Utilizing Inheritance in Requirements Engineering

Hermann Kaindl
Siemens AG Österreich, PSE
Geusaugasse 17, A 1030 Vienna, Austria
Tel: +43-1-71600-288 Fax: +43-1-71600-323
E-Mail: kaih@siemens.co.at

KEY WORDS AND PHRASES

Hypertext, inheritance, requirements engineering, semiformal representation.

INTRODUCTION

Specifying the requirements of a new system to be built is one of the most important parts of the life cycle of any project. In the field called *requirements engineering* many approaches have been proposed [1]. However, few methods and tools have been available for practical use. In fact, for the early phase of defining the requirements, nearly no support is available.

While from a theoretical point of view it would be desirable to have *formal* representations of requirements, in practice unstructured natural language is often used *informally*. Our approach attempts to bridge the gap between these extremes in providing *semiformal hypertext* representations. Therefore, our approach and the tool supporting it are named RETH (Requirements Engineering Through Hypertext). Actually, RETH uses a combination of various technologies, including *object-oriented approaches* and *artificial intelligence* (in particular *frames*). We do not attempt to exclude or replace formal representations, but try to complement and to provide means for gradually developing them.

The scope of this paper is the utilization of *inheritance* for *requirements specification*, i.e., the tasks of analyzing and modeling the domain, as well as forming and defining requirements.

Among others, RETH has been applied in the CERN (Conseil Européen pour la Rechereche Nucléaire) *Cortex* project. While it would be impossible to explain this project in detail here, it should be sufficient to know that it deals with a generic distributed control system. Since this project is not finished yet, it is difficult to state its size precisely. In order to give an idea, its final goal is to substitute the many existing similar control systems at CERN by this generic approach. Currently, RETH is also tested using real-world requirements for the *Pastel Mission Planning System* at ESOC in Darmstadt.

First, we outline how hypertext is integrated into a frame system in our approach. Moreover, we demonstrate the usefulness of inheritance as performed by the tool RETH. We then summarize our experiences of utilizing inheritance in the Cortex project. Lastly, we relate RETH to existing work.

HYPERTEXT INTEGRATED INTO A FRAME SYSTEM

A hypertext node is represented as a frame in our approach. (The original notion of a *frame* was coined by Minsky [2], but the frame systems implemented the original ideas only partially. In the context of this paper, a frame can be viewed as a data structure that combines data stored in *slots*.) According to the differences between object-oriented languages and frame systems as discussed in [3, 4], we selected the frame system of PROKAPPA as the basis of our tool RETH.

Our approach of integrating hypertext into a frame system is similar to the one described and used by Kaindl and Snaprud [5, 6] for *knowledge acquisition* in the course of building knowledge-based (expert) systems. One distinctive feature lets the user define disjoint *partitions* of nodes that together cover the whole node. Such a partition of a hypertext node is comparable to a slot of a frame. The idea is to support the user in partitioning the textual content in a machine recognizable form, serving as an additional means of introducing more formality.

In order to make the example below understandable, we shortly sketch the hypertext user

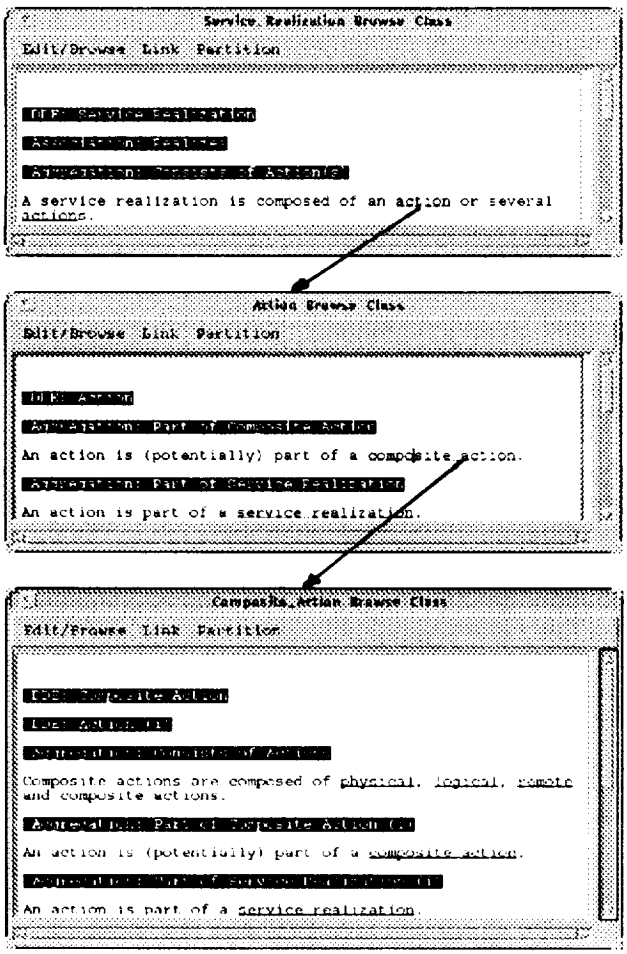

Figure 1: RETH windows showing object representation and inheritance.

interface of RETH (see Fig. 1 showing a screen dump). The presentation level handles hypertext links as follows: if the underlined string representing the link is clicked with the mouse, the window of the target node is displayed by the tool. The arrows in the figure are drawn to indicate the effect of following links on the screen. The windows shown in the figure actually poped up one at a time.

