Requirements Engineering: An Integrated View of Representation, Process, and Domain*

Matthias Jarke, Klaus Pohl, Stephan Jacobs, Hans W. Nissen,[1]
Janis Bubenko, Petia Assenova, Peter Holm, Benkt Wangler,[2]
Colette Rolland, Veronique Plihon, Jean-Roch Schmitt,[3]
Alistair Sutcliffe, Sara Jones, Neil Maiden, David Till,[4]
Yannis Vassiliou, Panos Constantopoulos, Giorgios Spanoudakis[5]

Reuse, system integration, and interoperability create a growing need for capturing, representing, and using application-level information about software-intensive systems and their evolution. In ESPRIT Basic Research Project NATURE, we are developing an integrative approach to requirements management based on a three-dimensional framework which addresses formalism as well as cognitive and social aspects. This leads to a new requirements process model which integrates human freedoms through allowing relatively free decisions in given situations. Classes of situations and decisions are defined with respect to the three-dimensional framework through the integration of informal and formal representations, theories of domain modeling, and the explicit consideration of nonfunctional requirements in teamwork. Technical support is provided by a conceptual modeling environment with knowledge acquisition through interactive as well as reverse modeling, and with similarity-based querying.

1 Introduction

In software engineering, the requirements phase has traditionally been perceived as the fuzzy and somewhat dirty part in which a formal specification is gained from informal ideas. Everything prior to the formalization tends to be lost: in theory due to the inability to integrate informal information into the formal specification framework, in practice through burying requirements knowledge in bulky, practically unusable and unmaintainable documentation.

In contrast to this tradition stands the growing demand for "corporate ownership" of information technology. Organizations cannot take full advantage of information systems

---

* This work is supported in part by ESPRIT Basic Research Project 6353 (NATURE).

1 Informatik V, RWTH Aachen, Ahornstr. 55, 5100 Aachen, Germany

2 SISU-ISE, Isafjordsgatan 26, 1250 Kista, Sweden

3 Universite Paris 1, rue de al Sorbonne 17, 75231 Paris, France

4 Business Comp., City University, London ECIV OHB, UK

5 ICS-FORTH, Dedalou 36, 71110 Heraclion, Greece

which they don´t understand and therefore don´t trust. The only way they can understand software systems is in application terms, i.e. from the requirements point of view. It is also from this perspective that they wish to evaluate software and to direct its change.

When presented to users, their managers, and other stakeholders, requirements information should adequately reflect the actual status and ongoing change of existing systems. Thus, consistent relationships between requirements information, system specification, and implementation must be maintained. In an ideal world, these relation–ships would be created during initial development and continuously maintained. In the real world, there are lots of incomprehensible legacy systems; *reverse modeling* is therefore becoming a necessary if difficult complementary technique for requirements capture.

Such new ways to *capture* requirements information are only justified because there are also new *uses* of requirements information. In addition to the traditional forward design, there are:

- explicit and computer-supported *use of requirements*, especially non-functional ones, to drive design decisions in the systems development process;
- *reuse and shared maintenance of requirements models and processes* in the evolution of a system family or in the development of new applications;
- support of reuse and heterogeous system interoperability by standardized *reference models;*
- the *re-engineering* of systems and business processes based on requirements models.

All of these are no longer relevant only to information systems development. They also invade computer-integrated manufacturing, office systems, process control, and other areas where systems interact intensively with their environment.

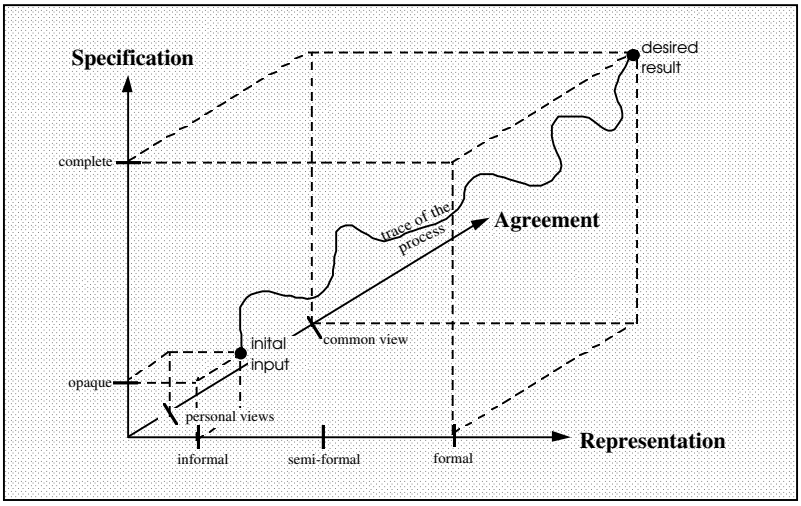

Fig. 1. The three dimensions of requirements engineering.

What does it take to capture, maintain and use requirements information? The NATURE framework [Pohl93] derives three dimensions (cf. figure 1) of interest from the major sources of problems in requirements engineering:

- As a basis for formal system development, requirements engineering moves along a *representational axis* typically from informal to more formal representations, ideally ending with a formal specification that can be transformed to executable code. Modeling and relating these representations is mostly a technical problem albeit one whose solution needs good knowledge of human-computer interaction.
- A second obvious goal is to end up with a complete system *specification*. We see this as orthogonal to the first problem: it is perfectly possible to hide poor understanding behind a lot of formalism, as well as to describe a very deep understanding in quite informal terms. Moves along this axis mostly face cognitive and psychological problems of requirements engineering.
- The third axis is concerned with the *agreement* reached on the current specification. Requirements engineering is understood here as a negotiation process which should lead towards at least sufficient agreement to start building a system. Hence this axis mainly deals with the social aspects of requirements engineering.

