Tool Integration in Evolving Information Systems Environments*

Matthias Jarke, Hans W. Nissen, Klaus Pohl

Informatik V, RWTH Aachen
Ahornstr. 55, 52056 Aachen, Germany
{jarke, nissen, pohl}@informatik.rwth-aachen.de

Abstract. Evolution is a fact of life in information systems. Not only systems evolve but also their development processes. IS environments must therefore be designed for accommodating and managing change. The management of process meta models in repositories is one important step; we show how process traceability models and process guidance models can be developed and related in a standard repository framework. In addition, the currently available tool integration along the presentation, data, and control perspectives have to be augmented for process integration. In our process-adaptable and interoperable tool concept, tool behavior is directly influenced by the process guidance model and automatically traced according to the traceability model. The approach is demonstrated with a prototype requirements engineering environment developed in ESPRIT project NATURE.

1 Information Systems and the Process Improvement Paradigm

Information systems (IS) are networks of humans and information technology components intended to serve some organizational purpose. It is widely agreed that next-generation IS will have to be better aligned to organizational goals and processes.

Business processes and goals change even more rapidly with a more turbulent and globalized environment. The management of change therefore becomes a critical issue also for information systems. Current reality is studied by reverse analysis, creating a conceptual model. A desired change is defined in terms of this model. This change must be propagated to the level of the real system by a partial re-implementation process, in a careful trade-off between accomplishing the change goal and blending into the existing context. A typical re-engineering step is shown in figure 1.

Today's situation is characterized by the presence of many IS components and accumulated experiences. On the positive side, products, models, and processes gained from these experiences can be reused. On the negative side, this valuable,

* This work was supported in part by the Commission of the European Communities under ESPRIT Basic Research Project 6353 (NATURE).

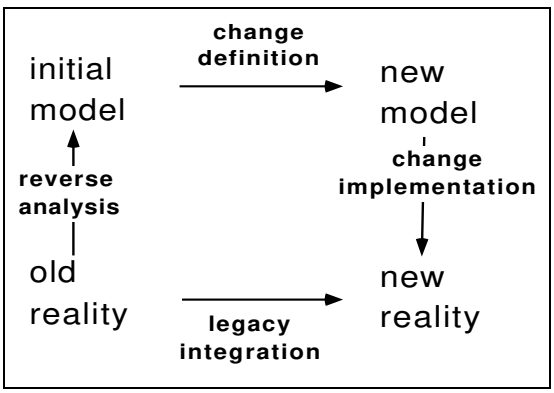

Fig. 1. A change management cycle for information systems

but often dated legacy forces us to integrate multiple generations of systems with vastly heterogeneous intents, technologies, and organizational environments.

Next-generation IS development environments will have to consider evolution of products to be developed (object evolution), of development methods and processes through which they are developed (process evolution), and of the tools used in those processes (tool evolution). In the database and software engineering areas, many people are investigating object evolution; we shall therefore focus on the other two aspects.

Due to the heterogeneous nature of the development environment, and the closer cooperation of systems, consideration of international standards is becoming increasingly important. In section 2, the NIST/ECMA reference model for tool integration is briefly reviewed as a background for our work.

In software engineering, a number of formalisms for process definition have been proposed. A few of them also consider aspects of customization and evolution [4]. To a large degree, this work has been stimulated by the Software Engineering Institute at Carnegie Mellon University (SEI) [9], [5] which associated the maturity of a software development organization with the ability to understand, define, and manage its processes. In previous work [21], we have shown that the SEI maturity levels can be characterized through a *process-oriented IS repository*. Section 3 reviews this work and discusses the relationship between tools and repositories according to a view concept.

The main contribution of this paper, described in section 4, concerns the role that *IS development tools* can play in the change management scenario. The motivation for this work is twofold.

First, capturing process knowledge according to the above-mentioned models (process *traceability*) can be very labor-intensive and costly, and should therefore be automatically done by the tools.

Second, current CASE tools tend to have a fixed functionality whose relationship to the process has to be determined by the development engineer. What we need, are tools that can adapt their behavior to the process even while they are

running. In section 4, we develop a concept for process-adaptable interoperability of tools that satisfies these requirements to a large degree.

Our approach is demonstrated in a prototypical requirements engineering environment, developed in ESPRIT Project NATURE[2]. As requirements processes are poorly understood and existing methods tend to be woefully incomplete, process adaptability and traceability as a basis for learning and improvement are particularly relevant. Major lessons learned are summarized in section 5.

2 Background on Tool Integration

The integration of development tools is a long-standing topic in the CASE literature, culminating in the NIST/ECMA reference model shown in figure 2 [3]. This so-called "toaster model" distinguishes four aspects of integration:

- *Data integration* concerns the ability of tools to exchange data with little conversion effort and a high degree of consistency;
- *Control integration*, supported by message services, concerns the sharing of functions among tools, i.e., the ability to control each others behavior;
- *Presentation integration* offers the user of a tool a uniform interface to all tools;
- *Process integration* concerns the cooperation of tools with respect to the steps, events, and constraints of a process definition.

Detailed requirements for each kind of integration have been explicated in [28].

Most progress has been made in the field of data integration. In fully integrated closed environments, deep integration can be achieved through detailed modeling of document structures (e.g., in IPSEN [17]), whereas open environments such as DECs CDD/Repository have to rely on tool envelopes to achieve a more modest data integration.

In control integration, the idea of message servers or software buses to support message exchange among tools has proven quite successful, e.g. in Hewlett-Packards SoftBench system or in the Eureka Software Factory.

For presentation integration, there exist a number of standard platforms with associated style guides, such as X11 and OSF/MOTIF, although further improvements remain desirable.

Most authors agree that little is known about process integration [30]. There exist a number of process modeling facilities, along with process drivers to enact the models, and with monitoring functions to supervise the execution (cf., e.g., [15]). However, little is said about the assumptions such models make about the tools they work with, how process tracing, evaluation, and evolution are supported.

We have found that the goal of process integration has implications for all the other three kinds of tool integration. Therefore, the next sections look in some

[2] NATURE stands for Novel Approaches to Theories Underlying Requirements Engineering [13].

