such as adding pointers, altering time-stamps, changing attribute values, etc. Binary operations are useful when operating on compatible nodes of different perspectives, for instance, to concatenate or merge the contents of two nodes. That is, operations do not relate nodes from the same perspective but compatible nodes from across perspectives. In the rest of this paper we concentrate mainly on binary operations (r) although our discussion can be expanded to cover other types of operations as well.

For each operation r on node contents there exists a corresponding operation R on compatible nodes, such that:

$$R([n, C_n^i], [n, C_n^j]) \text{ is defined as } [n, r(C_n^i, C_n^j)]$$

When defining an operation, special care should be taken for the special cases involving missing or empty nodes. For example, $r(\varepsilon, C_n^j)$ and $r(\emptyset, C_n^j)$ should be defined.

We can now proceed to define an operation on two perspectives. Let P_1 and P_2 be two perspectives defined on the same node identifier space. Any operation r on node contents has a corresponding operation R on the two perspectives defined as follows:

Let N_1 and N_2 be the sets of node identifiers present in P_1 and P_2 respectively. The operation $R(P_1, P_2)$ is defined as a perspective with the following nodes:

- For a node n in $N_1 \cap N_2$ the node in the resulting perspective is $[n, r(C_n^1, C_n^2)]$.
- For a node n in $N_1 - (N_1 \cap N_2)$ the node in the resulting perspective is $[n, r(C_n^1, \varepsilon)]$.
- For a node n in $N_2 - (N_1 \cap N_2)$ the node in the resulting perspective is $[n, r(\varepsilon, C_n^2)]$.

To understand operations on two perspectives it is helpful to think of them in terms of two steps, even if the algorithms to perform them may work in a different way:

Step 1: We *align* the nodes of the perspectives according to their node identifiers.

Step 2: We perform the operation on each pair of aligned nodes separately.

As examples consider three perspective operations: an *additive operation*, a *selective operation* and a *projection operation*. An additive operation accumulates the contents of the aligned nodes. A selective operation isolates the contents of a particular perspective for each node. Finally, a projection operation concentrates on only a subset of the nodes.

Consider two perspectives P_1 and P_2. The addition of two perspectives, called an *additive overlay*, is represented by $P_1 + P_2$ and defined as:

- for the nodes in $N_1 \cap N_2$ (common nodes) the contents of both the P_1 node and the P_2 node are retained.
- for the nodes in $N_1 - (N_1 \cap N_2)$ the contents of the P_1 node are retained along with an indication that P_2 had a missing node.
- for the nodes in $N_2 - (N_1 \cap N_2)$ the contents of the P_2 node are retained along with an indication that P_1 had a missing node.

Notice that the operation of additive overlay is commutative and associative.

As a selective operation consider an operation called a *masking overlay* of a perspective P_2 on a perspective P_1. A masking overlay is represented by $P_1 \circ P_2$ and defined as:

- for the nodes in $N_1 \cap N_2$ only the contents of the P_2 node are retained.

- the nodes in $N_1 - (N_1 \cap N_2)$ and in $N_2 - (N_1 \cap N_2)$ are left unchanged.

Note that if P_2 has a node which is empty it may potentially erase information, not only substitute it on P_1. In this way we can completely erase pointers or attribute values from P_1. However, even if the contents of a node on P_1 are deleted, the node itself will still be present. The existence of empty nodes is quite significant because they act like "black holes": they exist but cannot be seen by the user. Thus, an empty node in the topmost perspective in a masking overlay will make all the instances of that node in the other perspectives invisible to the user. In this way we can remove nodes from the user view.

Notice that the operation of the masking overlay is associative, but not commutative. Notice also that we can distribute the operations, for example:

$$((P_1 + P_2) \circ P_3) = (P_1 \circ P_3) + (P_2 \circ P_3)$$

There are two special perspectives: the first is 0 which is like a completely clear transparency (all nodes are missing), and the other is 1, where *all* possible nodes are empty. The perspective 1_S has S nodes empty, where S is a set of node names. The 1 perspectives are like opaque transparencies that mask everything.

