Suitable Databases for Process-centred Environments Do not yet Exist

Wolfgang Emmerich[1], Wilhelm Schäfer[1] and Jim Welsh[2]

[1] University of Dortmund, Informatik 10, P.O. Box 500 500, D-4600 Dortmund 50, Germany
[2] University of Queensland, Dep. of Computer Science, Queensland 4072, Australia

1 Introduction

As early as 1987, Bernstein has argued that dedicated database systems for software engineering, specialised with respect to functionality as well as implementation, are necessary [7]. He argued that the functionality and efficiency of existing systems (in particular, relational systems) does not adequately support the construction of software engineering tools and environments. Many researchers have picked up on this and a number of systems, some of which differ radically from standard relational technology, have been described in the literature. Some of these are now available as commercial products.

In this paper we argue that, despite the substantial number of proposed new database systems, a suitable database system for software development environments and especially process-centred environments does not yet exist. We do so by revising and refining some of Bernstein's requirements based on our own experiences in building such environments and tools. The second part of the paper briefly reviews a number of available database systems and shows that they still lack important features required by software engineering tools and environments built on top of them.

2 Database Requirements of Process-centred Environments

A Process-centred Software Development Environment (PSDE) consists of a process engine that coordinates the work of developers involved in a project, a set of integrated, syntax-directed tools that allow developers to conveniently manipulate and analyse documents and to maintain consistency between related documents of different types, and an underlying database which is capable of storing and manipulating process information and documents.

The process engine executes a formal description of a software development process in order to coordinate the work of developers involved in a project. It thus determines for each developer his or her personal agenda. The agenda indicates on which documents he or she may perform which particular actions. As usually a number of developers work in parallel the process engine must also manage the effects of the parallel work on each developer's agenda, i.e. it must possibly update one developer's agenda because of another developer's action. In addition, presentation and update of an agenda may happen in a distributed fashion.

The invocation of the tools that enable a developer to perform the actions contained in an agenda is controlled by the process engine via the agenda. The tools are used for example to edit, analyse, and transform documents. They should support the developers in producing syntactically correct documents and maintain interdocument consistency. They of course must also access documents in a distributed fashion.

Obviously all documents must be stored persistently. (The formal software process description plus the derived current state of a project is also considered as a document in itself which is accessed by the process engine.) Incremental update of all these documents is necessary to make the environment operate as safely as possible, i.e. to minimise work loss and to guarantee intra- and interdocument consistency in case of hardware or software failure, or when a single developer's session or even the whole project is suspended for some other reason. In addition, this incremental update must be fast to guarantee appropriate user response time of the environment.

The need for fast incremental update, the need for flexible access to shared documents (i.e. shared access is made possible by accessing different parts of a document) and the need for maintenance of intra- and interdocument consistency all require that documents are stored in some very fine-grained format. The most common representations used today are attributed directed graphs ("abstract syntax graphs") which have been proven to be particularly suitable for all kinds of syntax-directed tools [17], [8], [11], [18] and even the process engine itself [15].

Representing all documents together with their interdocument dependencies in a graph-like fashion results in a project-wide syntax-graph, in which documents are syntactically isolated subgraphs. The overall structure of such a graph has to be defined in terms of the data definition language of the database system and it must be established and controlled by the database's conceptual scheme. This implies that the data definition language is appropriate to cope with the complexity inherent in project-wide syntax-graphs. To manage this complexity the distinction of objects and types, encapsulation of objects' attributes by operations and information-hiding as well as inheritance to express generalisation/specialisation should be applicable to the data definition. Therefore, a powerful type mechanism including object constructors for expressing different types of aggregation and method definitions to achieve encapsulation must be provided.

As the overall process and all corresponding documents cannot necessarily be determined in advance, schema updates must still be possible without corrupting previously developed (parts of) documents.

In addition the database system must provide a query language enabling ad-hoc query facilities. The process engine, for example, frequently accesses the database to retrieve information about the current project state, i.e. the different documents' states. The nature of such queries cannot usually be determined fully in advance.

Views to support tool-oriented restriction of the project-wide conceptual schema are a further requirement.

Multi-user support by process-centred environments requires a sophisticated version concept for subgraphs which correspond to documents in order to enable the implementation of sophisticated check-in/check-out mechanisms. Supporting multiple software developers requires not only access control on the level of types like expressed by the conceptual schema but it must also be applicable to instantiated objects, i.e. objects of the same type may be granted different access rights. Those access rights serve as a basis for implementing flexible and programmable transaction schemes to support cooperating software developers as proposed by [6] or by [16].

Finally, a normal project size (let's say up to 20 developers) allows the database to run in a client-server mode. A substantial project size (let's say more than 20 developers) may require the database to be distributed across several servers.

3 Why Existing Systems Fail

It has already been argued a lot that relational technology already falls short in supporting an efficient manipulation of such fine-grained information as a project-wide syntax-graph. This has also been observed in other areas like CAD (cf. [14]). In addition, even the more sophisticated data modeling languages like Entity-Relationship based languages do not provide the powerful type system necessary. Finally, access control on the level of single tuples in a relation (which would be needed) as well as view definitions and versioning is either inadequately or not at all supported by relational systems.

So-called structurally object-oriented systems like PCTE/OMS [12], CAIS-A [2], Damokles [10], PGraphite [19] and GRAS [13] suffer from what we call the *granularity problem*. Their implementations assume objects to be of a certain level of granularity. Either objects are supported which have the size of complete documents and then correspond essentially to files or objects which have the size of basically a string together with some attributes, are the basis for the implementation. Whichever assumption is made, the systems fail to provide efficient manipulation of objects of other sizes.
None of these systems allows the definition of appropriate views.
In addition, these systems include either none or only predefined transaction schemes (and corresponding lock modes). They thus do not allow programming or adjustment of a transaction scheme to a particular project structure.
Finally, none of these systems except PCTE/OMS provides a mean for distribution of databases, nor do they allow distributed access to data stored on a server.