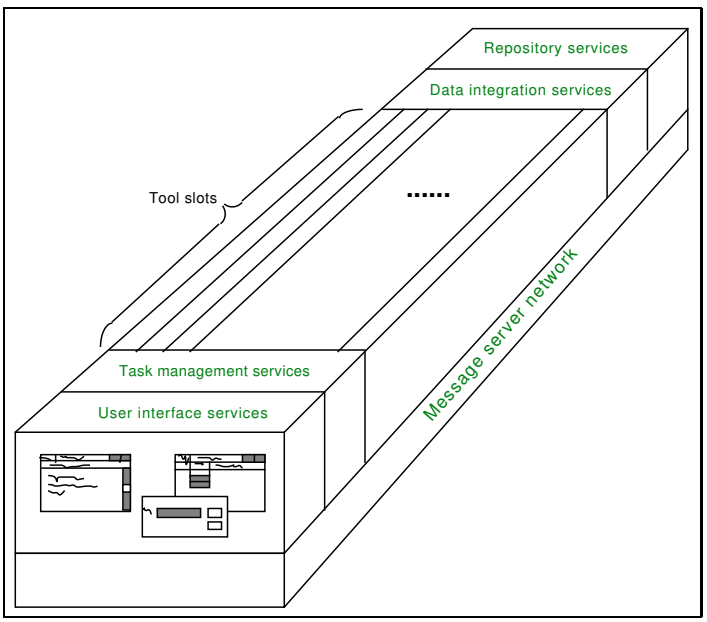

Fig. 2. The NIST/ECMA reference model for tool integration

detail at extensions to data integration (via repositories) and control integration (via communication servers) required for process integration. There are also implications for presentation integration, such as group awareness facilities, but we shall elaborate them here only indirectly, through the discussion of view mechanisms for repositories in section 3.4.

3 Process Repositories: Extending The Data Integration Perspective

In this section, we present the repository concept required to maintain process knowledge, and illustrate it with models used for guiding and tracing requirements engineering processes in the NATURE project.

3.1 The IRDS Standard

Meta information systems, also called data dictionaries, information resource dictionaries, or repositories are information systems about information systems. We shall use the term repositories which was popularized by IBMs AD/Cycle initiative [24]. More precisely, we shall start from the Information Resource Dictionary Standard (IRDS) as recently defined by ISO [10]. Figure 3 sketches this framework.

ISO IRDS is organized along the classification abstraction of semantic data models. In a layered type universe of four levels, level *n+1* (called the *defining*

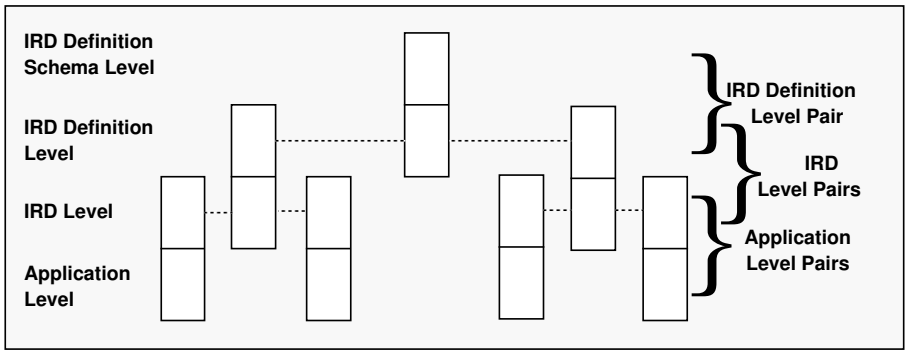

Fig. 3. ISO-IRDS repository framework

level) constitutes a type system for level n (the *defined* level). In other words, it defines the language in which level n can be specified. In increasing order of abstraction, the four levels are the *Application Level*, the *IRD Level*, the *IRD Definition Level*, and the *IRD Definition Schema Level*.

ISO IRDS additionally promotes a so-called "level pair" architecture in which the relationships among the four levels are defined by a hierarchy of three SQL database specifications on top of each other. *Application Level Pairs* correspond to the actual applications to be integrated. Dictionaries or *IRD Level Pairs* contain as their data the schemata of the application level pairs, and definitions of the underlying data models as their schemata.

A single *IRD Definition Level Pair* manages descriptions of the dictionary schemata under a fixed schema, defined in the standard by a set of SQL tables, together with some basic features of the dictionary level pairs.

3.2 Process Repositories

Due to the level pair concept, application systems as well as dictionary systems can be distributed and heterogeneous. Looking at figure 3 in a different manner, we can see two *application environments*, each coordinated by a dictionary. This is the environment available at system runtime.

Overlapping with the application environments, we also see a *development environment*, consisting of the *IRD Definition Level Pair* and the two dictionaries. This is where the development process is managed. To be consistent with the application systems coordinated by the dictionaries, the development environment must, of course, contain information about the products of IS development – the application data structures, programs, interfaces, and maybe manual operating procedures.

However, the development environment must also contain information about intermediate results, human agents and tools involved, process plans, design decisions, and steps taken to execute them. In short, the product respository must be extended to a process repository.

Ignoring for the moment the distributed nature of the development environment, figure 4 shows what kind of process information is defined at what level of the IRDS hierarchy.