A *projection operation* $|P|_S$ can be defined as retaining in a perspective P only a subset of its nodes S. The projection operation is very handy when we need to concentrate on a few nodes of a perspective. The projection operation can be expressed by masking overlays in the following manner:

Let perspective P_{mask} be defined as:

$$P_{mask} = \{ [n, \emptyset] \text{ where } n \in (N_1 - (N_1 \cap S)) \}$$

Then,

$$|P|_S = P \circ P_{mask}$$

In a similar manner we can express a masking overlay through projections and additive overlays. Note, finally, that an additive overlay cannot be expressed through projections and masking overlays because an additive overlay is the only operation which retains all information in the nodes.

Additive and masking overlays are examples of a much more general form of the binary operation $P_1 \textcircled{r} P_2$ where r is some kind of rule for selecting or combining the node contents. For example, we can choose the most recent contents according to a time-stamp or the most relevant contents according to their source.

2.2 Perspective expressions

Consider a set of independent perspectives $P_1, \ldots, P_k$. By using the operations $+$, $\circ$ or any other operation we can obtain expressions of these perspectives. An expression defines a perspective which combines the contents of the compatible nodes of the different perspectives.

An expression can be evaluated by carrying out the operations on each set of compatible nodes separately. Perspective operations and expressions, although they are defined as set operations, can be implemented as a sequence of operations on individual nodes. In addition, the evaluation of an expression on perspectives can be performed in parallel for each set of compatible nodes.

Any expression $E(P_1, \ldots, P_k)$ can be evaluated as an operation on the contents of the nodes. The additive expression $P_1 + \cdots + P_k$ keeps all the node contents properly aligned. Since the additive operation does not lose information, it follows that from $P_1 + \cdots + P_k$ any arbitrary expression can be computed without the need to retain any additional information.

The notion of change in such an environment is rather analogous to updates in base and derived relations in relational database systems. We can thus have two kinds of perspectives: *base* perspectives which usually contain entire networks and can thus stand on their own, and *differential* perspectives that contain only changes (see figure 1).

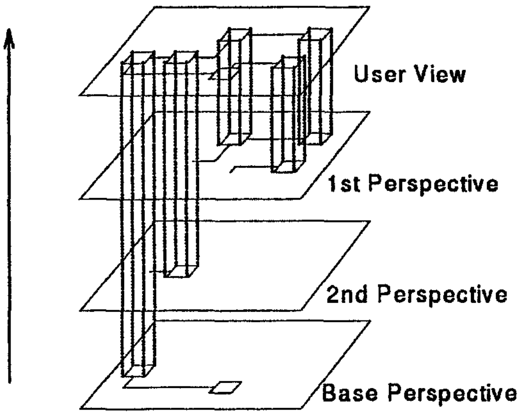

Figure 1: An overlay of perspectives.

Consider a perspective defined by an expression on a set of base perspectives:

$$P = E(P_1, \ldots, P_k)$$

Suppose a user viewing P performs a single change u, or for that matter a group of changes in one or more nodes. There are many possible interpretations of such a change.

- The change u may be temporary and associated only with the current session.
- The change u should be part of a newly defined differential perspective P' which is kept and can be viewed either separately or in combination with other perspectives.
- The change u redefines the perspective P. P overrides its definition in terms of the expression and in this manner becomes a base perspective.

We also have to investigate how changes can possibly propagate forward or backward. A change in a base perspective can be defined as propagating forward to any expression involving the perspective. A change in a derived perspective P can also (if possible) be defined to propagate backwards to the base perspectives. Note that for such an operation to be feasible we must assume that the change can be attributed in a unique way to the base perspectives. To illustrate the last point consider a program which has been derived by mixing designs, or objects, coming from two different persons. A change in this program may be well defined but there may not be a way, or a unique way, to map the changes to the original pieces.

