Integration of Formal and Informal Techniques for Requirements Engineering*

Roel Wieringa†

November 22, 1995

1 Introduction

Goal of the research reported here is to develop an integrated toolbox for writing and validating a specification of required system behavior. The toolbox is to consist of methods, techniques and heuristics in the first place; eventually, some of this may be implemented in a software tool. The toolbox is intended to help the analyst bridge the gap between informal and formal requirements. The analyst must bridge this gap in two directions: from informal requirements to formal requirements (called *requirements elicitation*) and back again (called *requirements validation*).

The kind of systems for which requirements are sought is, first of all, the class of information systems. Recent developments in practice and in research have however led to the dissolution of boundaries between information systems, systems for electronic data interchange (EDI) among organizations, and control systems (e.g. embedded real-time systems). Another boundary that tends to be dissolved is that between knowledge-based systems and these other groups of systems. Whatever the exact nature of the boundaries between these systems, note that they all are *systems*, and that, therefore, all general principles of system development pronounced here should apply to all of them them.

The reason for wanting a conceptual *toolbox* containing components of different methods is that no method in this area contains merely good ideas, and any method contains at least one good idea. It should therefore be a sensible approach to try to pick the best elements of the different methods. If anything, this should lead to an accumulation of good ideas instead of to a series of revolutions in which each step is radically different from the previous one. In the area of methodology, at least, the period of religious wars and parochial nationalism should be over. But if we are going to use the best elements of different methods and techniques, care must be taken that these fragments produce specifications that form a coherent whole — hence the need for an *integrated* toolbox.

In section 2, we look at the place of requirements engineering in system development and in section 3 we analyze the structure of requirements engineering. The principles uncovered in these sections should be applicable to the development of any kind of system. This allows us to focus in section 4 on the area where the promise of integration is greatest, that of conceptual modeling. In section 5, we finally look at the possibilities to integrate formal and semi-formal (diagram-based) techniques for conceptual modeling. Arguments for the usefulness of this focus for some applications are given later.

[2]Faculty of Mathematics and Computer Science, *Vrije Universiteit*, Amsterdam

*Appeared in *Modelling Languages for Knowledge-Based Systems*, M. Willems, D. Fensel and F. van Harmelen (eds.), Faculty of Mathematics and Computer Science, Vrije Universiteit, Amsterdam and Department of Social Science Informatica, University of Amsterdam, 1995.

2 The structure of development

In this paper, system development is **product** development, where a product is defined as a system constructed by man to perform some useful function. A **system development process** is defined here as a sequence of actually performed activities that leads from a user need to a product that satisfies those needs. As argued at length elsewhere [17], any system development process can be performed in a rational way by combining two kinds of methods:

- **System development methods** to perform the proper activities, so that a useful product is delivered. System development methods do not deal with resource limitations or environmental disturbances.
  - *Requirements determination methods* to analyze user needs and to define a solution space of feasible solutions. Examples are change analysis, critical success factor analysis, value chain analysis, etc.
  - *Conceptual modeling methods* to specify the observable (external) behavior of a feasible solution. Examples are entity-relationship modeling, data flow modeling, external behavior modeling with state machines, etc.
  - *Product implementation methods* to implement a solution. Examples are buying a product, generating it from a specification, building from reusable components, programming.
- **Process implementation methods** to deal with finite resources and a disturbing environment and determine a sequencing of activities that will lead to the desired product.
  - *Process management methods* to deal with finite resources and a disturbing environment. Examples of universal management methods are methods for planning, organizing, staffing, directing and controlling a project. User participation, much emphasized in methods for information system development, is an example of one of the many principles that can be followed in organizing, staffing and directing a development process.
  - *Process strategies* to determine the optimal sequence of activities. Examples are the linear (waterfall) strategy, incremental strategies, and the experimental strategy.