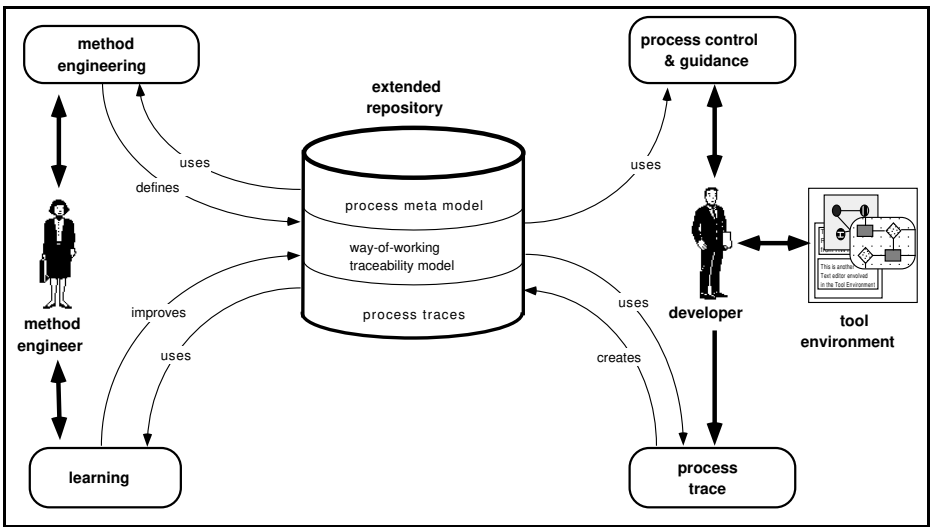

Fig. 4. Process repositories and process-related activities

The *IRD level* (the lowest in figure 4) stores the trace of the actual process; note that part of this trace are descriptions of the product at the schema level (DB schemata and program code, for example).

The *IRD Definition level* defines the actual way-of-working (guidance model), but also the record structure (traceability model) according to which the process trace is to be organized. Note that both structures should not necessarily be the same, as we want to detect discrepancies and use them for process management and improvement. However, the two structures must at least make the actual trace (organized by the traceability model) and the intended trace (defined by the way-of-working) comparable. Traceability model and way-of-working are therefore defined under the same meta process model which is fixed at the *IRD Definition Schema level*, supposedly as a proposed extension to the IRDS standard.

In [21], we have shown how process repositories can be naturally linked to the so-called SEI process maturity model. The Software Engineering Institute at Carnegie Mellon University claims that the maturity of an IS development organization can be characterized through the way the process of gradual system evolution is managed [5]. SEI assesses process maturity by questionnaires followed by in-depth interviews and inspections; we are more interested in the specific repository implications of their model. Briefly, each such level can be

associated with the different kinds of actions on the process repository, as shown in figure 4.

At the *initial level*, this process is ad-hoc and no organizational learning takes places from one change cycle to the next. This corresponds to the case where there is no process repository.

At the *repeatable level*, the organization understands sufficiently well what it is doing to have repeatable processes. Technically, this can be supported by capturing process traces according to some record structure and making them accessible for reuse. This is indicated in the lower right of figure 4.

At the *defined level*, process definitions (ways-of-working) can be abstracted from these collected experiences and enforced through process guidance mechanisms (upper right of figure 4).

Once this has been established, the organization can introduce procedures for measuring the actual process execution against the defined process, thus promoting continuous organizational learning of new processes – the *managed level* (lower left of figure 4).

Finally, there may be also an *optimizing level*, in which continuous goal-oriented and measurement-based optimization of process plans takes place. The repository will, in this case, have to be enhanced with corresponding decision support tools. A new group of people, the process engineers, come into play, which changes the process definition (way-of-working), as shown in the upper left of figure 4.

3.3 Requirements Process Models in NATURE

The process meta model and its specializations for defining traceability and guidance are strongly influenced by the domain in which the process operates. In this subsection, we briefly sketch the models we have designed in NATURE for the task of requirements engineering.

Requirements engineering is a poorly understood field, in which no comprehensive process model can be expected. In such unstructured settings, it is known that humans react with situated behavior and cooperative decision making [27]. The NATURE process meta model (cf. figure 5) therefore comprises concepts of:

- *actions* which transform input *objects* to output objects, and can be composed of sub-actions organized, e.g., as nondeterministic finite automata;
- *situations* in which certain actions are applicable, and which can be characterized by views on objects, on the process state, a combination of both, or simply a perception of the application engineer;
- *decisions* taken by *agents* reacting to specific situations according to certain *arguments* that help with the choice among actions applicable to that situation.

Situations and related decisions together form the process *context*.

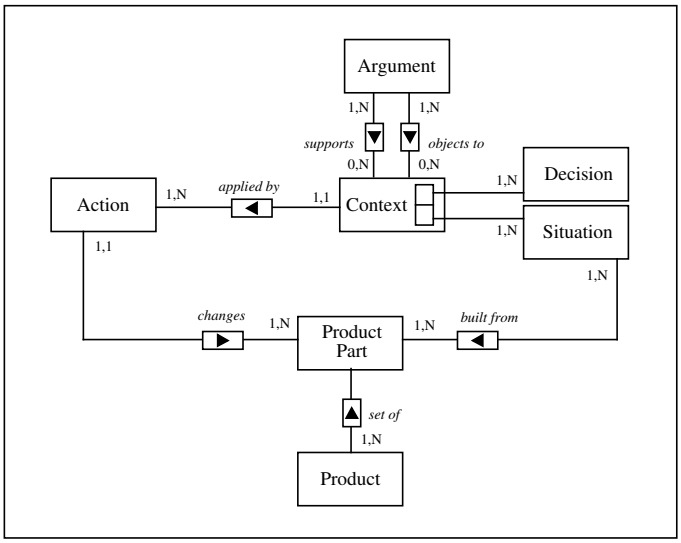

Fig. 5. NATURE process meta model (cf. [25])

Process Guidance Model. The process guidance model (cf. [25]) contains a set of reusable process chunks. Each chunk is described by a situation pattern in which it applies, the possible action patterns known in this situation, and arguments for guiding the choice among them. [3] A chunk can be invoked when its situation pattern is instantiated.

Process guidance can happen at multiple levels of granularity, ranging from long range project planning issues to detailed technical transformations. Firstly, some contexts may be *compound* from more *atomic* ones. Secondly, some contexts are directly applicable by performing product transforming actions while others may be applied in several alternative ways, thus requiring a choice from the developer. Direct action contexts are also called *micro contexts*, contexts with user choices are called *macro contexts*.

As a simple example, consider a requirements engineer working on the refinement of an entity relationship (ER) diagram. The way-of-working for ER contains, for example, a context associating a situation consisting of a pair entity type/relationship type and a decision to "improve" it. As there are several ways to improve entity relationship groupings, this context is a macro context. The process chunk thus relates it to several other contexts that are its alternatives, along with arguments that support the choice between these alternatives.