At the level of process execution, RE can be understood as a movement in this space which, in addition to the technical, cognitive, and social barriers, is also influenced by economical factors and the methods used. For example, the traditional requirements capture process starts near the origin with different personal views, little system understanding and a completely informal representation; it should end with agreement on a well-understood and formally described specification.

At the level of process definition and evaluation, we can evaluate particular methods or combinations thereof with respect to their contribution to overcoming the problems associated with each dimension.

This paper studies the impact of the views stated above on support for requirements information management. Section 2 argues that a requirements process model cannot be fully pre-planned but should take advantage of situations and human decisions. Section 3 presents, for each dimension in figure 1, partial answers NATURE has developed for the obvious question: what is a situation and what decision options and guidance do we have? Section 4 discusses the needs these models imply for representation and reasoning tools that create and search them. Section 5 sketches an extension of the well-known library example to illustrate the impact of our approach; a more detailed version of this example is found in the NATURE Initial Integration Report [JJP*92] on which this paper is based.

2 NATPROC: A Process Model Based on Situations and Decisions

According to [Dows87], process models can be classified in three categories. *Activity-oriented process models* [Royc70; Boeh88; HeEd90] come from an analogy with problem-solving and provide a frame for manual management of projects developed in linear fashion. This linear view is inadequate for methodologies which support

backtracking, reuse of previous designs, and parallel engineering. *Product-oriented process models* [FKGo90; AHTo90] represent the development process through the evolution of the product. They are more synergetic with systemic methodologies that do not place constraints on the design process. They also allow design tracing in terms of the performed transformations and their resulting products. Finally, the *decision-oriented paradigm* integrates more deeply the semantics attached to evolutionary aspects. The notion of design decision facilitates under–standing of the designer's intention, and thus better reuse of its results [Pott89, RJG*91].

When considered from the viewpoint of requirements engineering, only the last of these approaches appears to be partially appropriate. It is difficult to impossible to write down a realistic state-transition diagram (to cite a popular activity-oriented model) that adequately describes what has to happen or actually happens in requirements engineering. But relying purely on the object history is also insufficient. Even the decision-based approaches offer only limited hints when and how to decide on what. The central idea of the NATPROC process model is therefore that it makes the notion of *situation* (in which to decide) explicit and relates it to the broader question of context handling.

Figure 2 gives an overview of the model. The basic building block of any process is modeled as a triple *<situation, decision, action>* [GrRo90]; a further component, *argument-types* underlying design decisions, is discussed later.

The triple associates the situation the requirements engineer has to deal with to one of the decisions he can take to solve the local problem and to one of the actions to be performed to apply the decision. The record of the selected triplets comprises a trace of the development. Situation perception can be based on activity status, product status, or just subjective impressions of the analyst.

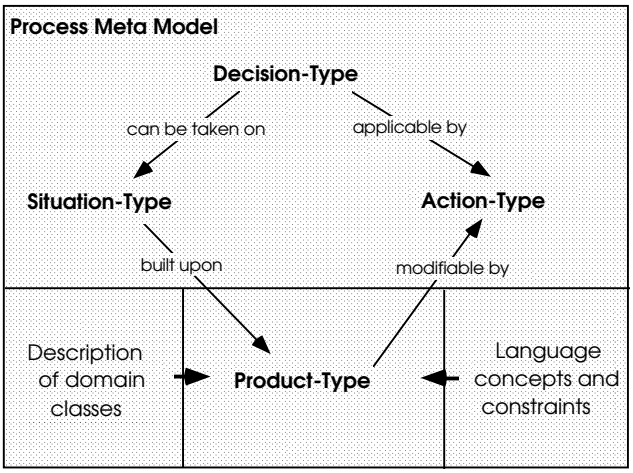

Fig. 2. The NATPROC requirements process meta model.

By refining the interrelated lattices of product, situation, decision, and action types, the NATURE process theory models software development activities at macro and micro levels of detail. Engineering activities deal with the RE artifacts (the specifications) and allow their creation, evolution, modification and deletion. Monitoring activities organise

and order engineering activities, select the appropriate engineering activity for a certain situation, and allocate the necessary resources. Figure 3 shows a partial taxonomy of decision types.

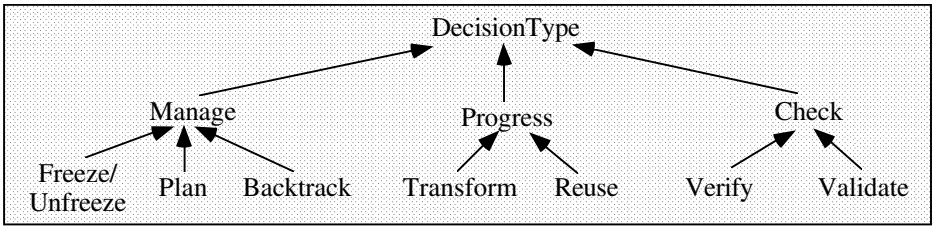

Fig. 3. Parts of a lattice of micro-level decision classes in NATPROC.

3 Support for Situation and Decision Modeling

Figure 2 simply says that a situation is defined on product information. This is a gross over-simplification. This section applies the three dimensions shown in figure 1 to elaborate contributions to the modeling of situations, actions and decisions made in NATURE.

3.1 The Specification Dimension: Domain Theory

The domain theory aims to describe the knowledge structures people develop, and ultimately remember, when they are investigating problems. Abstractions for various domains have been proposed as templates and mechanisms for analogical transfer of knowledge between domains belonging to the same class [GiHo83; Grei88].

The importance of domain knowledge has been recognised from two directions. First, cognitive studies of software engineering have demonstrated that experts use memory schema of domain knowledge to help construction of specifications [GuCu88; Guin90]. A model of domain abstraction appears to be necessary for retrieving application domain knowledge from memory, then understanding the implications of that knowledge in a new context. Studies of the matching problem [MaSu92; MaSu93] have demonstrated that software engineering domains can be described as abstract models and matched using a meta schema of goal related and structured domain knowledge. This approach synthesises concepts of templates, object-orientation and structure mapping theory [Gent83], which defines analogy as the matching of a structured set of propositions.

