

The GOODSTEP Project: General Object-Oriented Database for Software Engineering Processes*

The GOODSTEP Team

May 24, 1994

1 Objective of GOODSTEP

The goal of the GOODSTEP project is to develop a sophisticated database system dedicated to the support of Software Development Environments (SDEs) and make the basis for a platform for SDE construction with a software process tool-set and generators for graphical and textual integrated tools implemented on top of it.

The GOODSTEP project started September 1992 and will last for three years. This paper mainly reports on the first six months of work within the project.

The baseline of the project is an existing European commercially available object-oriented database product: O_2 [O2 93]. Rather than developing a new database management system from scratch, GOODSTEP will enhance and improve this product.

Besides the enhancements and improvements of O_2 which make it an admirably suited system for SDEs, the project will provide a number of test cases and will perform case studies to evaluate and justify the approach. This will include porting and developing a number of existing software engineering tools on top of the new platform, the development of tool generation capabilities to exploit the features of a fully object-oriented system, and the development of advanced software process modeling capabilities which, once again, exploit the provided features of O_2 .

The choice of an object-oriented database management system as baseline of the project derives from the inadequacy of relational database systems for software engineering applications, which has been recognized for quite some time [Mai89, ZB92]. This has resulted in a great effort in both the industrial and research communities to extend the current database technology towards more powerful and flexible database management systems [LZ92]. In particular, in this context we are interested in the work done on the development of object-oriented database systems.

The class of object-oriented database systems can be roughly classified in two categories: those offering the full functionalities of the object-oriented data model, which we will refer to from now on as *fully* object-oriented databases¹, and those offering only a subset of the object-oriented model, which we will call *structurally* object-oriented database systems. The main distinction between structurally object-oriented databases and fully object-oriented, which is also the main drawback of the first class, is that most of the structurally object-oriented database systems, such as PCTE/OMS [GMT87], Damokles [DGL86], assume a certain level

*This work is partly funded by the CEC under contract No. 6115 (ESPRIT-III Project GOODSTEP)

¹Fully object-oriented database systems are sometimes also referred to as object database systems

of granularity of the objects to be stored and retrieved and that they further can not define encapsulation of data structures by operations. These systems either support relations between coarse-grained objects such as products of the SE life cycle (e.g. A is the specification of B, A is owned by developer Q, etc.), or else support a rather fine-grained level of objects such as syntactic units of programs or specifications (i.e. statements, variables, procedures, etc.).

In practice, many software engineering tools require support for both levels of granularity. By contrast, fully object-oriented database management systems (OODBMSs) are the best approach to support sophisticated efficient storage and retrieval of objects at arbitrary levels of granularity [EKS93].

The project will amply demonstrate two important features of a fully object-oriented system. In the first place, *it will enable much easier implementations of SE-tools than when using conventional DBMSs*. This is because the richer semantics of the object oriented model is most suited to the complex data handled in software engineering applications, and also because there is no problem of fixed granularity of objects. In the second place, *it will improve significantly the functionality of software tools which can be based upon it*.

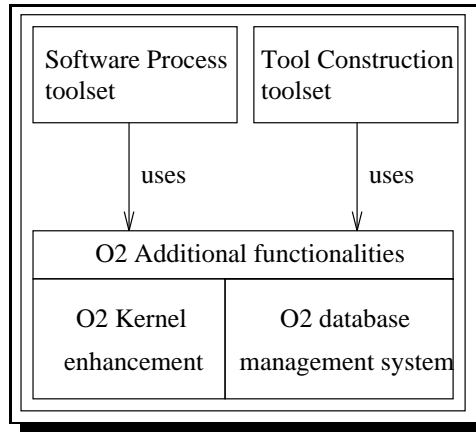


Figure 1: The GOODSTEP SDE platform

The software architecture of GOODSTEP is illustrated in Figure 1. GOODSTEP will enhance the functionalities offered by O_2 , by adding new functionalities to its kernel or on top of it. Moreover, two tool sets will be developed together with the enhanced O_2 system to constitute the GOODSTEP platform. The first is a set of tools supporting development, simulation and execution of software processes called Software Process Tool-Set (SPT). The other tool-set called Tool Construction Tool-Set (TCT) supports the development (generation) of new tools. Both SPT and TCT will support the development of a customized SDE.

A more detailed description of the project can be found in [GOO93b, GOO92, GOO93a, BFGL93, Sof93, BBD⁺93].