One of these alternatives is the context that "subtypes" the entity type. The "subtypes" context is a compound context as it is built from more atomic

[3] In the context of process improvement, these arguments can be seen as abstractions from arguments underlying important design decisions in the traceability model from which a chunk has been gained [12].

ones, including the creation of a sub entity type, of the isA relationship between the sub entity type and the initial entity type, and possibly the addition or re arrangement of attributes.

Although very simple, the example shows how the concepts of our approach can be used to precisely define ways-of-working. This can be done both at a very low level of granularity (as in the example) or at a more macroscopic one, such as defining the organization of project inspections in certain critical project situations.

Process Traceability Model. The traceability model (structure of the design record) is organized by three dimensions along which progress in the requirements process is needed [20]:

- The *specification dimension* records the degree of understanding achieved, compared with a model of the international requirements engineering standards.
- The *representation dimension* records progress from initially completely informal representations to more formal and verified requirements models, according to formalizations of hypertext structures, semi-formal graphics, formal knowledge representations, and their possible process interrelationships ("derived from", "explains", etc.).
- The *agreement dimension* records progress from different individual viewpoints to shared understanding and goals, according to a decision-oriented adaption of the IBIS model similar to the one chosen in REMAP [22].

The actual traceability model relates the three dimensions to the process meta model, to give a structure for the trace. Decisions are views a group of agents takes on an argumentation space within the IBIS model of the agreement dimension. Actions are traced by recording (a) the products obtained, (b) the dependencies from existing products to old ones along any of the three dimensions, (c) the agent or tool actions creating these dependencies, plus some other annotations (e.g., criticality of the dependency). To achieve uniformity, decisions are treated like other objects when it comes to traceability, i.e., they depend on certain input objects and other objects may depend on them.

In a practical sense, the traceability model is supported by specific tool suites for each dimension, and by some generic tools for browsing products and dependencies. For example, a flexible hypertext editor for informal requirements, semi-formal graphical tools for ER and SA, and interfaces for the formal knowledge representation language Telos have been developed for the representation dimension, along with a set of dependency types that characterize the possible relationships between such objects. For the agreement dimension, a tool for recording informal argumentation histories according to the IBIS model has been developed, and a group decision support tool for actually making the choices is under development. Along the specification dimension, tools mainly support reuse of generic domain models or of abstractions of existing instances [13].

3.4 Tool Views

To formally describe and technically implement the relationships between tools and repositories, view concepts are an obvious candidate. But what is the view and what is the database?

Goals to consider when making this choice include consistency of the database, awareness of change across tools within the whole development team, autonomy and coordination of users, and other factors. In our prototype, we have initially followed the usual approach where tools have views on a knowledge base. As the experiences were not uniformly positive, we also briefly discuss alternatives we are currently evaluating.

Our Approach: Tools Have Views on the Repository. Like [14, 23], the NATURE environment uses a central repository in client-server architecture (the ConceptBase system [11]) as the only information store. Tools do not keep any persistent knowledge locally. Typically, a client connected to the database is only concerned with a very small part of this information, possibly structured in a different way – it has a specific view on the global information. Such a view is realized as virtual information which is not explicitly stored in the database but can be computed when needed. An integration of the different views is not necessary because all are derived from the same (global) representation of the world. This enables the application of sophisticated integrity checking techniques [19, 2] for global and local constraints. The tools access the database (i.e., ask and tell) only through their specific view specification which prevents other parts of the database against uncontrolled access.

Views are specified like any other query, but may be parameterized. Updates on explicitly stored information in a database often affect intensional knowledge defined by views. If the view is materialized or in use by a tool, the content of the view must be recomputed in order to capture the consequences of this update. We use incremental view maintenance techniques to propagate updates on base data to updates of a view, i.e. the data used by a tool [8, 29].

The reverse direction, in which updates coming from the tools must be translated to updates on the underlying base data, is much more complicated. In general, there exists no one-to-one mapping. There may be zero, one or many possible translations. An unique translation is only possible for a very restricted number of views. To implement a (partial) translation, the ConceptBase system uses the concept of query classes to specify views in the conceptual modeling language Telos [16] and translates them into event-condition-action rules [26].

Experiences and Alternatives. We have chosen the above approach because of the deductive nature of our repository ConceptBase and the availability of well-understood and efficient algorithms for problems like integrity checking and view maintenance. Our experiences have shown that this approach is very suitable for display tools but has difficulties when updates have to be propagated across tools which do not look at the repository with the same structural organization.

We now briefly describe two alternatives where in the first one the tools have a distributed database on which the repository maintains a view. In the second, the repository is fully distributed across tools (then called viewpoints rather than views) and there is no repository at all.

The Repository Holds Views of the Tools. In [1] the authors describe a way how to manage the coexistence of tools which locally organize their data and a database also representing the data of the tools. They avoid redundant storage and therefore the maintenance problem by using a database not containing the data itself but only a view on the tools. The database extracts the data from the tools (resp. the file) only when needed, e.g., for query evaluation. One can look at the database as an abstract interface to the information stored in the tools while offering standard database facilities. The integration of the different views enables the handling of restricted updates on the database (view updates) and updates on the internal tool data (view maintenance). Then it is also possible to use the database as a medium for tool interoperability in the sense that they can exchange information if their views overlap.

The responsibility of every single tool to keep its own data consistent seems to be a main disadvantage of this approach, because this leads to the implementation of integrity checking techniques in every tool. Only if the views are materialized is it possible to use facilities offered by the connected database for this task. But this contradicts the original approach.

Tools Represent Independent Interrelated Viewpoints. The Viewpoint-Oriented Systems Engineering (VOSE [7]) framework proposes the use of largely distributed, locally managed viewpoints as a way of managing the software development process. In contrast to the above approaches, there exists no common database or repository to support the viewpoints. The global perspective on the development process is given by the collection of all involved autonomous viewpoints. They may overlap, complement and contradict each other. Even though absolute consistency can be an unnecessary restriction of the individual development strategy, it is necessary to keep the specifications consistent in some way. This requires constraints satisfied by all viewpoints. The missing common description of all perspectives in, e.g., a repository, makes the check and enforcement of such constraints complicated.