3 The Role of Perspectives in an Object Management System

Given the definition of perspectives we can see that they can be used as a means of organising the data in an OMS. Users can have their own, private, views of the system, while being able to request different organisations by selecting appropriate perspectives. Users can remove perspectives that contain information that is of no interest to them and concentrate on the perspectives that they need.

Similarly, we can restrict access to certain parts of the system by either refusing access to the perspective containing the information or by forcing the user to use a masking perspective that effectively removes the privileged information from the user view.

We can have a system where all the changes made during a long transaction are stored in one perspective. In this way users can 'back out' changes by simply removing the corresponding perspectives from their overlays.

Another advantage of a perspective-based organisation is that there is no need for locking of nodes or any kind of write protection. Read protection can be implemented via special masking overlays that the users must use to access the system.

In the rest of this section we will present four examples demostrating how perspectives can be used to satisfy the key requirements we identified in the introduction: composite objects, versioning, configuration management and parallel activities.

3.1 Composite Objects

Let us consider a typical C++ class; this consists of at least two parts: the class declaration and the definitions of the class methods (the .h and .c files respectively). So to store this class in an OMS we would need to keep at least these two elements. However, for the class to be useful, more information must be supplied. For example, documentation, specifications, and in many cases a test suite to verify the correct operation of the class after modifications. If the class is under development, we will also need to know the person working on it and the various milestones associated with this task.

Links between the various elements may be used to represent an inheritance hierarchy for the class definitions and cross-references between documentation pages. There may also be other links, for example, from comments within the source code or from the inclusion of definition files that are outside the inheritance hierarchy.

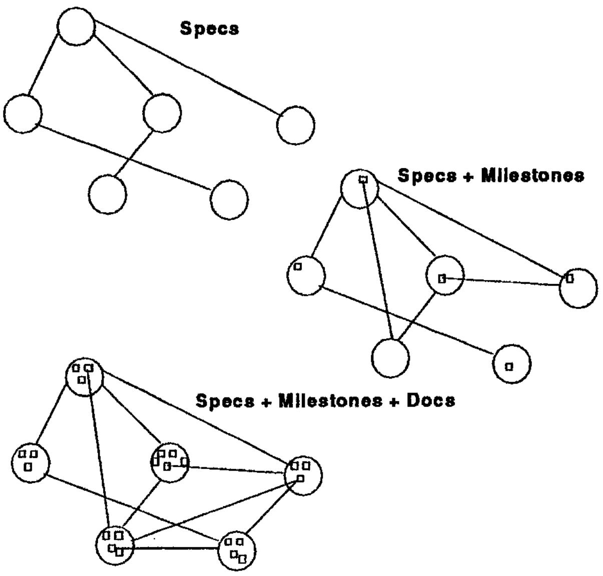

Figure 2: Detail increases as more perspectives are added.

We can either view all these elements as distinct objects and try to deal with them independently or we can construct a *composite* object with all these elements as attributes. We can then assign each type of element to a different perspective and then use the additive operation to gather only the attributes we want. The advantage of this approach is that the selection is performed on *all* the classes of our project.

To avoid getting too many nodes we can restrict our view to only the classes we need. For example:

$$(P_{specs} + P_{milestones} + P_{docs}) \circ P_{myview}$$

gathers the specifications, the milestones and documentation of all the objects in the project and filters them through the P_{myview} perspective. Isolating the components of a single class can be carried out by a projection operation. For example,

$$|P_{specs} + P_{milestones} + P_{docs}|_{myclass}$$

3.2 Versioning

Perspectives are rarely completely independent. We expect that users start with a base perspective P_1 and then do a series of changes and enhancements all of which can be expressed by another perspective P_1'. They can then view only the changes they have made (P_1'), the result of the changes on P_1 ($P_1 \circ P_1'$), or both P_1 and the changes they have made to it ($P_1 + P_1 \circ P_1'$). For example, if P_1 is a software version, P_1' represents the changes, $P_1 \circ P_1'$ is the new version and $P_1 + P_1 \circ P_1'$ is the base version plus all the changes to get to the new version.

We can, thus, establish a chain of perspectives and versions like those defined in figure 3.