Second, there has been a progressive growth in semantic richness of specification languages to model domain concepts. The latter trend, manifest in languages such as TAXIS and Telos [Jark93], has culminated in the recognition of different *descriptive worlds* (figure 4), in which domain knowledge is necessary for effective system development. Making these worlds explicit does not only provide a certain degree of guidance in building a requirements model but also allows the association of non-functional goals with stake–holder groups in requirements teams.

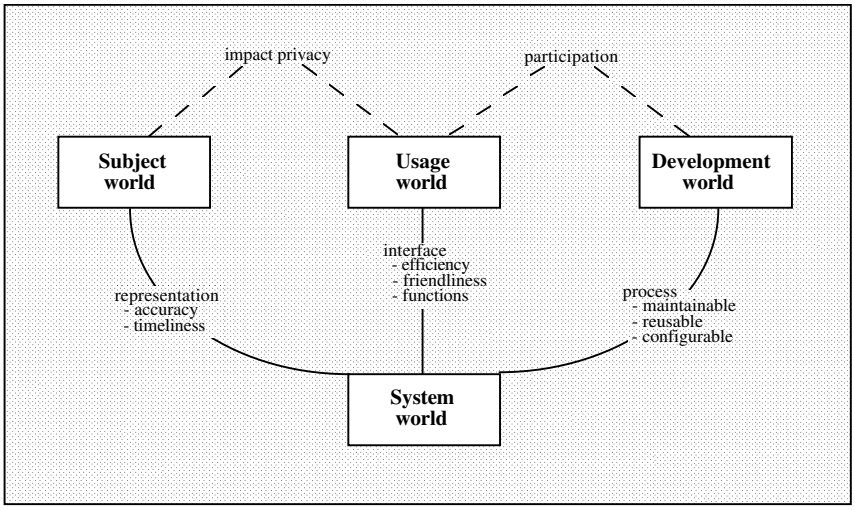

Fig. 4. Categories of domain models: the "worlds" of information systems.

The assumptions of the current theory [Maid92] are drawn from cognitive models of memory, in particular the hierarchical models of natural categories [Rosc91], hierarchical memory schemas [Ande90] and categories of dynamic memory [Scha82]. These assert in slightly different forms that human memory for several different types of knowledge (e.g. object, procedure, plan) is organised in an informal hierarchy of classes.

The domain theory recognises three broad knowledge areas, each related to a particular "world" and defined in a different model investigated in the project:

- domain model of the *subject world*, describing key state transitions and object structures in a software engineering domain. Generic domain models are similiar to the cliches proposed by [ReWa91];
- information systems model, describing computerised functions, design algorithms and their interaction with the domain (the *system world*). Knowledge reuse in the design space is tractable because mapping at the design level of abstraction have been effective in transformation programming [JJWS88]. The theory adds an intermediate level of abstraction linking designs to requirements through the concepts of system purpose;
- purpose model, defining goals of the domain and the required computer system; and organisation model, describing the business, its purpose, context and links to domains within the organisation [BNP*91], both related to the *usage world* of the system.

A fourth model defines the *development world*, statically by elaborating the three dimensions of figure 1, and dynamically by the NATPROC model.

Work so far has investigated the subject domain model and its links to the purpose and organisation models. The domain model consists of a meta-schema of domain knowledge, adapted from existing meta-schemata such as KAOS [DFLa91] and [Luba88], example-based analyses of software engineering analogies [MaSu92], and a set of domain abstractions modeled as classes and specialised by addition of further knowledge to

describe different views on a single domain. Most software engineering domains can be ascribed to one of a tractably small set of domain classes.

The current domain meta schema (figure 5) focuses attention on structural objects and their evolution through state transitions. *Structural objects* are categorized by key objects and possibly other (first-order) properties and composed/decomposed, as described by *object structure* which can themselves have (second-order) properties. *State transitions* are triggered and stopped by *events* but only when certain *stative conditions* hold, some of which are also given directly by a defined sequencing of possible state transitions. The domain model is linked to the purpose and organization model of the usage world through a special kind of object structure called a *goal state*.

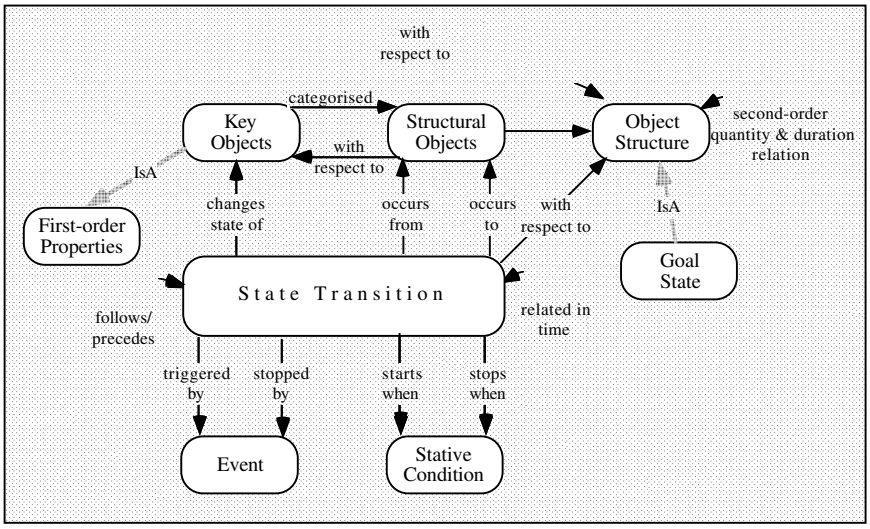

Fig. 5: Meta schema of domain model in which specific domain abstractions are defined.