The viewpoint approach has a number of very attractive features for cooperative work in unstructured application domains but is still in its early phases.

4 Process-Adaptable and Interoperable Tools: Extending the Control Integration Perspective

Looking again at figure 4, we have so far discussed models from which the requirements engineering team can obtain guidance for their process, and models into which they can document their activities. This approach is also roughly followed in other recent proposals for process-centered environment, albeit with different process models (e.g., [15]).

The problem is that this solution involves two man machine interfaces: one for guidance, one for tracing. The former creates a reliability problem since it is not clear whether the requirements engineering team will actually see and follow the guidance. The latter creates a workload problem since a detailed trace requires a lot of traceability data which are very inefficient to record manually.

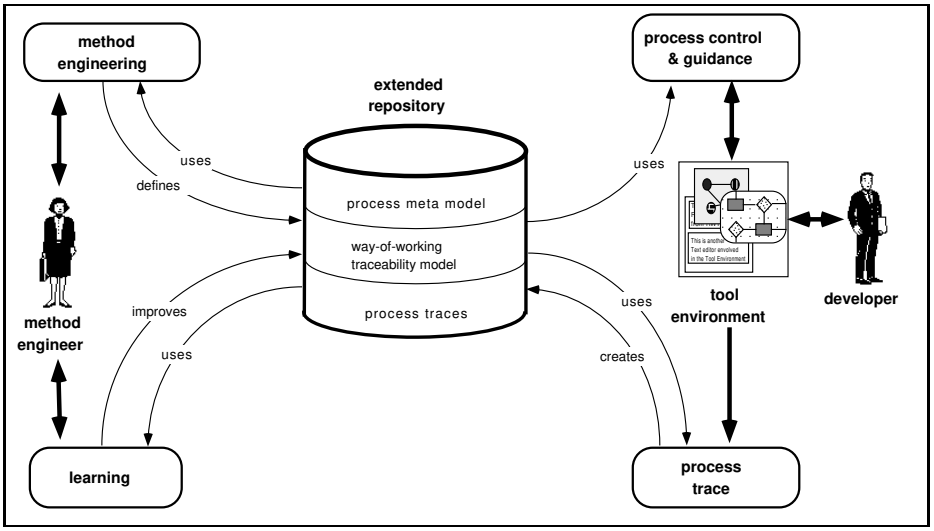

Fig. 6. Integration of tool environment and process repository

We therefore aim at offloading much of this work to the CASE tools available in the environment (cf. figure 6). CASE tools should be directly put into the right context by the process guidance model. They should also trace their own activities in the repository automatically according to the process traceability model. Bringing process guidance and traceability into the CASE tools is particularly demanding, as we cannot expect a fixed process model. Thus, tool behavior must adapt itself to changing process definitions in the process improvement scenario, and traceability should address the relationship of actions not only to the basic traceability model, but also to the guidance situation.

We achieve process adaptability through modeling tools as composite objects, and support process enacting and traceability through a communication server for coordinating short term interoperability of tools. Both are described in the sequel and illustrated with a scenario from the NATURE prototype, extending the entity relationship example from the previous sections.

4.1 Tool Models and Implementation

In our approach, tools have several similarities to objects in languages such as SmallTalk but there are also some important additional aspects to be considered.

It has already been recognized in the Eureka Software Factory project [6] that, as a minimal requirement for adapting tool behavior to processes, tools have to be composed of *individual services* which can be made available and traced independently of each other. At the user interface level, these services are often shown as menu items.

It is also common practice that tools can operate in *different modes* depending on the situation. A mode offers a particular subset of services. A simple example is a *Read only* mode in a text editor in which no change services are offered. Modes and mode combinations can, for example, be supported by selective bitmap masks on the menu services offered. For process adaptability, it is important that such masks can be defined and activated from *outside* the tool.

To set the working context right, it must be further possible to *parameterize* these process adapted tool invocations with a specific product model and focus within that model, typically gained from previous process steps. An example would be a *Refine* mode in an entity-relationship editor which only allows variants of refinement operations, and only on one particular entity of interest.

The definition of the service implementation may include choices and the invocation of sub-services from the same or other tools, without losing the *working context*. Thus, services must be enabled to send messages and maintain their state until responses to these messages have been received.

If all these conditions are satisfied, a specialized tool definition consists of a set of offered services, an appropriate object context, and service execution through choices and compound, multi-tool operations – all of which can be influenced from outside. *This corresponds exactly to the definition of a process chunk in our guidance model!* New tools can therefore be integrated into the repository by simply defining the appropriate process chunks for their available modes.

Conversely, by defining a process chunk composed from the basic services and object types available in the tools and in the repository, we can construct a specialized tool environment for exactly this process chunk. We have thus reached the goal of process-adaptable tool definitions.

4.2 Interoperability and Traceability of Tool Behavior

We now discuss how to make guidance available dynamically to the development environment, and how to execute and trace complex multi-tool process actions. Process-guided and communication-based interoperability is the main pre-requisite, for several reasons.

First, actions are often *nested*. For instance, during the creation of a data store in a data flow diagram, also the corresponding part of the data dictionary must be updated. This causes the need for direct communication between the actions which are provided by a particular tool, not between the whole tools. The "first" action, which leads to the execution of other actions, must be able to control the performance of the others. Figure 7 explains this issue.

Second, actions can be provided by *different tools*. For example, in nested actions, each action which needs another action for assistance, must know by

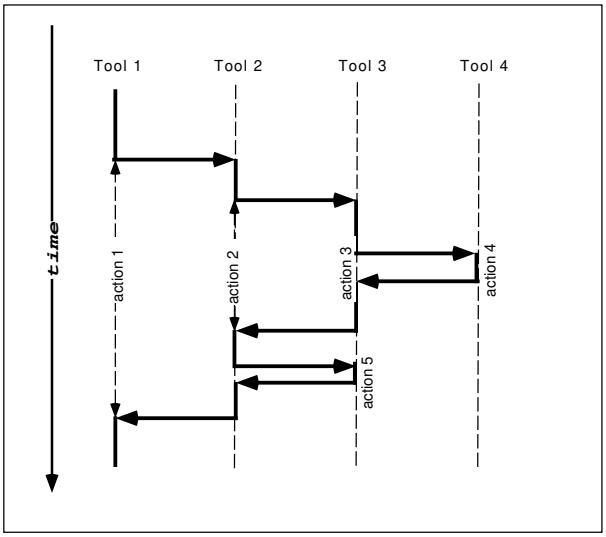

Fig. 7. An example of nested actions across multiple tools

which (set of) tools that action is provided.