Note that each time we only need to keep track of the changes $P_1', P_2' \cdots$. The rest can be computed, including arbitrary expressions on the perspectives.

The same idea can be extended when the resulting perspectives start developing in a different, somehow independent context. Consider, for instance, a base perspective P_1 of application concepts. These concepts can be enhanced, clarified and modified according to source code, documentation, multimedia representation, etc.

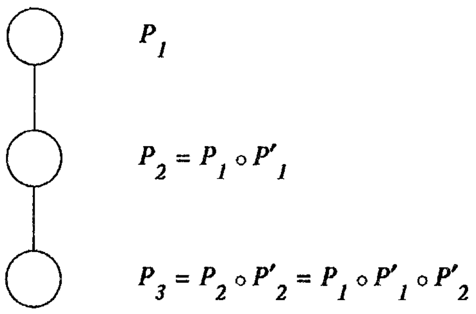

Figure 3: For new versions we need only differential perspectives.

We can provide a hierarchy of perspectives where each branch in the tree from the root defines a series of differential perspectives which can be used to obtain versions. Note that in this way we only store the minimum of information (the differentials). We also have two other advantages. All base information (closer to the root) can become available if needed at all. All information, both old and new, is retained by the system without necessarily cluttering up the graph. Contexts [18] can be viewed individually or in combinations, by obtaining expressions on the available perspectives. The same idea can be extended to lattices of perspectives where the minimum points are independent base perspectives and the rest differential perspectives.

Consider, for instance, the representation of application frameworks. In [19] we see that in developing a software component we start from a template and proceed to fill in the missing information (requirements, design choices etc.) until we end up with a complete framework for our particular application. By keeping copies of the framework during its evolution, we create a tree structure. In this way a developer working on a similar application can follow the evolution of this framework until the point where it diverges from the requirements of the new application. At this stage the developer will create a new branch in the evolution tree. In this way, the reuse of existing frameworks becomes easier.

3.3 Configuration Management

In a similar manner, configuration management operations can be performed with the help of perspectives. We can envisage a situation where there is a base perspective containing the platform-independent parts of the system, while the platform-dependent parts are placed in individual perspectives. So starting from version 3.0 we can construct the version 3 release 2 system for the VAX architecture using the expression $P_{v3.0} \circ P_{patch1} \circ P_{patch2} \circ P_{VAX}$.

3.4 Parallel Activities

Let us take a small workgroup as an example. This team consists of analysts, programmers, technical writers etc. All these people have to work on the same objects and make alterations as they go about their work. The objective of the system is to create a stable environment so that work done by one person is not immediately visible to the others. In this way, changes can be tested before they are released to the rest of the group. When the various members of the team are ready to share their code or other work, the process of integrating the new elements with the existing data should be as painless as possible.

Since perspective construction is based on long transactions, we will have a new perspective only when the programmer is satified with the changes. Even then, the other members of the team may not chose to include this new perspective in their view of the system, thus ignoring the new changes. When everybody is happy with the modification they can all include it in their views. If later-on, there is suspision that the change introduced some bug, the perspective can be removed and the system tested without the offending code.

On the other hand, discussions and comments on various hot topics must be made public immediately so that decisions can be reached and arguments settled. This can be achieved by keeping the transactions short and having the system automatically update all the user's views with the new transactions. Clearly this environment cannot rival a conferencing system, but it can support electronic mail discussions that normally take place within groups.

4 Prototype system

We have implemented a prototype system supporting perspectives with the following objectives:

- demonstrate the power and flexibility of perspectives.
- test algorithms for the construction and storage of perspectives.
- provide a test-bed where we can evaluate the existing operations on perspectives and experiment with new ones.

In order to satisfy these requirements we separated our prototype into two parts: one responsible for the data management operations (*Object Management System*) and another for the interaction with the user (*front-end*). The two parts communicate with each other via a special communication protocol called the *application interface*.

The OMS holds all the perspectives known to the system and is able to provide information about them. It also handles the construction of new perspectives by executing requests containing perspective expressions.