3.2 The Representation Dimension: Integrating the Formal and Informal

Requirements can be represented using formal specification languages, semi-formal graphical methods or informal aids. Each of these representational forms has certain strengths and weaknesses, so a combination is necessary. This combination should be preserved throughout the software life-cycle rather than being defined away through a once-for-all formalization. Firstly, informal information provides an illustration which may be necessary for understanding formalism. Second, since any formalization implies an abstraction from the informal realities, certain revisions of the formalization may be incomprehensible without the ability of backtracking to informal statements.

To illustrate, consider a situation in a library model. Due to a new requirement (R27), the concept of book should be specialized. Figure 6 shows that there was an earlier decision *„No specialisation of book"*. Retrieving this information needs, in addition to the formal recording of the decision, the integration of informal and formal representations.

Semi-formal and informal languages allow the necessary freedom at the beginning of the acquisition process, but do not offer much support for reasoning about the requirements in a later stage. The use of knowledge representation languages as a backend to *semi-formal techniques* was proposed in [BGMy85]. Besides the advantages of formalization in terms of consistency and completeness checking, mapping support and validation assistance, they emphasize organizational principles such as aggregation (composability of requirements), generalization (avoiding redundancy by inheritance), and classification (meta-modeling). Meanwhile, several formalizations of specific semi-formal methods were proposed. Perhaps the best-known are Greenspan´s formalization of SADT in RML [Gree84] and the formalization of an extended entity-relationship model in ERAE [Hage88]. Thus, the mapping between semi-formal representations and an underlying knowledge base is fairly well understood although few commercial CASE tools cover even the rather weak semantic checks for the standard methods [VJTr92].

The relationship between formal and informal representations is much less understood. In the NATURE prototype, we distinguish between static and dynamic aspects. *Statically*, a formal model can represent an informal object such as an image or a video clip as an uninterpreted node, described by some attributes which give an inkling about its content. Thus, the formal object can be considered as an access path to the informal object; this is a very familiar approach in software databases where the actual software is stored in files and the database contains a *description* of the software.Conversely, the informal object is a visualization, animation or illustration of the formal description.

Dynamically, we have to consider the two-way transition between formal and informal representations. In the creation process, the transition from informal to formal

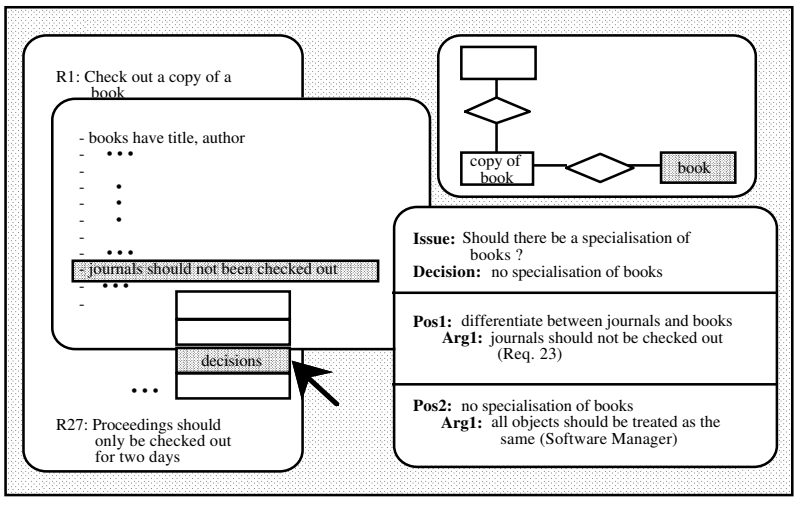

Fig. 6: The use of formal and informal knowledge during decision making.

corresponds to knowledge acquisition, whereas the transition from formal to informal is associated with validation. The informal-to-formal transition corresponds to an abstraction process, the other direction to concretization and authoring. Since we lose information in abstraction, it is important to maintain the dynamic relationship between formal and informal object -- an aspect omitted in the original formalization may become important later on, possibly after a change in perspective or due to the revision of some design decision (cf. figure 6).

The NATURE environment focuses on the representation of these static and dynamic relationships in a concise and usable manner and in furthering the basic understanding of their roles in the requirements process. The integration of hypermedia and knowledge bases (e.g., [EhJa91, JoHa90]) seems to provide a suitable technological basis for this work.

3.3 The Agreement Dimension: Modeling Teams and Process Quality

Requirements engineering is not an activity conducted out of the blue. Before an RE team is established within an organization, some need for changing the existing world is always decided in a process such as rationalisation of perceived opportunities and threats, or political decision-making. The need for change is typically stated in a simple manner as a small set of basic requirements — the *vision* of the system [JaPo93]. A classical example is John F. Kennedy´s *"send a man to the moon before the end of the decade"*.

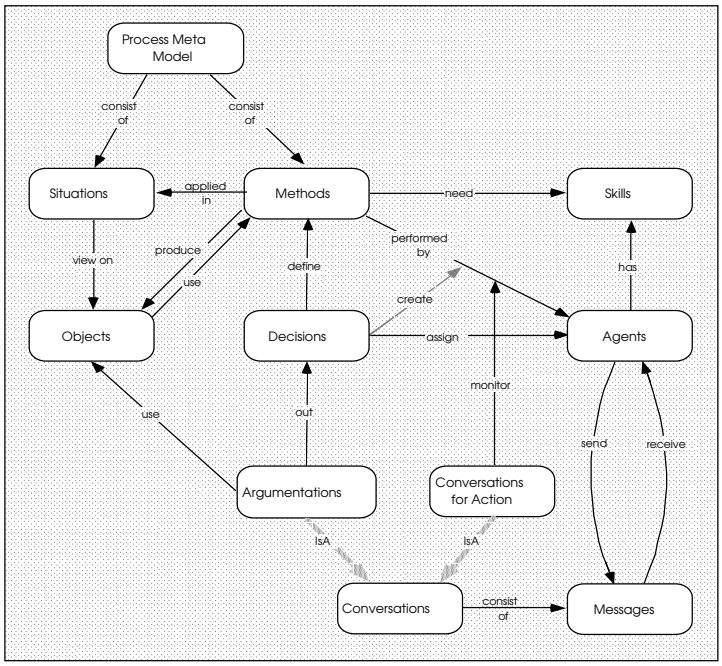

Fig. 7. Adding conversation-based teamwork support to the NATPROC model.