Third, actions must know the state of those tools which could potentially offer a required service. The action must know if any tool providing the needed action is currently running and if the "right" objects are loaded. If not, the tool must be started according to the needs, its mode must be reset, new objects may have to be loaded and the user must be focused on the current work task.

Fourth, dependent on the results returned by a sub-service, the further execution of the calling action can be different.

Fifth, for performance reasons, data exchange among tools should not always go through the repository. Especially nonpersistent data should be directly transferred as part of the message. This requires suitable abstraction mechanisms for the product descriptions, such as named aggregates or models which can be referenced by name.

To encapsulate these properties and to insulate the individual tools from control knowledge about other tools, the NATURE environment offers a communication manager (CM) besides the process repository (see figure 8). Similar to Hewlett-Packards SoftBench, the CM controls the communication among currently operational tool instances. That is, it knows (a) which tools are currently running, (b) how they can be accessed, and (c) what situation (mode and objects) they are in (i.e., whether they can provide the required services at present).

This approach has several advantages.

Firstly, the CM hides details of message protocols from the individual tools. Secondly, as all tools are invoked (and tool modes changed) through messages, the CM can easily maintain the status of the process situation at any given point. Last not least, to support process adaptability, the CM also abstracts from the

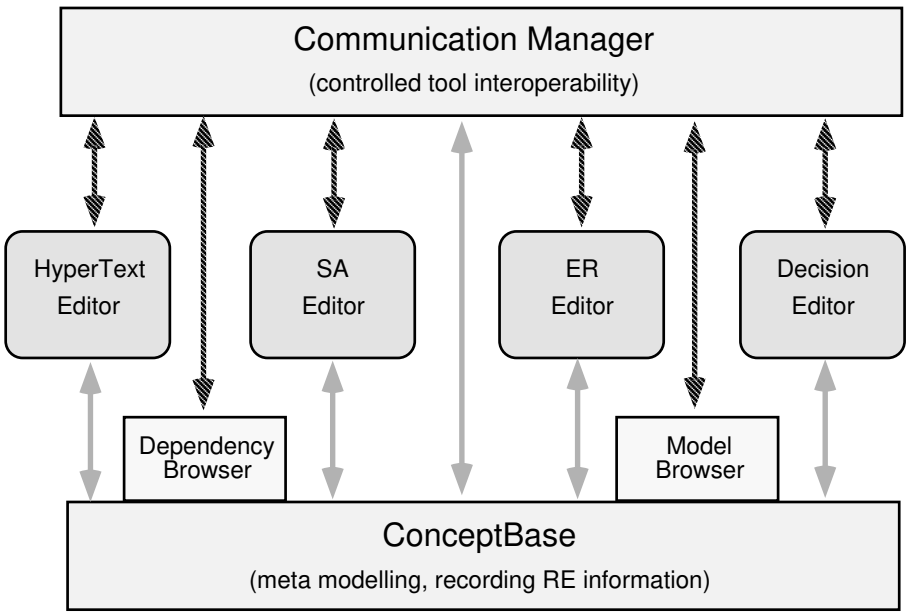

Fig. 8. The NATURE environment

grouping of services into tools. If a tool needs a service performed, it need not know which (combination of) tools offer this service because this information will be brokered by the CM. This is further strengthened by associating default tools for providing a service in particular types of process chunks.

Taken together, we have the following scenario. A service requests a sub-service from the CM. The CM finds the appropriate tool that offers this service, makes it available in the right mode and with the right objects, and restricts its use to the current process state. After the process steps associated to the sub-service have been executed, control is returned to the calling tool. In the current implementation, the calling tool is meanwhile blocked although we are thinking about more pipelined ways-of-working.

The trace depends on the results of nested actions. Each sub-action captures the "operational semantics" in the repository by creating dependency links from its output objects to its input objects or to decisions underlying the action. Often, sources and sinks of such dependencies, their types and annotations, depend on sub-actions by other tools. The compound action model of the process guidance must then be used to create a meaningful trace.

4.3 A Session with the NATURE Environment

To illustrate the tool interoperability approach and the use of the guidance and traceability models in it, we shall consider a requirements engineering scenario as supported by the NATURE prototype.

Assume the user has activated an entity-relationship editor to refine a particular entity. The process model for this activity says that he should inspect the reasons behind the current version of a particular entity or relationship, and possibly update them to justify the refinement.

