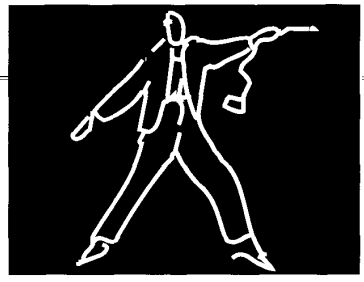

Strategies for Integrating CASE Environments

MATTHIAS JARKE, *Technical University, Aachen*

◆ *A framework based on a process model lifts integration from files and documents to the conceptual level, taking care of lower level details with mapping assistants.*

Much of today's software market involves data-intensive information systems, and as databases are extended to design, process control, and multimedia applications, this market share may become even larger. Yet information systems remain hard to maintain and reuse. The primary reason is their lack of integration. Although programmers have many individual development tools at their disposal, there is no formal integration across development stages, between the system and its environment, or across development tasks.

One way to address this problem is to view the development environment itself as a data-intensive information system centered around a repository. The question then becomes how to formalize and implement such a repository.

An experimental information-system environment, called DAIDA (Development Assistance for Integrated Database Applications), was developed as part of the European Community's ESPRIT program to examine this question. The DAIDA project team, which I managed, found that by making process-oriented conceptual models operational through knowledge representation and database techniques, we could integrate development stages and development tasks.

DAIDA goes beyond traditional knowledge-based techniques for CASE (described in the box on p. 56) by addressing three important dimensions of integration in a process-oriented model: how to handle dependencies among development stages, how to manage the evolving relationship among systems and their technical and social environments, and how to integrate development tasks — from both development in the small, in which the focus is the content of actions and results, and development in the large, which is concerned with object and process man-

0740-7459/92/0300/0054/$03.00 © IEEE

agement and the collaboration of people involved in developing and using systems.

Because information-system development is a continuous, cooperative process of analysis and reanalysis, design and redesign, and programming and program reorganization, process information should be stored in a repository of experience. We developed such a repository, called ConceptBase, and used it to define our process-oriented integration model.

> Conceptual modeling is a way to overcome developer-user communication problems.

ConceptBase considers integration at two levels. At the specification level, it uses metamodeling to formally interrelate languages, methods, and tools through object structures, rules, and constraints. At the implementation level, it integrates external tools using the trigger concepts from database technology. Triggers, or event-condition-action rules, are programs activated when a certain event like a database update happens and some additional constraint is satisfied. ConceptBase relates the two levels by constraint- and rule-compilation techniques originating from research in deductive databases.[1]

THREEFOLD INTEGRATION STRATEGY

Integration in DAIDA begins with concept-based specifications or *conceptual models*. It then uses *process-centered development* to arrive at *quality-assured application software*. The purpose of each phase is to enhance communication between developers and users in requirements analysis and system specification — notoriously difficult areas.

Conceptual modeling — using object-oriented representations, hypertext-like interface technologies, and animated prototypes — appears to be one of the few ways to overcome developer-user communication problems. Conceptual models also specify how a system fits into its environment. If you implement these models in an information system, you can base design decisions on them and document the relationships between those decisions and the models.

The need for process-centered development stems from the need to have process details travel with the information system as it evolves. Many information systems live beyond single generations of hardware, system software, and development teams, but their longevity is proportional to how much of the developer's experience transfers with the system's history. Given the high turnover of software personnel, it is wise to keep a detailed record not only of outcomes, but also of the design decisions and tool applications involved in development.

Quality assurance is an integral part of integration because integration is often driven by organizational requirements and goals. Any integrated environment should have a range of formal tools for producing and evaluating system quality. These tools also make it easier to reenact development decisions during maintenance. Total quality assurance using appropriate formal methods and verification and testing tools may be too expensive for many applications, but the decision not to invest in it should be a conscious one.

Conceptual modeling. Conceptual languages let you work with adequate concepts when specifying an application's semantics. In requirements specification or analysis, you need the freedom to define application-specific concepts and terminology. In contrast, during the design phase, you need a predefined but powerful set of constructs to represent a system perspective. To integrate at the implementation level, you need database-programming models.

Figure 1 shows what conceptual modeling in DAIDA consists of. Requirements modeling is not confined to describing the system's requirements but takes into account the broader context of system use. The *world model*, which encompasses the subject world, the usage world, and the development world, captures knowledge about the role of system components. The subject world serves as the basis for the information system's data model. It describes how the system model represents objects. The usage world describes where and how the system model will be used. The seeds for specifying system functions and the user interface are in the usage world. The development world is the environment of system versions, configurations, and development teams, in which the system evolves.