Gaining a requirements specification out of the vision is not an easy task. Within the world in which the vision has to be realized, many habits exist. Some of these habits are

based on formally stated goals, policies, or competing visions. Others are just regularly observable phenomena for which no predefined structure or reasons are known.

Of course, the habits depend on the different contexts (worlds) people are in and the roles people in the requirements team own. Thus, and because of personal factors, they are often conflicting. Such conflicts can only be detected and resolved by communication within an interdiciplinary requirements engineering team.

Our proposition is that non-functional goals are expressed in such communications. Argumentation models such as Rittel´s IBIS or Toulmin´s Argumatics provide the formal framework for introducing non-functional requirements in this manner [CoBe88, HJRo91, RaDh92]. Taking a speech-act perspective [WiFl86], an extended NATPROC model organizes this cooperation in conversation structures. Decisions select among actions to be performed by agents (persons or machines) based on *conversations for argumentation*; figure 6 contains a partial trace following the IBIS model. Decisions do not only depend on the process situation, but also on the availability of agents with the necessary skills. Action performance is controlled through *conversations for action*.

These structures make conversations traceable and thus more understandable. Figure 7 shows how conversational modeling is integrated with the situation-decision-action model of section 2 [PoJa92]. Formally, this integration is achieved by linking conversations to objects and decisions, thus making them part of the working context (situation description along the agreement axis of figure 1).

4 Representation and Reasoning Tools

Technical support for the creation, management, and use of the models discussed in the previous sections must be based on a formal language which offers a rich spectrum of abstractional, constraint checking, and dynamic clustering facilities, and is sufficiently mature in terms of implementation so that substantial experiments can be made. In NATURE, we are using the language *Telos* [MBJK90] and its implementations *ConceptBase* [Jark92] and *SIB* [CJMV92].

4.1 Telos as a Requirements Modeling Language

The knowledge representation and reasoning mechanism has to be expressive enough to capture the essential specializations needed for representing domain knowledge and proccess dynamics adequately and concisely. Our modeling framework relies heavily on the kind of abstraction mechanisms offered by advanced semantic modeling, object-oriented databases, and knowledge representation languages, connected to semi-formal and informal information through the notions of view and structured hypertext, respectively.

Meta modeling is the most crucial requirement. Telos allows the association of predi–cative deduction rules and integrity constraints with meta classes, thus supporting semantic definition and consistency checking of multiple interacting metamodels. Thus, it cannot manage only a requirements model as an evolving knowledge base but also the meta models driving the process. Metamodels can be specialized to particular domains and methodologies. This provides the basis for integrating domain and process

engineering theories into a knowledge representation framework for requirements engineering. Figure 8 shows the basic organization of the requirements information base and its integration of data and process information, essentially a semantic enrichment of the IRDS standard.

The full version of Telos implemented in ConceptBase appears to cover a large portion of the features to be expected from an extensible language for requirements modeling:

- full support for the abstraction principles of aggregation, generalization, and classification;
- no distinction between attributes and relationships -- attributes are first-class objects;
- full extensibility through the combination of meta–classing with deductive rules and integrity constraints;
- integration of temporal information about the evolution of the modeled worlds as well as about the history of the database content;
- data model that lends itself to hypertext-like switching between graphical and frame-oriented representations, fully compatible with semi-formal graphical techniques;
- client-server KBMS with support for teamwork in requirements modeling and process management which is itself based on appropriate meta models.

Considering the intended use of requirements models in capture, reuse, and re-engineer–ing, several extensions to the present Telos version are needed, without giving up the advantages of formal semantics and automated reasoning support. These concern freedoms regarding the completeness and consistency of requirements models [Balz91, FeFi91], which are productive as a basis for finding out what are the real goals, what is important, etc., but also to support oversimplification of models to make them more understandable.

Telos has a formal semantics grounded in deductive database theory [Jeus92]. It offers

IRDS	Product-Model		Process-Model
	Structure	Behavior	
Schema Description Layer	Definition of Product Type	Definition of BehaviorType	Definition of ProcessType
Schema Layer	Definition of Product-Models e.g., Entity-Type	Definition of Behavior-Models e.g., Activity-Type	Definition of Process Models, e.g., refine "Entity"
Data Layer	Definition of Products, e.g., Entity "Customer"	Definition of Behavior, e.g., Enter "Customer"	Trace of Process, e.g., refine "Person to Customer"
Production Layer	Application Data, e.g., Customer "Smith"	Application Data, e.g., Enter "Smith"	

Fig. 8: Meta modeling as a prerequisite for extensible requirements handling.

predicative querying and update facilities similar to those in deductive relational databases, extended to the object-oriented framework. For requirements engineering, these fall short in at least two areas: the acquisition of requirements information from existing systems through *reverse engineering*; and the search in requirements databases where exact matching is highly improbable and therefore *similarity-based search* is necessary.

4.2 Reverse Modeling as a Knowledge Acquisition Facility

Reverse modeling (RM) concerns analysing components of existing computerised or manual systems in order to acquire knowledge of the application domain:

- *concept formation* during the initial requirements acquisition. This is particularly valuable in situations where the structure of the domain is large and complex;
- *validation*: A conceptual schema for a new system can be compared to that of an existing system in order to check if it is reasonable and complete.
- *identification* of reusable components of existing systems and integration points when two systems are to be integrated.