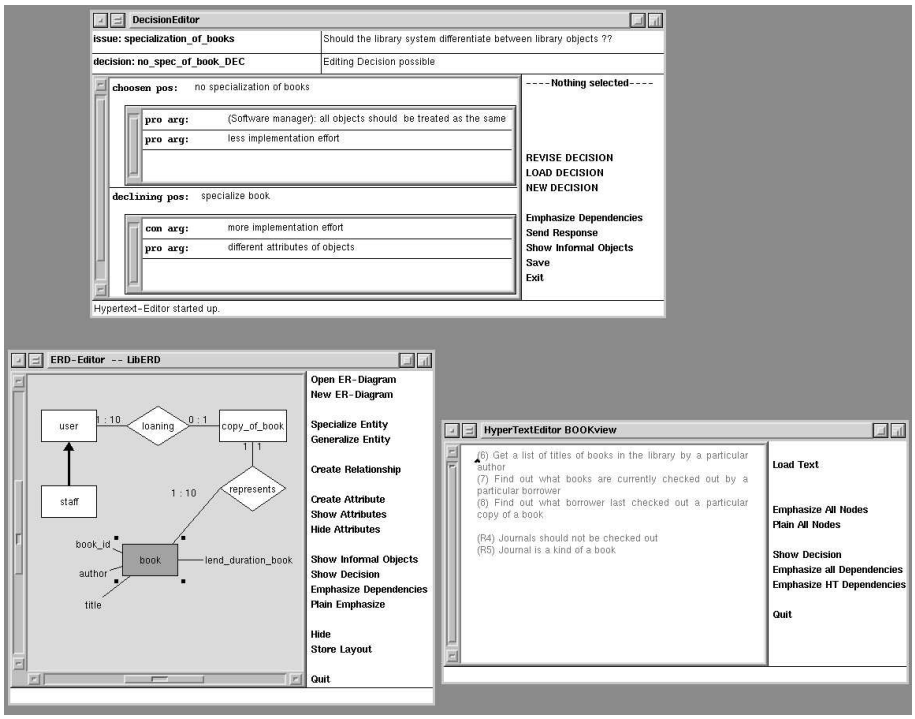

Fig. 9. A screendump demonstrating tool interoperability between entity relationship, decision and hypertext editor

The requirements engineer first selects the entity to be refined and then the menu option *specialize entity*. Due to the process model, the action *specialize entity* asks the communication manager (CM) for the service of showing the objects the existing entity depends on. The CM knows that this service is provided by the dependency editor, but also, from the kind of message received, that the requirements engineer is not really interested in the dependencies themselves. It therefore starts the dependency editor in "silence mode" and gives it the task to trace back the dependencies from the entity in question.

Tracing involves retrieving these dependencies from the repository and displaying the target objects. The CM is able to start automatically tools which can display the target objects. Since there may be very many such dependencies, the process model allows the user to select the objects to be displayed from the set of retrieved target objects, and thus the tools to be activated.

For the selected entity there are two decisions and a hypertext view consisting of five informal objects retrieved as target objects. The user selects the decision *no specialization of books* and the hypertext view. Depending on the type of object, the CM chooses the tools to be started. It opens an instance of the IBIS browser to show the selected decision in the context of its underlying argumentation history. It also opens one instance of the hypertext editor, for displaying the informal objects. Assume that the decision editor is already open; the CM will then bring it to the front of the screen. All these activities are defined in a process chunk based on the situation pattern for the tool invocation from dependencies.

The user decides to revise the decision *no specialization of the entity book*, due to the retrieved informal objects and argumentation of the decision. He selects the action *revise decision* which is offered by the decision editor. The action *revise decision* calls the CM to display the related informal objects to the user – as defined in the process chunk (cf. figure 9).

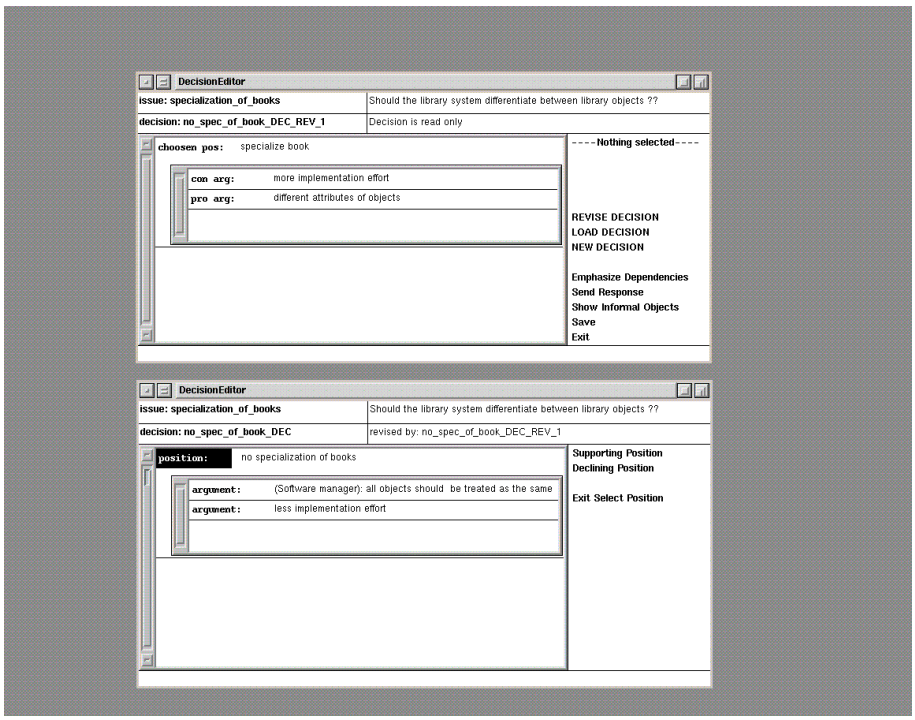

Fig. 10. A screendump demonstrating the revision of a decision

The CM retrieves - again using dependency links - the informal objects and starts automatically the hypertext editor in a predefined mode, the *selection* mode.

The control rests now with the hypertext editor. Within the hypertext, the informal objects, which are related to the decision by dependency links, are automatically highlighted. The user has the possibility to highlight additional parts of the document which he wants to relate to his current action – the revision of the decision *no specialization of books*. After finishing with selection, control returns to the action *revise decision* of the decision editor and the "highlighted" parts of the hypertext are automatically grouped to a view on the hypertext and related to the revised decision.

The user now revises the decision (cf. figure 10) by changing the weighting of the positions (arguments) and adding new ones. After finishing with the revision, the control is returned to the ER-Editor – to the action *specialize entity*. Here, the refinement of the entity can now be actually conducted and is automatically related to the revised decision through an additional dependency link derived from the call history of service invocations.

All of the tool interactions and choices are defined by process chunks in the repository, and the actual trace uses the process chunks as well as dependency types coming from the general traceability model to record the process execution.

5 Conclusions

Developing and evolving complex structures do not just pose a challenge to database management, knowledge representation and reasoning. They also require significant extensions to tools and tool integration concepts in IS development environments. The reasonably well established techniques for data integration, control integration, and presentation integration have to be augmented and carefully combined to achieve comprehensive, cost-effective and improvement-oriented process integration.