Figure 1 also shows the two other levels of the conceptual model: conceptual design and database programs. During conceptual design, you organize the system components from the specialized

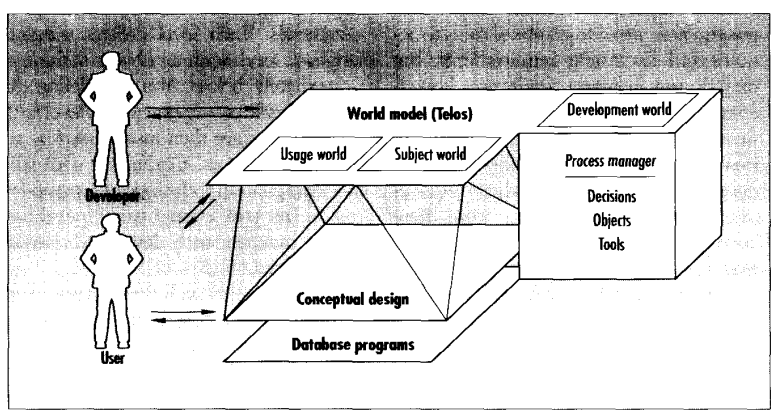

Figure 1. Conceptualization tasks in DAIDA.

KNOWLEDGE-BASED CASE

David Barstow gives a good overview of how formally based tools have been used in knowledge-based software support.[1] While most of these projects have studied knowledge-based assistants (expert systems) for individual development tasks, a few have also looked at integration, often on a narrower scale than the DAIDA team has and without actual proof of concept through the use of operational integrated prototypes.

The Programmer's Apprentice project at the Massachusetts Institute of Technology sees integration as how to horizontally compose so-called cliches for a single representation level, but it does not deal with information systems specifically. The Knowledge-Based Software Assistant of the US Air Force which pioneered knowledge-based assistants, is just beginning to consider integration seriously.

Besides these AI-oriented approaches, our integration approach has similarities to several ideas developed in the database community. Rule-based technology for integrating and controlling external tools is also used in the Marvel project at Columbia University.[2] The representation and consistent maintenance of dependency structures created by applying such integrated tools is also supported by the Cactis database developed in the Arcadia consortium.[3]

Neither KBSA nor Cactis supports a specific process model of information-system development in the context of how the system's environment evolves.

REFERENCES

1. D. Barstow, "Artificial Intelligence and Software Engineering," *Proc. Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., 1987, pp. 200-211.
2. G. Kaiser et al., "Database Support for Knowledge-Based Engineering Environments," *IEEE Expert* Spring 1988, pp. 18-32.
3. S. Hudson and R. King, "Cactis: A Self-Adaptive, Concurrent Implementation of an Object-Oriented Database Management System," *ACM Trans. Database Systems*, Sept. 1989, pp. 291-321.

viewpoint of integrated information systems. At the database-programs level, software-specific concepts serve as the basis for integrated program production.

World model. The heart of the world model is the conceptual-modeling language Telos.[2] Telos integrates predicative assertions and an interval-based time calculus in a semantic network with built-in axioms for aggregation, generalization, and classification. It thus lets you manage conceptual models as an evolving knowledge base.

Telos is more flexible than most specification languages and tools. These languages often provide graphical tools to acquire and document requirements, but they do not maintain results as a knowledge base, they cannot transfer requirements to system specification and implementation, and they cannot formally reuse development experience when requirements change. Telos differs from these languages because it prescribes the information system's dynamic behavior. World and system models evolve as learning occurs or reality changes because you can manipulate the requirements model as a dynamic knowledge base, not just as one-shot documentation.

Telos also provides a way to represent time, an important feature because requirements analysis is a dynamic process that describes a dynamic world. Historical time in Telos indicates how application processes happen in time; transaction time records how your understanding of these processes evolves.

Application areas for information systems vary widely, yet as a basis for user involvement, the language should provide a means of communication close to the application. The language must therefore let you define application-specific concepts and reference models dynamically. Classification in Telos lets you stratify the knowledge base in any number of metalevels. Each level defines the sublanguage for describing objects of the level immediately below. You can define domain-specific concepts at a metalevel fairly easily. These are then instantiated by actual requirements. Combined with suitable compilation techniques, meta modeling helps you create interoperability among independently developed software components.