3 The structure of requirements engineering

Requirements determination and conceptual modeling are jointly often referred to as *requirements engineering* in the literature. Here, I define this concept slightly differently. The essence of **engineering** is that system is developed rationally, i.e. that before we perform an intervention in the course of nature, we deliberate upon the optimal intervention. More in detail, rational action proceeds according to the following general pattern of rational problem-solving: analyze the current situation, generate alternative solutions, estimate their effects, evaluate these effects, and choose a solution. In the **engineering cycle** is this cycle is applied to the problem of finding an optimal product design. Any engineering process cycles over the following tasks: *Analyze* the needs, *synthesize* a product design, *simulate* the design to uncover its properties, *evaluate* the design by its potential to meet the user needs, and *choose* a design by iterating over this cycle until a satisfactory design is found. The key feature here is to simulate before to implement. In the context of software product development, useful simulation techniques include throw-away prototyping and formal specification techniques to predict the properties of the implemented product. In many cases, however, just using our gut-feeling

in understanding a design suffices to discover enough properties of the design to be able to do an evaluation.

The engineering approach assumes that there are general rules and guidelines by which to predict the properties of the product from its specification before it is implemented. If there is insufficient knowledge of the domain, or if the domain itself is highly irregular, there is no other way to discover the properties of a design than to implement it and watch it behave in its actual user environment. This is the **experimental** strategy, also called the evolutionary development strategy, mentioned briefly above (as a process strategy). It is to be expected that in ill-understood AI applications, this strategy is to be preferred above the engineering approach.

Requirements engineering is defined in this paper as requirements determination performed according to the engineering cycle. In other words, requirements engineering is defined here as the application of rational problem-solving to the analysis of user needs and to the definition of a solution space of feasible products. We assume that the output of this task is a set of required product properties called the **requirements specification** of the product. Given a list of desired product properties, the task of **conceptual modeling** is then to find an explicit description of external, i.e. observable system behavior. A conceptual model may be specified informally (in unrestricted natural language), semi-formally (e.g. in restricted natural language or with the help of diagrams) or formally (e.g. in logical or algebraic formalisms). The distinction between needs analysis, solution space definition and conceptual modeling is also found in Davis [3], who uses a slightly different terminology.

The reason to separate requirements engineering from conceptual modeling is that requirements engineering focuses on *utility* and conceptual modeling focuses on *validity*. The key question during requirements engineering is "are we specifying an answer to user needs?" and the key question during conceptual modeling is "are we specifying the intended answer correctly?" Thus, requirements engineering is a normative activity, oriented towards finding out what is (instrumentally) good, and conceptual modeling is a descriptive activity, oriented towards finding a true description of required behavior. As a consequence, the general problem-solving cycle mentioned above specializes in the case of conceptual modeling to a variation of the well-known empirical cycle: *Analyze* the required properties, *induce* a conceptual model of required behavior, *test* this model and *evaluate* the results of the test to see if the model correctly describes required system behavior. All conceptual modeling methods agree on this general modeling method. This is at it should be because it is the *only* method known to lead to valid models. However, different conceptual modeling methods differ in the *structures* that they can model and in the *heuristics* used to find (i.e. induce) and validate models.

4 Integration of conceptual modeling methods

Investigation of a number of semi-formal conceptual modeling methods [1, 2, 4, 5, 12, 20, 13] reveals that they contain heuristics to find and validate one or more of the following kinds of structures.

- The **function decomposition** of the product represents required product behavior by decomposing the overall product function into subfunctions until the level of atomic transactions is reached. Note that the function of a product is defined here as its *use*, i.e. the *service* the product delivers to its environment. This is the reason why someone would pay money to acquire the product, which is in turn the reason why the product exists in the first place.
- Product **behavior** is represented semi-formally by various means such as finite state diagrams, state charts and its variants, process graphs, etc. Behavior is always specified as consisting of transactions which are atomic, plus some constraints on the possible occurrence of transactions.

- The **data structure** of the system is represented semi-formally by means of data dictionaries or entity-relationship diagrams and its many variants.
- The **system architecture** is the way the system is built from components. This is irrelevant from a conceptual modeling point of view as long as the system is simple, because a conceptual model only represents the external system view, but for systems with several active components, such as embedded systems, EDI systems or distribute database systems, system architecture is an important part of a conceptual model.