2 Building the GOODSTEP Platform for Software Development Environments

The methods and languages to be used for the development of a software application – be they graphical or textual languages – depend on the application domain. For example, realtime applications require different specification and implementation languages than financial or medical applications. Moreover, the process models that support an application development best can not be predefined. They depend not only on the application domain, but also on

the organization and on the people who are running the processes. Current SDEs suffer from their selection of specific combinations of languages and often assume a particular process. Both usually do not completely cover the needs of any software development and impose a pre-specified development mode. Instead, the software industry demands customizable SDEs, which may be easily adapted to the evolving needs of software development. GOODSTEP addresses this need by providing means to

- define the process used for developing a software system within the software process tool-set,
- define SDE conceptual schemas and
- generate the tools needed during the course of a software process using the tool construction tool-set.

Using the process tool-set a model tailored to the needs of a particular institution or even project can be modeled, analysed and later used for running a software project. Using the tool construction tool-set, it will be possible to define and generate conceptual schemas and define and generate a set of integrated syntax-directed software development tools from appropriate specification languages.

Software Process Modeling and Enactment Software process modeling and enactment in GOODSTEP is achieved using the SPADE (Software Process Analysis Design and Enactment) environment [BBFL94]. SPADE provides a domain-specific language for the modeling and enactment of software processes called SLANG (Spade LANGuage) [BFG93]. SLANG is based on high-level nets and is given formal semantics in terms of a translation scheme from SLANG objects into ER nets. ER nets [GMMP91] are a mathematically defined class of high-level Petri nets that provide the designer with powerful means to describe concurrent and real-time systems. In ER nets, it is possible to assign values to tokens and predicates to transitions, describing the constraints on tokens consumed and produced by transition firings.

Conceptual Data Modeling The abstract specification phase for data is traditionally called *conceptual data modeling*; its output is called the *conceptual schema* [BW92]. The GOODSTEP platform will contain a tool for modeling the schema of an SDE on a conceptual model, using a set of high-level abstraction mechanisms which improve the expressiveness of software engineering database conceptual schemas. An initial semantic data model called SDM^2 , has been defined, which enables modeling of software engineering processes and products in a direct and natural way. SDM^2 supports the definition of the dynamics of software development (i.e. the order in which software artifacts are built) and the concept of design refinement applied to product modeling. SDM^2 is partially supported by Sidereus and Galileo. Galileo [ACO85] is a strongly typed interactive database programming language. Sidereus is a software system composed of two components. The first component is a graphical database schema editor for Galileo. The second is a code generator whose input is a conceptual schema specified with the graphical editor and whose output is Galileo code supporting creation and manipulation of the database. By interpreting the generated code, the database conceptual design can be quickly validated against the application's requirements.

Tool Construction The GOODSTEP platform will contain two tool generators. The first one is the GraphProject compiler that is capable of generating graphical tools. The second is the GENESIS compiler which takes a specification written in the object-oriented tool specification language GTSL [EK93] and automatically derives textual syntax-directed tools. We

first discuss the requirements that users ² put forth on generated tools and then sketch some issues on tool design which later on have an impact on the database functionality required to make tool generation feasible.

3 How to Meet the Requirements: The Enhanced O_2 Object Database System

Relational database systems are inappropriate for storing project graphs, since (1) the data model can not express syntax graphs appropriately, (2) relational database systems do not support versioning of document subgraphs and (3) relational database systems cannot be used to implement customized transaction schemes. No structurally object-oriented DBMS meets all of our requirements. Those that are capable of efficiently managing project graphs, lack functionality w.r.t. views, versioning, access rights and adjustable transaction mechanisms, and distribution. Others that offer these functionalities are unable to manage the large collections of small objects as they occur in project graphs and are therefore inappropriate. The approach of GOODSTEP is therefore to start from an existing object-oriented DBMS which is the O_2 system since this already addresses some of the requirements posed on a DBSE by an SDE. A major effort in GOODSTEP is then devoted to enhancing O_2 enhancing or adding functionalities for

Change management:

- schema updates,
- versions management,
- view definition capabilities,

Active databases:

- triggers,
- constraints

Change Management

Object-oriented database management systems are often used for complex data whose structure is likely to change over time, yet the problem of change management has not not been completely solved by any commercial or research system. The support for change management in an object-oriented database system is a fundamental requirement. The process of software design is incremental by nature and implies changes at various levels throughout the entire life-cycle of development. Change management is inherently more complex for an OODBMS than it is for a conventional relational database system due to the richness of the underlying object-oriented data model. It is naturally related to the mechanisms for supporting views, advanced transactions, and configuration management.

Schema evolution Schema evolution in an object-oriented database system refers to the ability to change both the schema and consequently the database. Every time a schema is modified then the database has to be updated to be brought to a consistent state with respect to the new schema.