The interest in RM has increased recently due to the large number of legacy systems presently in operation. Often, the exact understanding of the data and procedures in the existing systems has been lost after years of evolution, and reverse modelling is seen as a means to regain that understanding.

An example of reverse modelling is to extract a conceptual model out of examples of filled-in forms [TaWa90]. This technique could be used as a means for concept formation. It increases the role of the end user and decreases the role of the system analyst. Cost and time of requirements capture can be reduced considerably.

Another approach used within NATURE is to construct a conceptual model out of a relational DB schema [Kalm91]. First, relational schemas are transformed into a form appropriate for identifying object structures; certain cycles of inclusion dependencies are removed, and certain relation schemes are split. Second, the relational schema is mapped to a conceptual schema; each relation scheme gives rise to an object type, and the inclusion dependencies give rise to either attributes or generalisation constraints, depending on the structure of the keys in each relation scheme.

This transformation is not just a syntactic one but can be viewed as a knowledge acquisition process. It makes semantics hidden in application code and organisational assumptions explicit in an enriched representation. To get the needed information for the translation, we intend to use domain abstractions that provide suggestions for objects and relations which the data base gives rise to. An advantage of the method is that it makes clear what types of interactions with a user are needed to construct a conceptual schema.

4.3 Principles of Similarity-Based Search

In addition to classification-based retrieval of suitable domain abstractions (sec. 3.1), NATURE also investigates direct analogical reuse of requirements model instances through the definition of a quantitative similarity relation between desriptions of requirements.

The approach only assumes a fairly general conceptual modelling framework, namely the Telos language, but not any particular model for expressing requirements [Pern90]. This choice is motivated by our purpose to develop a rather generic model of similarity computation, applicable to any requirements description model. However, it restricts the final method to be applicable only to requirements specifications employing a certain degree of formality (structured descriptions as opposed to natural language specifications).

In the context of analogical retrieval, similarity is either used as a fine evaluator of source cases initially selected by some coarser search method [CoKj87], or to provide effective trial hints to the subsequent stage of mapping knowledge from source specifications to evolving ones. As a basis for prototypicality estimates [Tver77], similarity can be exploited in selecting prototypical instances of abstract specification models. Such prototypical instances are known to improve the ability of humans in comprehending abstractions, a prerequisite for successfully using them. Moreover, similarity estimates provide useful criteria for classification. Finally, similarity provides solutions to special problems in RE. For instance, parts of specifications which have been detected as sources of consistency violations can be replaced by similar parts, serving analogous roles in other specifications but not leading to the same contradictions.

Having as a final aim the formulation of a computational model of similarity, consistent with properties inherent to relevant kinds of human processing [Sjöb72, Tver77], the following principles have been identified:

- The principle of *ontological uniformity* says that similarity will be only computed at the same meta level of classification.
- The principle of only *partially uniform representation* implies that positive matching attempts are more relevant than discrimination attempts.
- The *normalization* principle requires that similarity must be defined as a monotonically decreasing function of a normalized distance measure.
- The principle of the *domain dominant schema* uses the requirements information base schema to identify key features salient for reuse.
- The principle of the *pragmatic utility subsumption* says that domain dominance subsumes other measures of salience.
- The principle of *classification optimality* defines the trade-off between type A and type B classification errors.

In a way consistent with these six principles, we are going to define similarity on the basis of an overall distance metric, aggregating partial measures of object distances. These partial distances correspond to the three abstractions along which requirements can be organized in Telos: classification, generalization and attribution [SpCo93].

Any elaboration of these concepts must take into account the trade-off between the quality of the final estimates and their computational cost for obtaining them. Moreover, it will be necessary to validate and calibrate the resulting method of estimating similarities between objects within the particular context of requirements engineering .

5 A Standard Example Revisited

To illustrate the interplay among the different NATURE models and tools, the NATURE demonstrator shows an extension of the standard requirements engineering and specification example of a *library system* [Wing90]. We sketch a small subset below.

Process model. Assume that a partial set of such requirements has been captured in a structure linking hypertext into the Telos language through static means and a recorded process history. At a certain point in the past, the existence of both books and proceedings led to a *situation* where the *decision* had to be made whether the same or different lending policies were to be adopted for the different subclasses. The team decided at that time to have a uniform policy, with the nonfunctional goal of a simple system (cf. figure 8). As a consequence, it did not formalize the distinction between books and proceedings (*action*).

Informal-formal integration / nonfunctional goals. Later on, it becomes known that the library will also contain journals which may not be lent for more than one day. The team now decides to create a generalization hierarchy of document types containing books and journals. Only the fact that the initial informal requirements in which both books and proceedings were mentioned reminds them to include proceedings as well in this hierarchy (they are nowhere present in the formal model!). In this situation, the process model can invoke a backtracking decision to revise also the original lending policy, for example, treating proceedings like journals rather than like books. The argument for that revision is that the original simplicity goal is no longer satisfiable anyway.

Similarity-based search. Assume the library system is to be developed by a software house with a large base of experience in similar systems. Through similarity-based search, the requirements model of a car rental system is found. Comparing the two models, the team decides to introduce borrower cards for identification and to distinguish different library users with different lending rights.

Reverse modeling. Suppose our library already has a simple and poorly documented relational database which describes the existing documents and their subject areas. In order to be able to reuse it as part of the new lending system, the team reverse-models it and tries to fit it together with the existing specification through view integration.

Domain abstraction. The team originally thought that the existing relational database would be sufficient for stock control. Comparison with a generic stock control domain abstraction -- triggered by informal critiques of the database -- shows, however, significant deficiencies (e.g., no handling of missing and damaged books) and leads to a completion of this part of the requirements model.

6 Summary and Conclusions

From a discussion of new sources and uses of requirements information, and from a three-dimensional framework derived from the major problems faced in the RE process, we arrived at a situation and decision-oriented process model, together with its characterization through domain theories and knowledge representation and management facilities.