Requirements analysis is a major cooperative task with contributions from various stakeholder and developer groups. A language should give you enough modularity to model the evolution of individual opinions as well as their integration in a common requirements model.

A conceptual language should also let you visualize requirements through graphical or text-based interaction. You should not have to learn a formal syntax. Even the support team needs a lot of guidance. Telos integrates graphical semantic network principles with frames and rules. It thus provides a basis for the hypertext interface implemented in ConceptBase. You can use the predicative sublanguage as a filter to make only relevant parts of the world model visible in the hypertext network.

Users often need animation to understand formal requirements analyses. A conceptual modeling language should let you run examples through the requirements description, using derivation rules or similar approaches to simulate system behavior. You can use Telos's deduction rules to interactively animate the world model by making deductive queries about prototypical sample objects.

Conceptual design. Conceptual design consists of formally modeling the system itself. It still requires a semantically rich set of concepts, but this set is fixed as a uniform structuring mechanism for information systems. There is a delicate balance to maintain. On the one hand, the design language should not be too different from the requirements language, but on the other hand, as a starting point for formal refinement methods, the conceptual design must be formally consistent and complete. Thus, a heuristic understanding of the specification with only partial formalization is insufficient as a basis for integration.

Many existing semantic data models consider only database design. DAIDA's Taxis Design Language offers generalization hierarchies of data and transactions as well as set-oriented assertions, but no metalevel extensibility. Instead of a time concept, TDL adopts a state-based view of computation as most programming languages do. Data management is organized as entity classes related by attributes; transactions bring about atomic state transitions, while scripts describe the long-term pattern of global coordination and timing.

We learned from working with Telos and TDL that a direct transition from the requirements model to a formal system specification is problematic with large information systems. The key is to reorganize the application knowledge gathered in the requirements phase from the viewpoint of how the information system will manage information. The perspective is from an overall view like that of a data dictionary, beyond the individual database program's specification.

Database programs. The development of correct and efficient database software is neither a database design task nor a classical programming effort, in which the emphasis is on optimizing individual applications. Instead, this task requires integrated concepts based on advanced database- and systems-programming technology. These concepts serve as a front end for integrating target database systems and application languages.

As part of our work on DAIDA, we used the DBPL database language[3] as a possible candidate for such a front end. We also constructed mappings from TDL to DBPL and from DBPL to commercial relational database systems.

Process-centered development. At least as important as conceptual models for individual tasks is the modeling of their interrelationships. Such a model must encompass an *abstract conceptual model* that describes individual concept models uniformly and a *process model* that captures the relationships among abstract objects. The second model requires more than just observing objects in isolation. It requires knowledge about how the system was developed. Neither development stages nor steps are predetermined, but emerge from development tasks, available tools, and the development team.

One of DAIDA's design goals was to make this process explicit and support it over long development periods. The process model is integrated with the conceptual model through the development world, as shown in Figure 1. For that reason, we use Telos for both the conceptual and process models.

DAIDA's process model is a metamodel called DOT (decision-object-tool).[4] DOT, which generalizes several earlier decision-oriented process models,[5] represents states of a development process by documenting relevant properties of results achieved in that state as objects.

We represent state transitions by documenting and justifying the decisions leading to the results; decisions can address refinement within a DAIDA level, mapping between levels, versioning to change previous decisions, or reconfiguring to group existing system components. Because the development environment may change over the information system's life, we also represent the tools that support decision execution. The purpose of managing all this information is to transfer development experience throughout a system's life.

ConceptBase is DAIDA's metadata management and reasoning facility. It provides the information required by the development process and ensures that the process is formally correct. We use Telos's metaclass hierarchy to document metaclasses, classes, and instances. The metaclass level defines the basic structure for development processes, the class level describes the development environment at hand, and the instances level consists of concrete development projects within the environment.

At least as important as conceptual models for individual tasks is the modeling of their interrelationships.

Each level may evolve along version histories under the control of the model at the level above it. The graphical view of Telos serves as a basis for browsing in version histories, along development levels and usage relationships. The formal view focuses your attention by making predicative queries before letting you graphically explore the system.