The front-end is an application (e.g. software development environment, hypertext etc.) that acts as a client to the OMS. The front-end then converts application-oriented operations into requests that can be handled by the OMS. For example, in the case of a configuration management system, the front-end will construct a perspective corresponding to the desired system configuration.

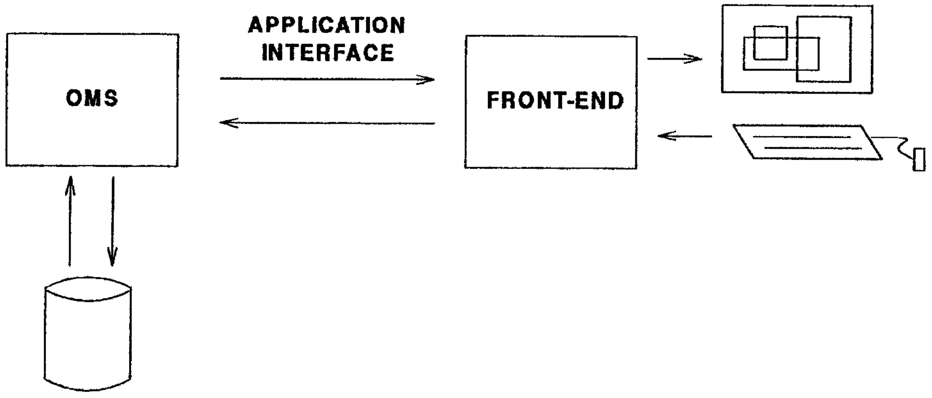

Figure 4: Major components of the prototype system.

The application interface handles the interaction between the two system components. The key requirement of the application interface is to allow application-independent manipulation of data. In this way information from different front-ends may be kept in the OMS.

The communication between the front-end and the OMS may be viewed as a long transaction involving three stages:

- *establish a perspective:* the front-end has to issue a request that contains an expression yielding a perspective that will be the basis of all future operations. Possible perspective expressions range from the trivial P_k that simply selects perspective k, to full expressions like the ones described in section 2.
- *node operations:* during this stage the front-end issues requests for the retrieval or storage of entire nodes chosen from the set of nodes defined in the current working perspective defined above. The front-end may also ask for directory type information on nodes (e.g. size, owner, type etc.).
- *termination stage:* if changes have been made during the previous stage, the front-end either commits the transaction by creating a new perspective, or throws away all the modifications by aborting the transaction.

The design of the OMS was geared towards the support of perspectives in read-only media (e.g. write protected Unix filesystems, perspectives stored on CD-ROM, etc.). Another requirement was that we wanted to associate application-specific data with each node. The application interface provides the OMS with information about the application so that the OMS can retrieve the data specific to that application. For example, if a perspective is used by many applications, there will be different application-specific data attached to it. The application interface gives the OMS the application identifier so that the OMS can send only the data relevant to the current application. The same procedure applies to user customisations since the application interface supplies the user name as well.

To be able to satisfy these requirements we chose an organisation where nodes and perspectives are kept separate and each node may exist in more than one file. Each file contains a *volume table of contents* (VTOC) structure in the beginning identifying what perspectives or nodes are stored in the file. The system need only read the VTOCs to construct its in-memory list of existing perspectives. The in-memory list also contains entries for each available node with information about the files where the various versions of the node contents are stored. It is at this stage that the OMS learns about any application-specific structures that are attached to the node. In this way when a retrieval request is made for a given node, the OMS first checks to see if the application expects some custom information along with the main node data. If this is so and the particular version of the node has this data, then the system sends it, otherwise the system has the option of either sending default values, or a code indicating that application-specific data is not available for that version of the node. In either case it is up to the front-end to sort things out.

Operations on perspectives can be performed by carrying out the operation on the nodes of the perspectives involved in the operation. It is, therefore, clear that to define a new operation on perspectives we only need to specify what happens when the operation is applied to the nodes.