A schema can be changed normally using special primitives, see for example [Zic89] and [Zic92]; while the database is modified using user-defined conversion functions which take as

²As the users of the GOODSTEP platform have different roles, we distinguish in the sequel users which are the developers that use the customized GOODSTEP platform in order to develop an application from SDE builders who use the features provided by the GOODSTEP platform for customizing it to obtain a particular SDE.

input parameters the old and new schema class definitions and when executed transform the objects of the database to conform to the new schema [Obj93a]. The body of a conversion function defines the desired database transformation, and it is normally defined using the data manipulation language offered by the OODBMS, e.g. C++ [Obj93a], [Obj93b].

The SDE builder has to define for each modified class in the new schema a conversion function. System default transformations are applied in case no explicit conversion functions are given by the builder [Ita91] [Obj93a], [Ver92].

What is important for the SDE builder is that after execution of the appropriate conversion functions, the entire database must be in a consistent state with respect to the new schema.

From an implementation point of view, conversion functions are updates to the database. There are mainly two strategies for implementing database conversion functions: immediate and lazy [HVZ90]. In the first case, all objects of the database are updated immediately after the execution of all the conversion functions. In the second case, objects are updated only when used (i.e. conversion functions are executed only when objects are effectively used).

No matter what strategy is used, the effect on the database has to be the same: the database must be in a consistent state with respect to the new schema [FZ93].

Versions In software engineering, versioning is applied to documents. Documents are usually persistently stored in a repository as a graph that aggregates a high number of objects. This graph is seen as a composite object. There are reference relationships between different objects in the graph and even to other graphs in order to represent the relationships between different parts of possibly different documents. To create a new version/revision of a document, a new version of the objects contained in the transitive closure of the aggregation relationship must be created. Reference relationships must be redirected to the new versions. The delta storage technique is applied to copy only those objects which are really modified. In this sense, the new version/revision is a “virtual” version of the document.

The O_2 version mechanism [GOO93a] will try to solve these versioning problems. It will also enable the history of an object to be maintained and different versions of the same object to be used concurrently without them waiting for each other (e.g. the system provides a concurrency control at the object level to ensure that no conflict arises when working on different objects.)

Views In our approach, the definition of a view is similar to the definition of a schema. a *virtual schema* is, like a normal schema, an organizational unit meant to encapsulate a set of related intentional definitions [AB91, GOO93a]. The main difference is that a real schema describes the structure and behavior of real data stored in the database whereas a virtual schema describes a virtual world.

A view is thus in our context a special kind of schema with certain restrictions. We use the term *virtual schema* as a synonym of view in what follows and the term *real schema* is used in contrast to virtual schema to avoid confusion.

Like a real schema, a view includes definitions of classes, methods, types, functions and named objects. It may also import and export definitions from other schemas, although these mechanisms are given a slightly different semantics. In addition, virtual definitions can be included in a virtual schema. As a matter of fact, the distinguishing point between a real and a virtual schema is the fact that the latter has at least one virtual or imaginary class (possibly imported) whereas the former has only real classes.

Active Databases In the framework of GOODSTEP active rules have been introduced into O_2 as a means of supporting SDE semantics, mainly for (i) users or SDE builders, (ii) access

logging of the SDE, (iii) organizing related application programs, (iv) tools communication, (v) change propagation, and (vi) maintaining data consistency [GOO93a]. Rules introduced in O_2 are not AI rules. They match production rules and are comprised of three parts: an Event part, a Condition part and an Action part. The general semantics of a rule is the following: "Whenever the event E occurs, if the Condition C holds, then execute Action A".

The members of the GOODSTEP team and their affiliations are S. Abiteboul (INRIA), M. Adiba (Uni Grenoble), P. Armenise (Engineering), S. Bandinelli (Cefriel), L. Baresi (Cefriel), P. Breche (Uni Frankfurt), C. Collet (Uni Grenoble), P. Corte (Engineering), C. Delobel (INRIA), W. Emmerich (Uni Dortmund), S. Even (Uni Frankfurt), G. Ferran (O_2 Technology), F. Ferrandina (Uni Frankfurt), A. Fuggetta (Cefriel), P. Habraken (Uni Grenoble), P. Kroha (Uni Dortmund), L. Lavazza (Cefriel), J. Madec (O_2 Technology), S. Sachweh (Uni Dortmund), M. Sakkinen (Uni Frankfurt), W. Schäfer (Uni Dortmund), C. Souza (INRIA), E. Waller (INRIA) R. Zicari (Uni Frankfurt).