With minor variations in terminology, a similar taxonomy of kinds of modeling structures is found in Woods & Woods [19] and Olle et al. [10]. The functionality of most CASE tools is also based upon the division between the functional/behavior view and a data structure view. A detailed analysis of the differences and agreements and of the potential of integration of a number of frequently used modeling methods is given elsewhere [16].

It is the data structure view that distinguishes conceptual models of data-manipulation systems from conceptual models of other kinds of products. The reason is that data-manipulation systems manipulate *data* and that data items are *symbols* with a meaning outside the modeled product. For example, the modeled product may manipulate data about customers, orders, devices to be controlled, contractual obligations, permissions, authorizations, bank accounts, machine failures, legal rights, data sampled from sensors — in short, about what is called the **universe of discourse** (UoD) of the system in conceptual modeling. The peculiar feature of conceptual modeling of data-manipulation systems is therefore that a well-structured conceptual model of external system behavior contains, as conceptually separate substructures, a model of the UoD of the system and a model of the required system functions. As argued by Jackson [6], this separation of concerns enhances modularity and maintainability of the system. There is no reason why this argument should not apply to knowledge-based systems. The partitioning of the conceptual model into a UoD model and a system function model would be a useful one for knowledge-based systems as well.

The separation of a conceptual model of the UoD from a conceptual model of required system functions has as a consequence that finding a conceptual model of the UoD can proceed according to an empirical cycle from observations of a UoD that already exists. The NIAM method, for example, makes UoD modeling a central theme [9]. Unless an existing system must be reimplemented, conceptual modeling of required system functions usually cannot start from observation of an existing system. If new functionality must be implemented, methods like function decomposition and scenario analysis must be used to derive the conceptual model.

5 Integration of semi-formal and formal conceptual modeling techniques

In some cases, it is useful to combine formal specification techniques with the semi-formal techniques for behavior modeling and data modeling mentioned above. There are no formal techniques to specify system function decomposition or system architecture. (Of course, formal specifications may have themselves a high-level modular structure, but this need not correspond to system architecture.) Focusing on behavior and data structure alone, there is a bewildering variety of logic- or algebra-based specification techniques, where each specification technique may have a multitude of formal semantics and derivation rules. A useful recent survey, without however attempting to integrate the surveyed techniques, is given by Ostroff [11]. A survey of possibilities of *integrating* a significant subset of available formal techniques is given by [7]. Integration of formal and *semi-formal* techniques has

	Semi-formal	Formal
Behavior	Process graphs to represent object life cycles, transaction decomposition tables to represent communication	Recursive specifications in process algebra
	Data dictionary to specify effect of events on objects	Dynamic logic axioms
Data structure	Class diagrams	Ordered sorts, unary functions and predicates in order-sorted logic

Figure 1: Integration of semi-formal and formal techniques in MCM.

been less well-studied.

In MCM (Method for Conceptual Modeling) this has been done by combining existing formal and semi-formal specification techniques as shown in figure 1 [14]. Unifying idea behind this integration is that a data-manipulation system can be modeled as manipulating a set of **surrogates**, each of which represents a UoD object. Objects (an therefore surrogates) have a local state and behavior, and are subject to local and global constraints on permissible states or behavior. These constraints may be soft (they may be violated but this is undesirable) or hard (they cannot be violated). Knowledge about the UoD is represented in these constraints. Assuming the separation of a UoD model from a model of system functions, the knowledge of a KBS would reside in the constraints of the UoD model; functionality would consist if reasoning with these constraints.

Given this simple picture, each system transaction is an update message about a set of one or more UoD objects sent to their surrogates in the system, or a control message sent by the surrogates to the UoD objects. Looking at the system only, each system transaction is a local event in the life of a surrogate or a synchronous communication between a number of surrogates. System functions may be queries about the current state, queries about the history of the current state or queries about possible future scenarios, and are evaluated with respect to this structure. The functions of knowledge-based systems will also fall under one of these categories. This simple picture allows us to represent much of what is common to a wide variety of semi-formal modeling techniques. Details are given elsewhere [14, 15, 16, 17, 18].