DOT is a methodology-independent representational framework. DAIDA assistants support a spiral model of top-down development, but other DOT instantiations, even using the same languages, might support very different development paradigms. For example, you can develop new applications by configuring reusable development histories. This extension is studied in another ESPRIT project, ITHACA (Interactive Toolkit for Highly Advanced Computer Applications).[6]

Quality-assured software. To provide a controlled degree of quality assurance, you have to do more than document the evolving relationships between representation levels. You need supporting tools to validate requirements or designs and to map the three conceptual levels: world model, conceptual design, and database programs.

DAIDA accommodates validation by prototyping in Prolog. For mapping, it offers two knowledge-based assistants: Iris, which maps from the world model to conceptual design, and DBPL-Map, which maps from conceptual design to database programs.[3] These knowledge-based assistants not only help you satisfy functional requirements but also support nonfunctional goals like efficiency and accuracy.

Iris recognizes that conceptual design is not simply an elaboration of the world model by supporting distinct design decisions over and above initial requirements.[7] Such decisions might include satisfying temporal conditions by transactions or scripts or defining how long data should be kept. Iris also lets you satisfy assertions at the world-model level by providing integrity constraints on data. You can either perform precondition tests on transactions or specify structures and operations that satisfy assertions by design (for example, error-preventing menu interfaces).

A design decision can even cause a regrouping of the system model's data structures; for instance, the designer may decide to organize generalization hierarchies of concepts by their temporal actuality rather than by content.

Mapping from conceptual design to database programs requires more formal and standardized support. DBPL-Map

derives application modules from the conceptual design using abstract machine specifications as an intermediate representation. It translates a coherent subset of TDL classes — the specification for the intended program — into an abstract machine specification and checks it for consistency and formal completeness. From this initial abstract machine, the designer can derive refined machines in part automatically, in part manually.

Each refinement step generates many proof obligations. You can either just sign them off as satisfied, or carry out a formal, computer-assisted proof — thus choosing among various degrees of quality assurance. The last refinement result should be so close to a DBPL representation that automatic translation is possible.

Each refinement step generates proof obligations. You can use these or conduct a formal proof.

ARCHITECTURE

As Figure 2 shows, DAIDA consists of a set of dedicated tool boxes coordinated by ConceptBase. Grafic (not shown), an adaptable graphical editor/browser for knowledge bases, supports the common functions of related languages, each of which has different constructs and use patterns that make up its individual environment.

Iris and DBPL-Map are organized as extensible tool kits because the development theories they comprise may change. They include theorem provers for partially automated programming and verifying of critical components. Similar assistants help elaborate, analyze, and prototype models at each conceptual-modeling level.

ConceptBase interacts with other tools by documenting and retrieving their results and the underlying decisions. It also models the evolution of the DAIDA environment itself. You can use an interactive tool to integrate externally developed CASE tools into the DOT framework. The mechanism views tool functionality as a set of DOT decision classes, determines the DOT object types for tool I/O, lets you specify pre- and postconditions for correct application, and integrates the actual tool calls at the implementation level.

DAIDA is implemented in a wide-area client-server architecture, with ConceptBase as the server and all other tools as the clients.

A hypertext-style standard client lets users browse, filter, and edit along dimensions like development hierarchies, version histories, and call relationships. Other standard clients include DOT-based conceptual front ends to commercial software for teamwork support and version and configuration management.[8] You can also add environment-specific tools. By adding hypertext editors instead of programming tools and changing the definition of methodologies, for example, we converted DAIDA into a coauthoring system for documentation rather than program code.

DEVELOPMENT EXAMPLE

Figure 3 shows a detailed example extracted from an actual information-systems project. The figure shows the four system stages defined at (from the top) the world-model, conceptual-design, and database-program levels. The world model contains persons, some of whom are employees of research companies. The model assumes initially that each employee works on at most one project. Telos reflects this by making workson an instance of the attribute class Single. Similarly, the attribute class Unique reflects that each employee name is unique.

Persons may turn into employees by hireEmp activities, which instantiate the belongsto link to a company. Persons may also be hired directly for specific projects, which is designated by hireEfP (hire employee for project). An integrity constraint restricts an employee to projects from his particular company.

Because the developer has decided to store only information about persons who are employees, Iris collapses the generalization hierarchy Employee-Person to a single TDL entity class, called EmplPers. EmplPers inherits the attributes of both its origin classes, hireEmp and hireEfP. In contrast, each Telos activity class is separately mapped into a TDL transaction. The same direct mapping applies for project and company objects.