The general form of the procedure defining the new operation takes the contents of two aligned nodes as arguments and returns a third node which is the result of the operation. The procedure is responsible for creating the new node (if necessary) and copying the information from the other two nodes. Unary operations can be defined in a similar manner. The system has a special node identifier that is given to missing nodes in the corresponding perspective. Again, it is up to the user-supplied procedure to determine what happens in this case.

The front-end is just an application that acts as a client to the OMS. It uses transactions involving application interface requests for its communication with the OMS. For this prototype we have chosen a hypertext application as a front-end. The emphasis for this particular hypertext front-end is version management. This choice of application domain is particularly suitable as a demonstration of the power of perspectives as it combines an application (hypertext) where the network of the nodes is clearly visible to the user and a versioning system where the version layers (or planes) can easily be expressed as perspectives. Our choice of version management for hypertext was also influenced by the fact that current hypertext systems lack effective versioning support [20]. We thus provide both a demonstration of our system while alleviating a well known problem. Work is also under way in the integration of perspectives in the XOS object server currently under development at the CUI [21].

5 Conclusion, future plans

In this paper we have presented perspectives as a mechanism for organising and manipulating groups of nodes and links in a hypertext network. The description was in three stages: presentation of the model, a discussion on how the model copes with the OMS requirements, and a prototype system that is used as a testbed for the evaluation of perspectives.

We have outlined the use of perspectives:

- to provide alternative configurations for an ER network,
- to serve as a container of changes in a versioning system,
- to provide different levels of detail or different aspects of the same node.

Given these features we believe that perspectives can alleviate some of the problems associated with versioning, user views of the OMS contents and the management of composite objects.

An important area where further work is needed is the evaluation of the perspective expressions. Since the evaluation of each node can proceed independently of the other nodes, there is clear potential for both parallel processing and "lazy evaluation" (i.e. calculating only the nodes that are required by the front-end). In the current prototype, however, a request for the available nodes in a perspective will result in the complete evaluation of the perspective expression so that the list of the nodes present in the resulting perspective can be returned. In the cases where the front-end does not issue this request, the OMS is able to evaluate the nodes as they are requested.

Another issue is the "alignment" of the nodes. Given a perspective expression $P_1 \circledcirc P_2$, there are many ways that the system can select how the nodes from P_1 will be matched (or aligned) with the nodes of P_2 so that the operation can be carried out. In our system we have chosen the conservative approach of using the node identifiers as a criterion for the alignment. A more user-oriented method of alignment may use the position in the graph, which is closer to the real world analogy of transparencies. The latter approach however must rely heavily on various heuristics that allow the system to make "intelligent" decisions (e.g. how close should two nodes be to be considered aligned).

Although allowing users to reconfigure their network is quite useful the trend is to allow the system to do the reconfiguration on its own. On some systems the reconfiguration is carried out as a response to a query by the user [22], while on others the user supplies (or selects) the criterion and the system constructs a new perspective that arranges the nodes accordingly [23].

Our current plans include the addition of extra services to our prototype system and the implementation of multiple front-ends so that we can demonstrate the flexibility of perspectives in a number of application domains. We also hope that our experience with this prototype will help us refine our model.

References

[1] Ira P. Goldstein and Daniel B. Bobrow, "Descriptions for a Programming Environment," *Proceedings of the 1st Annual Conference of the National Association on Artificial Inteligence,* pp. 187-194, Stanford, CA (Aug 1980).

[2] Maria H. Penedo and Don E. Stuckle, "PMDB - A Project Master Database for Software Engineering Environments," *Proceedings ICSE,* pp. 150-157 (1985).

[3] Robert N. Britcher and James J. Craig, "Using Modern Design Practices to Upgrade Aging Software Systems," *IEEE Software,* pp. 16-24 (May 1986).

[4] M. J. Rochkind, "The Source Code Control System," *IEEE Transactions on Software Engineering,* 1, 4, pp. 364-370 (Dec 1975).

[5] Walter F. Tichy, "RCS: A System for Version Control," *SP&E,* 15 (1985).