4 The Way Ahead

So-called fully object-oriented database systems [4] like Gemstone [9], Ontos [3] and O_2 [5] seem to be the most promising as environment platforms. By definition, the object constructors required to define the aggregations that occur in syntax-graph definitions such as tuples, sets, multi-sets and lists are provided by these systems. They also allow encapsulation and information-hiding. The inheritance mechanism supports late-binding, overloading and overriding of methods. As these databases are fairly general, their designers cannot assume any particular degree of granularity for the objects they manage. Thus, the granularity problem does not exist in these systems.

The way views are proposed in [1] for fully object-oriented databases is appropriate for presenting to a tool exactly those parts of a project-wide syntax-graph which the tool requires.
Some systems already provide different lock modes and thus enable definition of different transaction schemes. They all offer a client/server model to access the database in a distributed fashion.

These systems do still lack the more sophisticated transaction management and versioning needed by PSDEs. Nevertheless, their available functionality is an excellent basis to start from. In pursuit of this approach, an ESPRIT III project called GoodStep (General Object-Oriented Database for SofTware Engineering Processes) is currently under way to build an environment platform on top of O_2.

References

1. S. Abiteboul and A. Bonner. Objects and Views. In *Proc. of the ACM SIGMOD Conf. on Management of Data, Denver, Co*, pages 238–247. ACM Press, 1991.
2. Ada Joint Program Office. Common Ada Programming Support Environment (APSE) Interface Set (CAIS), Revision A. Technical Report DoD-STD-1838A, U.S. Department of Defense, 1988.
3. T. Andrews and C. Harris. Combining Language and Database Advances in an Object-Oriented Development Environment. In *Proc. of Object-oriented programming systems languages and applications, Orlando, Florida*, pages 430–440, 1987.
4. M. Atkinson, F. Bancilhon, D. DeWitt, K. Dittrich, D. Maier, and S. Zdonik. The object-oriented database system manifesto. In *Proc. of the* 1^{st} *Int. Conf. on Deductive and Object-Oriented Databases, Kyoto, Japan*. Elsevier Science Publishers B.V (North-Holland), 1990.
5. F. Bancilhon, C. Delobel, and P. Kanellakis. *Building an Object-Oriented Database System: the Story of* O_2. Morgan Kaufmann, 1991.
6. N. S. Barghouti. *Concurrency Control in Rule-based Software Development Environments*. PhD thesis, Columbia University, 1992. Technical Report No. CUCS-001-92.
7. P. A. Bernstein. Database System Support for Software Engineering. In *Proc. of the* 9^{th} *Int. Conf. on Software Engineering, Monterey, Cal.*, pages 166–178, 1987.
8. P. Borras, D. Clément, T. Despeyroux, J. Incerpi, G. Kahn, B. Lang, and V. Pascual. CENTAUR: the system. *ACM SIGSOFT Software Engineering Notes*,

13(5):14–24, 1988. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Mass.

9. R. Bretl, D. Maier, A. Otis, J. Penney, B. Schuchardt, J. Stein, E. H. Williams, and M. Williams. The GemStone data management system. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 283–308. Addison-Wesley, 1989.
10. K. R. Dittrich, W. Gotthard, and P. C. Lockemann. Damokles – a database system for software engineering environments. In R. Conradi, T. M. Didriksen, and D. H. Wanvik, editors, *Proc. of an Int. Workshop on Advanced Programming Environments, LNCS 244*, pages 353–371. Springer, 1986.
11. G. Engels, C. Lewerentz, M. Nagl, W. Schäfer, and A. Schürr. Building Integrated Software Development Environments. *ACM Transactions on Software Engineering and Methods*, 1, 1992. To appear, also Technical Report 60, University of Dortmund, Dept. of Computer Science, Chair for Software Technology.
12. F. Gallo, R. Minot, and I. Thomas. The object management system of PCTE as a software engineering database management system. *ACM SIGPLAN NOTICES*, 22(1):12–15, 1987.
13. C. Lewerentz and A. Schürr. GRAS, a management system for graph-like documents. In *Proc. of the* 3^{rd} *Int. Conf. on Data and Knowledge Bases*. Morgan Kaufmann, 1988.
14. D. Maier. Making database systems fast enough for CAD applications. In W. Kim and F. H. Lochovsky, editors, *Object-Oriented Concepts, Databases and Applications*, pages 573–582. Addison-Wesley, 1989.
15. B. Peuschel and W. Schäfer. Concepts and Implementation of a Rule-based Process Engine. In *Proc. of the* 14^{th} *Int. Conf. on Software Engineering, Melbourne, Australia*, pages 262–179, 1992.
16. B. Peuschel, W. Schäfer, and S. Wolf. A Knowledge-based Software Development Environment Supporting Cooperative Work. *International Journal for Software Engineering and Knowledge Engineering*, 2(1), 1992. To appear.
17. T. W. Reps and T. Teitelbaum. *The Synthesizer Generator – a system for constructing language based editors*. Springer, 1988.
18. J. Welsh, B. Broom, and D. Kiong. A Design Rational for a Language-based Editor. *Software — Practice and Experience*, 21(9):923–948, 1991.
19. A. L. Wolf, J. C. Wileden, C. D. Fisher, and P. L. Tarr. P Graphite: An Experiment in Persistent Typed Object Management. *ACM SIGSOFT Software Engineering Notes*, 13(5):130–142, 1988. Proc. of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments, Boston, Mass.

This article was processed using the LaTeX macro package with LLNCS style