DBPL-Map converts this structure to a relational database with a relation for each TDL entity class, artificial keys c# and pr# to ensure object identity, and a referential constraint that makes sure employees work in existing companies and on existing projects. The implementation of transaction specifications like hireEfP must take into account the inherited parameters and functionality of the hireEmp transaction; it must also add a precondition to the execution of the transaction code: the integrity constraint of the world model (mapped to a subset invariant of EmplPers in TDL) must be satisfied before execution.

In evaluating this system concept, the developer has two major criticisms, which result in Decisions 1 and 2 (gray bars in the figure). First, prototyping shows that users are confused if they have to deal with two kinds of transactions. To eliminate this confusion, the developer uses inheritance to change the world-model-to-design mapping so that the isa hierarchy of transactions collapses. He does not have to change the DBPL code as long as an artificial project, called general hiring, is introduced for employees not hired for projects. He can then discard the transaction program hireEmp and use hireEfP.

In Decision 2, the developer determines that employees may in fact work on more than one project. He then removes the instantiation link to Single in the world model. To map this change to TDL, he simply makes workson set-valued. However, to retain a normalized relational database schema at the implementation level, he must add a new relation, Workson, which represents the many-to-many relationship between em-

ployees and projects. Together with a referential constraint, this addition ensures that the relationship between existing projects and employees will be constructed appropriately. On the other hand, the developer must omit pr# from EmplPers. The new database structure then implies changes in hireEfP.

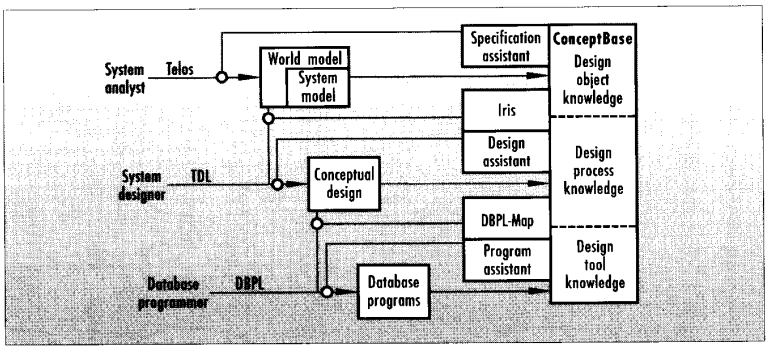

Figure 2. Functional view of the DAIDA architecture.