The question arises when it is cost-effective to apply the machinery of some formal specification technique. From an engineering point of view, the role of formal specification with respect to requirements engineering is to facilitate *simulation* of the behavior of a product before the product is implemented. Software tools that can execute a formal specification or that can derive properties of the specified behavior can be used to determine whether the specified product would indeed answer the identified user needs. From a conceptual modeling point of view, the same tools can be used to validate the specification, i.e. to verify that the required system will exhibit the desired behavior and nothing else. This effort will be cost-effective if it is less expensive than throw-away prototyping or if it uncovers more relevant system properties using less resources than could be uncovered otherwise (e.g. by throw-away prototyping or by experimental development). The time and effort needed to write and execute a formal specification should be justified by the usefulness of the properties uncovered this way. This is the case in areas where safety is paramount and requirements are stable, such as air traffic control [8]; it is not the case for systems with fastly changing requirements, such as executive information systems. The same criterion also applies to knowledge-based systems: only if requirements

are stable and product safety is paramount does it pay to formally specify the product requirements.

References

[1] G. Booch. *Object-Oriented Design with Applications, Second edition.* Benjamin/Cummings, 1994.

[2] P.P.-S. Chen. The entity-relationship model – Toward a unified view of data. *ACM Transactions on Database Systems*, 1:9–36, 1976.

[3] A.M. Davis. *Software Requirements: Objects, Functions, States.* Prentice-Hall, 1993.

[4] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. Trakhtenbrot. STATEMATE: a working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16:403–414, April 1990.

[5] D. Hatley and I. Pirbhai. *Strategies for Real-Time System Specification.* Dorset House, 1987.

[6] M. Jackson. *System Development.* Prentice-Hall, 1983.

[7] K. Kronlöf, editor. *Method Integration: Concepts and Case Studies.* Wiley, 1993.

[8] N. Leveson, M.P.E. Heimdahl, H. Hildreth, and J.D. Reese. Requirements specification for process-control systems. *IEEE Transactions on Software Engineering*, 20(9):684–707, September 1994.

[9] G.M. Nijssen and T.A. Halpin. *Conceptual Schema and Relational Database Design.* Prentice-Hall, 1989.

[10] T.W. Olle, J. Hagelstein, I.G. Macdonald, C. Rolland, H.G. Sol, F.J.M. van Assche, and A.A. Verrijn-Stuart. *Information Systems Methodologies: A Framework for Understanding.* Addison-Wesley, 1988.

[11] J.S. Ostroff. Formal methods for the specification and design of real-time safety critical systems. *Journal of Systems and Software*, 18:33–60, 1992.

[12] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-oriented modeling and design.* Prentice-Hall, 1991.

[13] P.T. Ward and S.J. Mellor. *Structured Development for Real-Time Systems.* Prentice-Hall/Yourdon Press, 1985. Three volumes.

[14] R.J. Wieringa. A method for building and evaluating formal specifications of object-oriented conceptual models of database systems (`MCM`). Technical Report IR-340, Faculty of Mathematics and Computer Science, Vrije Universiteit, December 1993.

[15] R.J. Wieringa. *Advanced Conceptual Modeling: Semantic, Real-Time, and Object-Oriented methods.* Wiley, To be published.

[16] R.J. Wieringa. Combining static and dynamic modeling methods: a comparison of four methods. *The Computer Journal*, To be published.

[17] R.J. Wieringa. *Structured Requirements Determination and System Modeling: A Framework for Understanding.* Wiley, To be published.

[18] R.J. Wieringa, R. Jungclaus, P. Hartel, G. Saake, and T. Hartmann. OMTROLL — Object Modeling in Troll. Proceedings of the International Worskshop on Information Systems — Correctness and Reusability (IS-CORE'93), Udo W. Lipeck and G. Koschorrek (eds), pages 267–283. Institut für Informatik, Universität Hannover, Postfach 6009, D-30060, Hannover., September 1993.

[19] D.P. Wood and W.G. Wood. Comparative evaluations of specification methods for real-time systems. Technical report, Software Engineering Institute, September 1989. Draft version.

[20] E. Yourdon. *Modern Structured Analysis.* Prentice-Hall, 1989.