The relationships between the different techniques presented here are summarized in figure 9 which shows how they contribute to the process of moving in the three-dimensional framework of figure 1; integration of formal and informal requirements, the

formal model itself, and the modeling of nonfunctional requirements are all covered under the label of knowledge representation theory.

Initial prototype implementations of the models and tools presented in this paper exist or are in advanced stages of implementation. However, the framework has opened up a lot of questions for further research. Some of the more interesting ones include the study of the business and purpose models in domain theory, the design of advanced interactive capture tools using the theories, the development of process drivers and controllers based on the NATPROC model, and in particular the combination of reverse modeling and view integration for system integration or re-engineering.

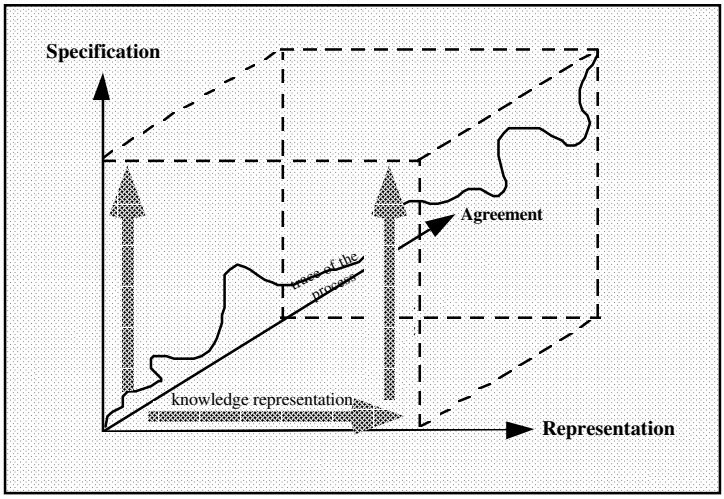

Fig. 9. Contributions of NATURE work to the requirements engineering tasks.

References

[AHTo90] Akman, V.; ten Hagen, P.J.W. and Tomiyama, T.: A Fundamental and Theoretical Framework for an Intelligent CAD System; Computer Aided Design 22, 6, 1990.

[Ande90] Anderson, J.R.: The Adaptive Character of Thought. Hillsdale, NJ: Erlbaum, 1990.

[Balz91] Balzer, R.: Tolerating Inconsistency; Proc. 13th ICSE, Austin 1991, 158-165.

[BiRi87] Biggerstaff, T. and Richter, C.: Reusability framework, assessment and directions; IEEE Software, March 1987.

[BGMy85] Borgida, A.; Greenspan, S.J. and Mylopoulos, J.: Knowledge Representation as the Basis for Requirements Specification; IEEE Computer, April 1985, 82-91.

[Boeh88] Boehm, B.W.: A Spiral Model of Software Development and Enhancement; IEEE Computer 21.

[BNP*91] Brian, P.M.; Niezette, M.; Pantatzis, D., Seltveit, A.H.; Sundin, U.; Theodolidis, B.; Tziallas, M.G. and Wohed, R.: A Rule Language to Capture and Model Business Policy Specifications; Proc. CAiSE 1991, Trondheim, Norway.

[CJMV92] Constantopoulos, P., Jarke, M.; Mylopoulos, J. and Vassiliou, Y.: Software Information Base: A server for reuse; ITHACA-Report, ICS-FORTH, Heraclion 1992.

[CoBe88] Conklin J. and Begeman M.L: A Hypertext Tool for Exploratory Policy Discussion; ACM Trans. Office Information Systems 6, 4, 1988, 140-151

[CoKj87] Cohen, P. and Kjeldsen, R.: Information Retrieval by Constrained Spreading Activation in Semantic Networks; Inf. Processing and Management, 23, 4, 1987.

[DFLa91] Dardenne, A.; Fickas, S. and van Lamsweerde, A.: Goal-Directed Concept Acquisition in Requirements Elicitation; Proc. 6th IEEE IWSSD, Como, Italy, 1991, 14-21.

[Dows87] Dowson, M.: Iteration in the Software Process; Proc. 9th ICSE, Monterey 1987.

[EhJa91] Eherer, S. and Jarke, M.: Knowledge Base Support for Hypermedia Co-Authoring; Proc. Database and Expert Systems Applications, Berlin, 1991, 465-470.

[FeFi91] Feather, M. and Fickas, S.: Coping with Requirements Freedom; Proc. Intl. Workshop Development of Intelligent Information Systems, Canada, 1991, 42-46.

[FKGo90] Finkelstein, A.; Kramer, J. and Goedicke, M.: Viewpoint-Oriented Software Development; Proc. Conf Le Génie Logiciel et ses Applications, 1990, 337-351.

[Gent83] Gentner, D.: Structure Mapping: a Theoretical Framework for Analogy; Cognitive Science 5, 1983, 121-152.

[GiHo83] Gick, M.L. and Holyoak, K.J.: Schema Induction and Analogical Transfer; Cognitive Psychology 15, 1983, 1-38.

[Gree84] Greenspan, S.J.: Requirements modeling: A Knowledge Representation Approach to Software Requirements Definition. Univ. Toronto, Tech.Report. CSRG 155, 1984.

[Grei88] Greiner, R.: Abstraction-based Analogical Inference, in Analogical Reasoning; D.H. Helman (ed.), Kluwer Academic Publishers, 1988, 147-170.

[GrRo90] Grosz, G. and Rolland, C.: Using Artificial Intelligence Techniques to Formalize the Information System Design Process; Proc. DEXA, Vienna 1990, 374-380.

[GuCu88] Guindon, R. and Curtis, B.: Control of Cognitive Processes During Software Design: What Tools are Needed? Proc. ACM-CHI, 1988, 263-269.

[Guin90] Guindon, R.: Designing the Design Process: Exploiting Opportunistic Thoughts; Human-Computer Interaction Journal 5, 1990, 305-344.