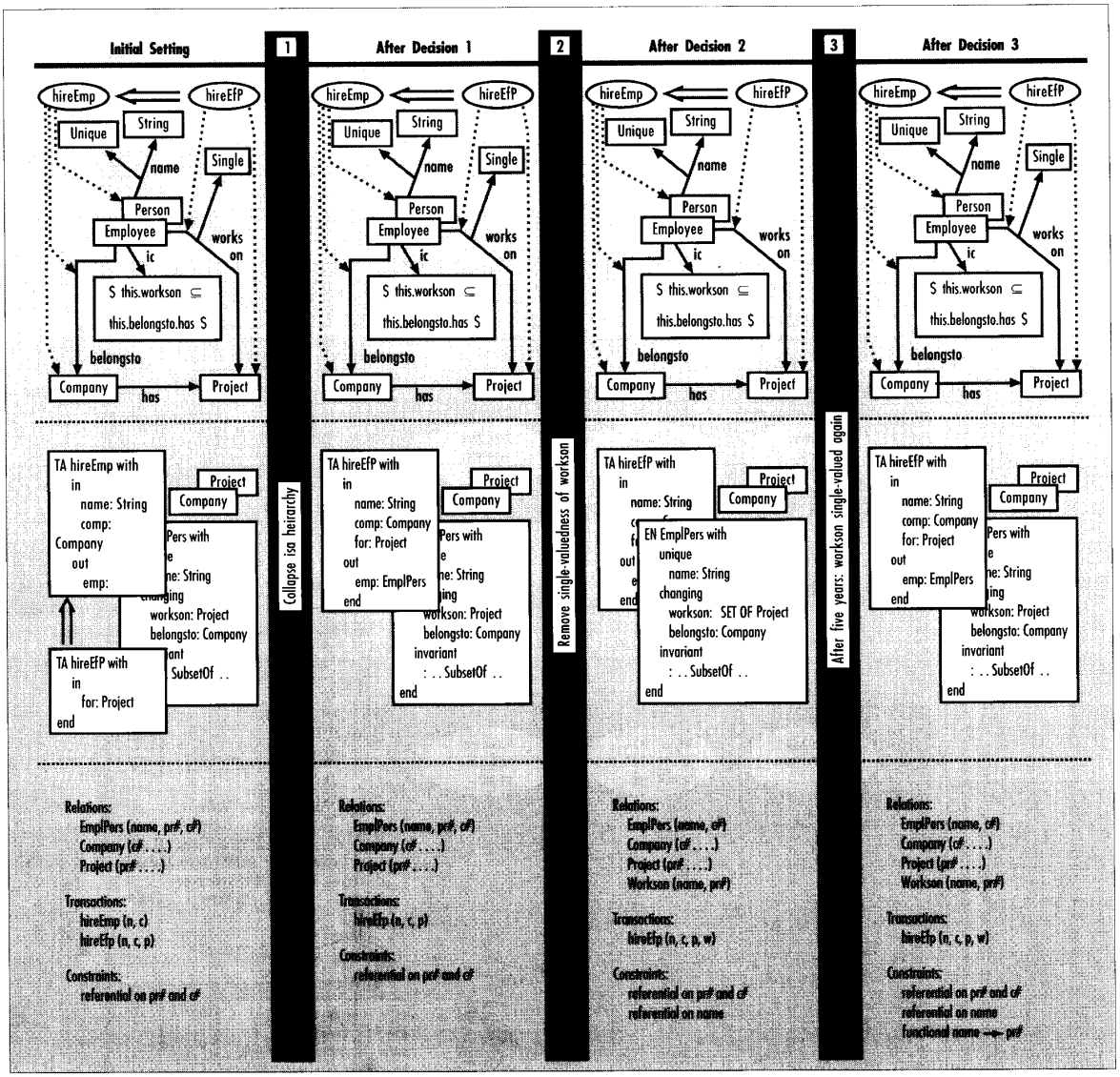

Figure 3. Multilayered development history of a personnel information system in DAIDA.

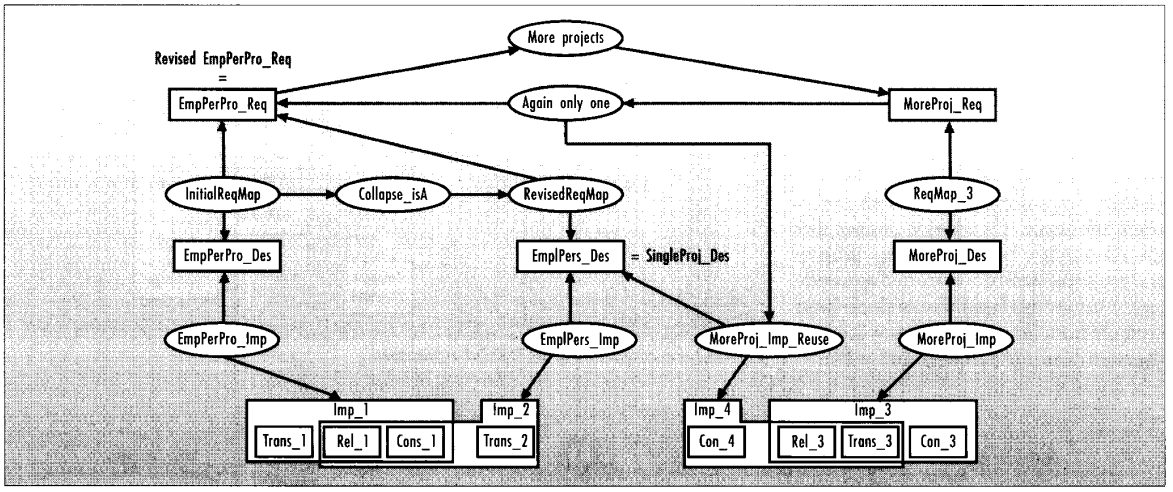

Figure 4. *Detailed dependency structure created by mapping the sample constraint.*