The NATURE approach attempts to achieve *comprehensiveness* through a process-oriented extension to the IRDS repository standard, *cost-effectiveness* through tools that are process-adaptable and offer automated traceability, correctness and *group awareness* through a view concept that relates tool and repository models. Improvement orientation is achieved through a quality-oriented process meta model, possibly supported by some machine learning techniques, which is presented in a companion paper [12].

Taken together, these techniques should constitute a significant step forward in managing change in complex information systems domains. Application experiments in the requirements engineering and industrial engineering domains are being conducted to substantiate this claim.

Acknowledgments. We are grateful to our NATURE partners for many discussions and contributions, especially to the group of Colette Rolland at Paris-Sorbonne, for their contributions to the process guidance model. Major portions of the NATURE prototype were implemented by our students Michael Gebhardt, Fred Gibbels, Peter Haumer, Ralf Klamma, Christof Lenzen, Ute Loeb, Klaus Weidenhaupt and Claudia Welter.

References

1. S. Abiteboul, S. Cluet, and T. Milo. Querying and updating the file. In *Proc. of the 19th VLDB Conference*, Dublin, Ireland, 1993.
2. F. Bry, H. Decker, and R. Manthey. A uniform approach to constraint satisfaction and constraint satisfiability in deductive databases. In *Proc. Intl. Conf. Extending Database Technology*, Venice, Italy, 1988.
3. M. Chen and R.J. Norman. A framework for integrated CASE. *IEEE Software*, March 1992, 18–22.
4. R. Conradi and M.L.. Jaccheri. Customization and evolution of process models in EPOS. *Proc. IFIP 8.1 Working Conf. Information Systems Development Process*, Como, Italy, 1993, 23–40.
5. B. Curtis. The CASE for processes. *Proc. IFIP 8.2 Working Conf. The Impact of Computer-Supported Techniques on Information Systems Development*, Minneapolis, Mn, 1992, 333–343.
6. C. Fernstroem, K.-H. Naerfelt, and L. Ohlsson. Software factory principles, architectures, experiments. *IEEE Software*, March 1992, 36–44.
7. A. Finkelstein, J. Kramer, B. Nuseibeh, L. Finkelstein, and M. Goedicke. Viewpoints: A framework for integrating multiple perspectives in system development. *Intl. Journal Software Engineering and Knowledge Engineering*, 2(1), March 1992.
8. A. Gupta, I.S. Mumick, and V.S. Subrahmanian. Maintaining views incrementally. In *ACM SIGMOD Conference on Management of Data*, 1993.
9. W. Humphreys. *Managing the Software Development Process*, Addison-Wesley, 1989.
10. ISO/IEC 10027. *Information Technology – Information Resource Dictionary System (IRDS) – Framework.* ISO/IEC International Standard.
11. M. Jarke, (ed.). *ConceptBase V3.1 user manual.* Aachener Informatik Berichte Nr. 92-17, RWTH Aachen, Germany, 1992.
12. M. Jarke, K. Pohl, C. Rolland, J.-R. Schmitt. Experience-based method evaluation and improvement: A process modeling approach. submitted for publication.
13. M. Jarke, K. Pohl, S. Jacobs, J. Bubenko, P. Assenova, P. Holm, B. Wangler, C. Rolland, V. Plihon, J.-R. Schmitt, A. Sutcliffe, S. Jones, N. Maiden, D. Till, Y. Vassiliou, P. Constantopoulos, G. Spanoudakis. Requirements engineering: an integrated view of representation, process, and domain. *Proc. 4th European Software Engineering Conf.*, Garmisch, Germany, 1993.
14. W.L. Johnson, M.S. Feather, and D.R. Harris. Representation and presentation of requirements knowledge. *IEEE Transactions Software Engineering*, 18(10):853–869, October 1992.
15. P. Mi and W. Scacchi. Process integration for CASE environments. *IEEE Software*, March 1992, 45-53.
16. J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: representing knowledge about information systems. *ACM Trans. Information Systems*, 10(4):325–362, 1990.
17. M. Nagl. Eng integrierte Software Entwicklungsumgebungen. Ein Erfahrungsbericht ueber das IPSEN Projekt. *Informatik Forschung und Entwicklung*, 8(3),1993.
18. NIST. *Reference Model for Frameworks of Software Engineering Environments.* Draft Version 1.5, Nat'l Inst. Standards and Technology, Gaithersburg, Md., 1991.

19. A. Olive. Integrity constraints checking in deductive databases. In *Proc. 17th VLDB Conf.*, Barcelona, Spain, 1991.
20. K. Pohl. The three dimensions of requirements engineering. Proc. 5th Intl. Conf. Advanced Information Systems Engineering, Paris, France, 1993, 275–292.
21. K. Pohl and M. Jarke. Quality information systems: support for evolving process models. Aachener Informatik Berichte 92-17, RWTH Aachen, Germany, 1992.
22. B. Ramesh and V. Dhar. Supporting systems development by capturing deliberations during requirements engineering. *IEEE Trans. Software Engineering*, 18(6):418–510, 1992.
23. H.B. Reubenstein and R.C. Waters. The Requirements Apprentice: automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering*, 17(3):226–240, March 1991.
24. J.W. Sagawa. Repository manager technology. *IBM Systems Journal*, 29(2),1990, 209–227.
25. J.-R. Schmitt. Product modelling for requirements engineering process modelling. *Proc. IFIP 8.1 Working Conf. on Information Systems Development Process*, Como, Italy, 1993, 231–245.
26. M. Staudt, H.W. Nissen, and M.A. Jeusfeld. Query by class, rule and concept. *Applied Intelligence, Special Issue on Knowledge Base Management*, 1994.
27. L.A. Suchman. *Plans and Situated Actions*, Cambridge University Press, 1987.
28. I. Thomas and B.A. Nejmeh. Definitions of tool integration for environments. *IEEE Software*, March 1992, 29–35.
29. T. Urpi and A. Olive. A method for change computation in deductive databases. In *Proc. of the 18th VLDB Conference*, Vancouver, Canada, 1992.
30. A. Wasserman. IDE: Tool integration in software engineering environments. *Proc. SofTec-NRW Workshop CASE – Stand der industriellen Technik*, RWTH Aachen, October 1992.

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