[Hage88] Hagelstein, J.: Declarative Approach to Information Systems Requirements; Knowledge Based Systems, 1, 4, 1988, 211-220.

[HeEd90] Henderson-Sellers, B., and Edwards, J.M.: The Object-Oriented Systems Life Cycle; Comm. ACM, Sept. 1990.

[HJRo91] Hahn, U.; Jarke, M. and Rose, T.: Teamwork Support in a Knowledge-Based Information Systems Environment; IEEE Trans. Software Eng., 17,5, 1991.

[JaPo93] Jarke, M. and Pohl, K.: Establishing Visions in Context: Towards a Model of Requirements Processes; Proc. Intl. Conf. Information Systems, Orlando, Fl, 1993.

[Jark92] Jarke, M., ed.: ConceptBase V3.1 User Manual; Aachener Inf.Berichte 92-17, 1992.

[Jark93] Jarke, M., ed.: Database Application Engineering with DAIDA, Springer 1993.

[Jeus92] Jeusfeld, M.: Änderungskontrolle in Deduktiven Objektbanken; DISKI Volume 17, Bad Honnef, Germany: INFIX Publ. (Diss. Univ. Passau, in German)

[JJP*92] Jarke M., et.al.: NATURE: Initial Integration Report; RWTH-Aachen, 1992.

[JJWS88] Johnson, P.; Johnson, H.; Waddington, R. and Shouls, A.: Task-related Knowledge Structures: Analysis, Modelling and Application; Proc. HCI '88, Cambridge University Press, 1988, 35-61.

[JoHa90] Johnson L., Harris D.: The ARIES project; Proc. 5th KBSA, Liverpool, 121-131

[Kalm91] Kalman, K.: Implementation and Critique of an Algorithm which Maps a Relational Database to a Conceptual Model; Proc. CAiSE, Springer, 1991, 393-415.

[Luba88] Lubars, M.D.: A domain modeling representation; MCC Technical Report STP-366-88, Austin, Tx, Nov., 1988.

[Maid92] Maiden, N.: Analogical Specification Reuse during Requirements Analysis; Ph.D. Thesis, City Univ. London, 1992.

[MaSu92] Maiden, N. and Sutcliffe, A.: Exploiting Reusable Specifications through Analogy. Comm. ACM 35, 4, 1992, 55-64.

[MaSu93] Maiden, N. and Sutcliffe, A.: Requirements engineering by Example: An Empirical Study; Proc. Intl. Symp. Requirements Engineering, San Diego, 1993.

[MBJK90] Mylopoulos, J.; Borgida, A.; Jarke, M. and Koubarakis, M.: Telos: Representing Knowledge About Information Systems; ACM Trans. Inform. Systems 8, 4, 1990.

[Pern90] Pernici, B.: Objects with Roles; Proc. ACM COIS, Cambridge, Mass, 1990, 205215.

[Pohl93] Pohl, K.: The Three Dimensions of Requirements Engineering; Proc. of the CAiSE Conference, 8-11 June, Paris, Springer-Verlag, 1993.

[PoJa92] Pohl, K. and Jarke, M.: Quality Information Systems: Repository Support for Evolving Process Models; Aachener Informatik-Projekte 92-36.

[Pott89] Potts, C.: A Generic Model for Representing Design Methods; Proc. 11th Intl. Conf. Software Engineering, Pittsburgh, 1989.

[PTP*88] Punchello, P.P.; Torrigiani, P.; Pietri , F.; Burion, R.; Cardile, B. and Conit, M.: ASPIS: A Knowledge-Based CASE Environment; IEEE Software, March1988.

[RaDa92] Ramesh, B., Dhar, V.:Supporting Systems Development by Capturing Deliberations during Requirements Engineering; IEEE Trans. Software Eng. 18, 6, 498-510.

[ReWa91] Reubenstein, H.B. and Waters, R.C.:The Requirements Apprentice: Automated Assistance for Requirements Acquisition; IEEE TSE, March 1991, p. 226-240.

[RJG*91] Rose, T.; Jarke, M.; Gocek, M.; Maltzahn, C. and Nissen, H.: A Decision-Based Configuration Process Environment; Software Engineering Journal 6, 5, 1991.

[Rosc91] Rosch, E.: Prototype classification and logical classification; In Scholnick (ed.): New Trends in Conceptual Representation: Challenges to Piaget's Theory? LEA 1991

[Royc70] Royce, W.W.; Managing the Development of Large Software Systems; Proc. IEEE WESCON, 1970.

[Scha82] Schank , R.C.: Dynamic Memory: A Theory of Reminding and Learning in People and Computers; Cambridge University Press, 1982.

[Sjöb72] Sjöberg, L.: A Cognitive Theory of Similarity; Goteborg Psych. Rep. 2, 10, 1972.

[SpCo93] Spanoudakis, G. and Constantopoulos P.: Similarity For Analogical Software Reuse: A Conceptual Modelling Approach; Proc. CAiSE, Paris 1993.

[TaWa90] Talldal, B. and Wangler, B.: Extracting a Conceptual Model from Examples of Filled In Forms. In N. Prakash (ed.), Data Management: Current Trends; New Delhi: Tata McGraw-Hill, 1990, 327-358.

[Tver77] Tversky, A.: Features of Similarity; Psychological Review, 44(4), July 1977.

[VJTr92] Vessey, I., Järvenpaa, S.I. and Tractinsky, N.: Evaluation of Vendor Products: CASE Tools as Methodology Companions; Comm. ACM 35, 4, 1992, 90-105.

[WiFl86] Winograd, T. and Flores F.: Understanding Computers and Cognition: A New Foundation for Design; Ablex, Norwood, NJ, 1986.

[Wing90] Wing J.: A specifier´s introduction to formal methods; IEEE Computer 23, 9, 8-26.