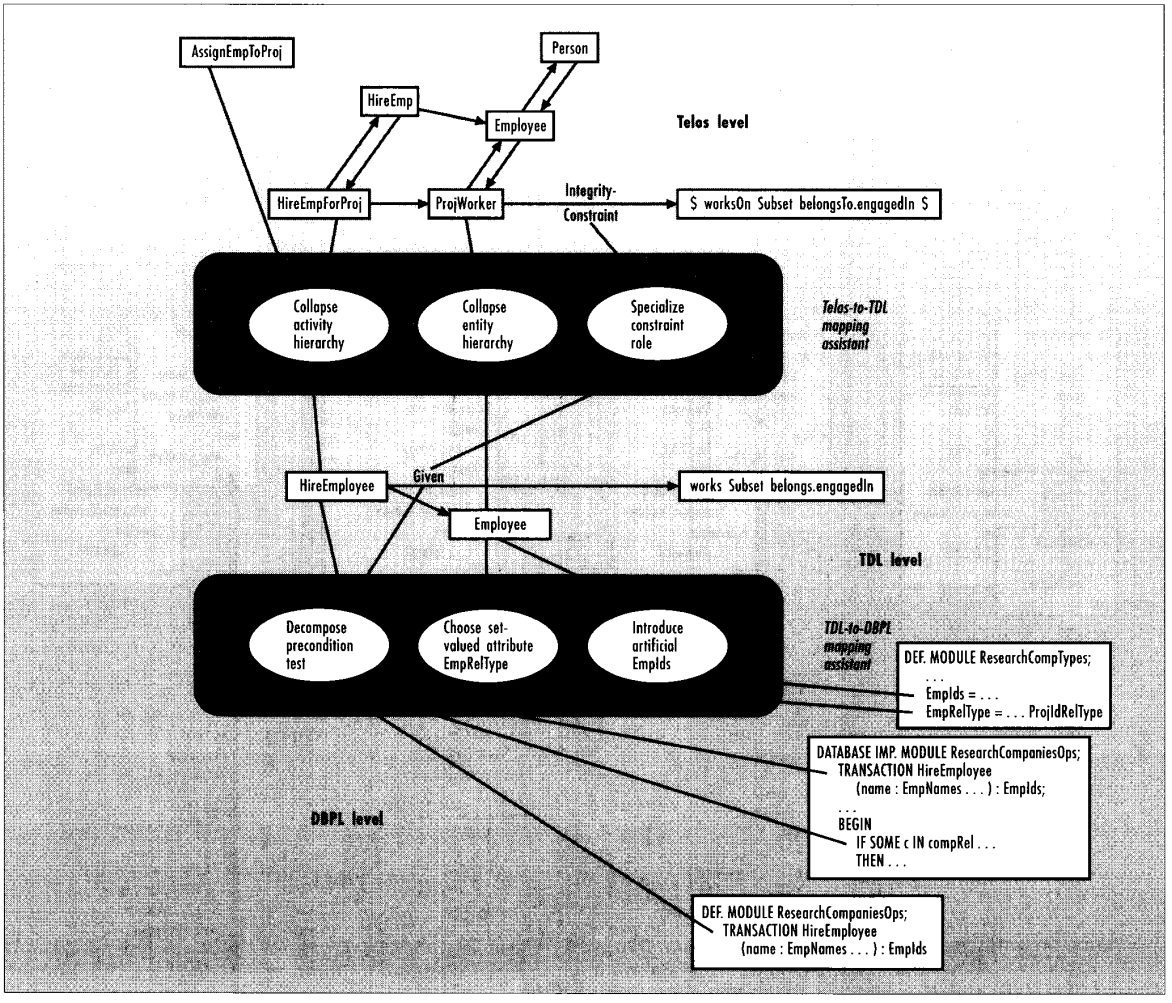

Figure 5. *In-the-large view of the sample development history.*

After the developer makes these revisions, the information system is filled with data. Its structure becomes a major factor to take into account in later requirements changes. When the company decides that employees should concentrate on one project again, the developer cannot simply return to the state after Decision 1. Instead, he creates Decision 3 to preserve the existing implementation and adds only the integrity constraint that each employee can be assigned to at most one project in the Workson relation. He does not have to change any transactions because the database-management system's integrity checker will verify the correctness of transaction results automatically.

Dependencies maintained by ConceptBase support this reuse of previous development experience. Figure 4 shows the detailed design record associated with the mapping of the sample Telos constraint in Decision 2. As the figure shows, however, the details in the set of dependencies to be maintained can easily overwhelm the user.

Fortunately, by simply introducing a class of configuration decisions, we can extend the DOT decision concept to programming in the large.[8] Figure 5 gives a more abstract, process-oriented representation of the history shown in Figure 3. The history of design decisions becomes much more understandable now, even compared with the verbal description.