References

- [AB91] 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.
- [ACO85] A. Albano, L. Cardelli, and R. Orsini. Galileo: A Strongly Typed, Interactive Conceptual Language. *ACM Transactions on Database Systems*, 10(2), 1985.
- [BBD⁺93] W. Beckmann, J. Brunsmann, D. Dong, W. Emmerich, P. Kroha, W. Reimer, S. Sachweh, and W. Schäfer. The Goodstep Tool Specification Language Reference Manual. Deliverable ESPRIT Project GOODSTEP 6115-6P, Commission of the European Communities, November 1993.
- [BBFL94] S. Bandinelli, M. Braga, A. Fuggetta, and L. Lavazza. The Architecture of the SPADE-1 Process-Centered SEE. In *3rd European Workshop on Software Process Technology*, Grenoble (France), February 1994.
- [BFG93] Sergio Bandinelli, Alfonso Fuggetta, and Carlo Ghezzi. Process Model Evolution in the SPADE Environment. *To appear in IEEE Transactions on Software Engineering. Special Issue on Process Evolution*, December 1993.
- [BFGL93] S. Bandinelli, A. Fuggetta, C. Ghezzi, and L. Lavazza. The SLANG 1.0 Process Modelling Reference Manual. Deliverable ESPRIT Project GOODSTEP 6115-3P, Commission of the European Communities, September 1993.
- [BW92] J. A. Bubenko and B. Wangler. Research Directions in Conceptual Specification Development. In P. Locopoulos and R. Zicari, editors, *Conceptual Modeling, Databases and CASE — An Integrated View of Information Systems Development*. John Wiley, 1992.
- [EK93] W. Emmerich and P. Kroha. GTSL — An Object-Oriented Language for Specification of Syntax Directed Tools. Technical report, University of Dortmund, Dept. of Computer Science, Chair for Software Technology, 1993. To appear.
- [EKS93] W. Emmerich, P. Kroha, and W. Schäfer. Object-oriented Database Management Systems for Construction of CASE Environments. In V. Marik, J. Lazansky, and R. R. Wagner, editors, *Database and Expert Systems Applications — Proc. of the 4th Int. Conf. DEXA '93, Prague, Czech Republic*, volume 720 of *Lecture Notes in Computer Science*, pages 631–642. Springer, 1993.

- [FZ93] F. Ferrandina and R. Zicari. Object Database Schema Evolution: are Lazy Updates always Equivalent to Immediate Updates? In *Proc. of the OOPSLA Workshop on Supporting the Evolution of Class Definitions, Washington, DC*. ACM Press, 1993.
- [GMMP91] Carlo Ghezzi, Dino Mandrioli, Sandro Morasca, and Mauro Pezzé. A Unified High-level Petri Net Formalism for Time-critical Systems. *IEEE Transactions on Software Engineering*, February 1991.
- [GOO92] GOODSTEP Team. Description of Software Engineering Applications and Requirements for an Object-Oriented Repository. Deliverable ESPRIT Project GOODSTEP 6115-1P, Commission of the European Communities, December 1992.
- [GOO93a] GOODSTEP Team. Architecture and Functionalities of the GOODSTEP Repository as implemented in the first Prototype. Deliverable ESPRIT Project GOODSTEP 6115-2R, Commission of the European Communities, September 1993.
- [GOO93b] GOODSTEP Team. The GOODSTEP Project: General Object-Oriented Database for Software Engineering Processes. GOODSTEP Technical Report 1, University of Frankfurt, November 1993.
- [HVZ90] G. Harrus, F. Velez, and R. Zicari. Implementing schema updates in an object-oriented database system: a cost analysis. Technical report, GIP Altair, 1990.
- [Ita91] ITASCA Technical Summary, Release 2.0. Technical report, ITASCA Systems, Inc., September 1991.
- [LZ92] P. Loucopoulos and R. Zicari. *Conceptual Modeling, Databases and CASE — An Integrated View of Information Systems Development*. John Wiley, New York, 1992.
- [Mai89] 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.
- [O2 93] O2 Technology. *The O₂ Programmer's Manual – Version 4.3*. 7 rue du Parc de Clagny, 78000 Versailles, France, 1993.
- [Obj93a] Object Design Inc. *Objectstore User Guide, chapter 9*, 1993.
- [Obj93b] Objectivity Inc. *Objectivity, User Manual, Version 2.0*, Mar 1993.
- [Sof93] Software AG, Italy. CASE Market Analysis. Deliverable ESPRIT Project GOODSTEP 6115-4P, Commission of the European Communities, September 1993.
- [Ver92] Versant Object Technology, 4500 Bohannon Drive Menlo Park, CA 94025. *Versant User Manual*, 1992.
- [ZB92] R. Zicari and C. Bauzer-Meideros. New Generation Database Systems. In P. Loucopoulos and R. Zicari, editors, *Conceptual Modeling, Databases and CASE — An Integrated View of Information Systems Development*. John Wiley, 1992.
- [Zic89] R. Zicari. Schema updates in the o₂ object-oriented database system. Technical Report 89-057, Politecnico di Milano, October 1989.
- [Zic92] R. Zicari. A framework for schema updates in an object-oriented database system. In *Building an Object Oriented Database System - The story of O₂*. Morgan Kaufmann Publishers, San Mateo, CA, 1992.