[6] N. Belkhatir and J. Estublier, "Experience with a Database of Programs," *Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, ACM SIGPLAN Notices,* 22, 1, pp. 84-91 (Jan 1987).

[7] Scott E. Hudson and Roger King, "Object Oriented Database Support for Software Environments," *Proceedings of the ACM*

SIGMOD Annual Conference on Management of Data, pp. 491-503, San Francisco, CA (May 1987).

[8] Timothy Andrews and Craig Harris, "Combining Language and Database Advances in an Object-Oriented Development Environment," *Proceedings OOPSLA'87,* pp. 430-440 (Oct 1987).

[9] Terrence C. Miller, "A Schema for Configuration Management," *Proceedings of the 2nd International Workshop on Software Configuration Management,* pp. 26-29, Princeton, New Jersey (Oct 1989).

[10] David G. Korn and Eduardo Krell, "The 3-D File System," *USENIX Summer'89,* pp. 147-156 (1989).

[11] Ellis S. Cohen, Dilip A. Soni, Raimund Gluecker, William M. Hasling, Robert W. Schwanke, and Michael E. Wagner, "Version Management in Gypsy," *Proceedings of the Symposium on Practical Software Development Environments, ACM Software Engineering Notes,* 13, 5, pp. 201-215 (Nov 1988).

[12] Commission of the European Community, "Requirements for Software Engineering Databases" (June 1983).

[13] Hausi A. Müller and Karl Klashinski, "Rigi: A System for Programming-in-the-large," *Proceedings of the 10th International Conference on Software Engineering,* pp. 80-86, Singapore (April 1988).

[14] S.J. Gibbs, "CSCW and Software Engineering" in *Object Oriented Development,* ed. Dennis Tsichritzis, pp. 31-40, Centre Universitaire d'Informatique (CUI), Geneva, Switzerland (July 1989).

[15] Ira P. Goldstein and Daniel B. Bobrow, "Browsing in a Programming Environment," *Proceedings of the 14th Hawaii International Conference on System Science* (Jan 1981).

[16] Gerard Boudier, Ferdinando Gallo, Regis Minot, and Ian Thomas, "An Overview of PCTE and PCTE+," *Proceedings of the Symposium on Practical Software Development Environments, ACM Software Engineering Notes,* 13, 5, pp. 248-257 (Nov 1988).

[17] Nancy Garrett, Karen Smith, and Norman Meyrowitz, "Intermedia: Issues Stategies and Tactics in the Design of a Hypermedia Document System," *Proceedings of the Conference on Computer Supported Cooperative Work (CSCW),* pp. 163-174, Austin, Texas (Dec 1986).

[18] Simon Gibbs, Dennis Tsichritzis, Eduardo Casais, Oscar Nierstrasz, and Xavier Pintado, "Class Management for Software Communities," *Communications of the ACM,* 33, 9, pp. 90-103 (Sep 1990).

[19] Rebecca J. Wirfs-Brock and Ralph Johnson, "Surveying Current Research in Object-Oriented Design," *Communications of the ACM,* 33, 9, pp. 104-124 (Sep 1990).

[20] Jeff Conklin, "Issues in the Design and Application of Hypermedia Systems," *Conference on Human Factors in Computing Systems (CHI'90),* Seattle, Washington (April 1990).

[21] Simon Gibbs and Vassilis Prevelakis, "Xos: An Overview" in *Object Management,* ed. Dennis Tsichritzis, pp. 37-61, Centre Universitaire d'Informatique (CUI), Geneva, Switzerland (July 1990).

[22] Carolyn Watters and Michael A. Shepherd, "A Transient Hypergraph-Based Model for Data Access," *ACM Transactions on Information Systems,* 8, 2, pp. 77-102 (April 90).

[23] Xavier Pintado and Dennis Tsichritzis, "Satellite: Hypermedia Navigation by Affinity," *Proceedings 1st European Conference on Hypertext (ECHT'90),* pp. 274-287, Paris, France (Nov 1990).