Integrating CASE environments is a problem with many facets. DAIDA has attempted to deal with these facets by letting you conceptually integrate information systems from the application, system, and implementation perspectives. The perspectives themselves are integrated through recorded design decisions and resulting dependencies.

The principle underlying our approach is to lift integration from files or documents to a conceptual level, taking care of lower level issues with mapping assistants. The specific choices of languages and tools in the DAIDA experiments are offered as examples, although they do seem to exhibit some general properties needed for integration.

Our approach relies on the availability of database technology that can handle conceptual models. Rule/constraint compiler technology provides an automatic mapping from the specification level to the implementation level, which not only generates the necessary code but also ensures that process traces in the form of dependencies are maintained for reuse. Recent results in the database literature indicate that these technologies are moving from research labs into industrial practice.

Working with integrated CASE environments is a team effort that involves developers, managers, and outside stakeholders. A software information system like ConceptBase can be seen as a communication and collaboration medium rather than as just a data store. This was one reason we included subject, usage, and development worlds in the conceptual model. With this approach, negotiation about quality criteria (so-called nonfunctional requirements), work organization, progress monitoring, and the adequate distribution of access rights to development data become important aspects of CASE integration. We are working on extending ConceptBase in these directions as well as on applying our method to different domains like computer-integrated manufacturing. ♦

ACKNOWLEDGMENTS

This work was supported in part by the Commission of the European Communities under ESPRIT contract 892 (DAIDA) and by the Deutsche Forschungsgemeinschaft in its program Object Banks for Experts (grant Ja445/1-2). The DAIDA team includes the software houses Bim in Everberg, Belgium (Raf Venken); Groupe Française d'Informatique in Nanterre, France (Alain Rouge); and Scientific Control Systems in Hamburg, Germany (Raines Haidan); as well as the Forth Computer Research Center in Iraklion, Greece (Yannis Vassiliou); and the Universities of Frankfurt (Joachim Smith) and Passau in Germany.

I am also grateful for the collaboration with the National Science Foundation's Remap project at New York University (Vasant Dhar) and with John Mylopoulos of the University of Toronto.

REFERENCES

1. M. Jeusfeld and M. Jarke, "From Relational to Object-Oriented Integrity Maintenance in Deductive Databases," *Proc. Int'l Conf. Deductive and Object-Oriented Databases*, Springer-Verlag, Heidelberg, 1991, pp. 460-477.
2. J. Mylopoulos et al., "Telos: Representing Knowledge about Information Systems," *ACM Trans. Information Systems*, Oct. 1989, pp. 327-362.
3. A. Borgida et al., "Support for Data-Intensive Applications: Conceptual Design and Software Development," *Proc. Workshop Database Programming Languages*, 1989, Morgan Kaufmann, San Mateo, Calif., pp. 258-280.
4. M. Jarke, M. Jeusfeld, and T. Rose, "A Software Process Data Model for Knowledge Engineering in Information Systems, *Information Systems*, Jan. 1990, pp. 85-116.
5. C. Potts and G. Bruns, "Recording the Reasons for Design Decisions," *Proc. Int'l Conf. Software Eng.*, IEEE CS Press, Los Alamitos, Calif., 1988, pp. 418-427.
6. P. Constantopoulos et al., "Software Information Base — A Server for Reuse," tech. report, ESPRIT Project ITHACA, Forth Computer Science Inst., Iraklion, Greece, 1991.
7. L. Chung et al., "From Information Systems Requirements to Designs: A Dependency-Based Mapping Framework," *Information Systems*, Oct. 1991, pp. 429-462.
8. T. Rose et al., "A Decision-Based Configuration Process Environment," *Software Eng. J.*, Sept. 1991, pp. 332-346.

Matthias Jarke is a professor of information systems at the Technical University of Aachen, Germany. He leads the Knowledge Bases group in ESPRIT's Basic Research Action on Computational Logic as well as several projects in team support. His research interests include extended data and knowledge bases for process-oriented integration of information systems in design environments and business organizations.

Jarke holds a Dr.rer.pol in information systems from the University of Hamburg, Germany. He has published widely on databases and their applications and is on the editorial board of *IEEE Transactions on Software Engineering*.

Address questions about this article to Jarke at Informatik V, RWTH Aachen, Ahornstr. 55, 5100 Aachen, Germany; Internet jarke@informatik.rwth-aachen.de.