In contrast, the display of *partitions* of hypertext nodes is implemented in our tool like *expand buttons* (cf. the hypertext system *Guide* [7]). When the name of a partition (inverted in the display) is clicked, the content is expanded or shrunk (implemented as a toggle). E.g., in the window at the top of Fig. 1 the partition DDE: Service Realization is currently shrunk, while Aggregation: Consists of Action(s) is expanded. In contrast to many hypertext systems, our approach lets users mix browsing and editing of nodes, though one node can either be edited or browsed at one point in time.

INHERITANCE IN REQUIREMENTS SPECIFICATION

Due to lack of space, we cannot describe here the details of using RETH for domain analysis and modeling, and for the formation and definition of requirements. The key ideas are to represent requirements as objects, and to organize these objects as well as the objects of the domain model in a taxonomy. Within this taxonomy, inheritance can be used in several ways (see below). Hypertext links are used to interlink the hypertext nodes representing the objects. For a detailed description, the interested reader is referred to [8].

Due to our tight integration of hypertext in a frame system, inheritance can be used already in the semiformal representation. There is a notable difference between frame systems and object-oriented languages relevant for our approach: in contrast to the latter, the former also support inheritance of *values* [3, 4]. Since classes (of the domain model as well as of requirements) are described in hypertext nodes, and since these are represented as frames, the text contained in them is inherited.

Together with the concept of partitions of nodes, inheritance supports *templates*, e.g., for requirements to be filled in. Whenever a node for a requirement is created as an instance of a class of requirements, the appropriate *structure* is already given initially through inheriting a template. Inherited partitions in the (requirements) instances provide for the representation of information on requirements such as their source, reason and priority.

Detailed information about requirements is especially important for large projects, but without sufficient tool support it is often omitted. Since all the instances inherit all the respective partitions, providing such information cannot be forgotten, and the user of the system just has to fill in the text.

When requirements are organized in *classes*, all the requirements of a specific class can have a special attribute in common — represented as a partition. Moreover, whole classes of requirements (defined by the user) can have the same value (text) of an attribute, and this value can be defined *once* in the description of the class. The subclasses and instances inherit this value, but inherited information can also be overridden.

An important point is that inheritance allows one to define special attributes (including a value

or not) *once* in the definition of the class, without the necessity to copy. Even more important is the possibility of re-inheriting changed values.

In contrast to most current OOA tools, RETH implements OOA inheritance already in the semi-formal hypertext representation (see also Fig. 1).

EXPERIENCE WITH RETH IN THE CORTEX PROJECT

According to our experience in the real-world project Cortex, all the features of our method and its supporting tool were useful to some extent. In fact, some of them were worked out in detail in the course of this application. Due to lack of space, we will only focus here on the utilization of inheritance.

The templates of requirements depending on their class helped to point out missing information. Actually, much of it was known by the people involved, but we found it important to get it written down.

Moreover, we would like to point out specifically the usefulness of domain-specific requirements *classes*, and the use of *inheritance* within the corresponding taxonomy. They allowed the explicit ordering of the requirements according to the classification principle. While this is of course not a new principle for ordering requirements, our approach and the tool provide inheritance. Therefore, it was possible and very useful to specify information such as priorities *once* for whole classes. When the priority of a class of requirements changes, it is only necessary to specify this once — in the corresponding partition of the node representing this class. The nodes representing requirements subclasses and instances of this class re-inherit this changed value.

Another interesting example of the use of inheritance that we came across during the work on Cortex is illustrated in Figs. 1 and 2 (in the notation of [9]). An Action is *part of* a Service_Realization. Since a Composite_Action, e.g., *is an* Action, it is also *part of* a Service_Realization. This inference has to be drawn by the viewer of the O-O diagram but is made automatically via inheritance in RETH. In the bottom window of Fig. 1, the inherited partition Aggregation: Part of Service Realization (i) shows this. Moreover, inheritance points to the fact that a Composite_Action is (potentially) also *part of* a Composite_Action (see the inherited partition Aggregation: Part of Composite Action (i) in the bottom window of Fig. 1). Especially this kind of inference may be difficult for people not so familiar with recursive structures in O-O diagrams. Of course, the diagram has its advantages, too. Therefore, both forms of representation are complementary in our view.

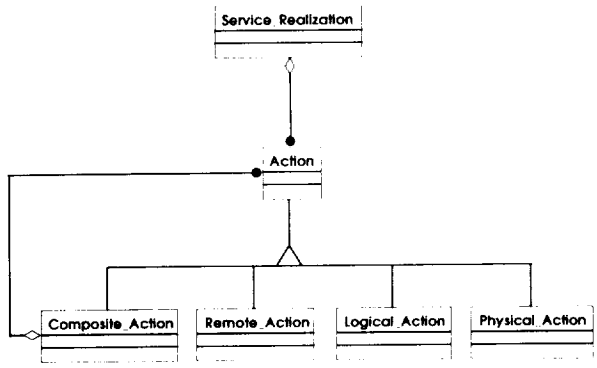

Figure 2: An object model diagram.

RELATED WORK

Due to lack of space we cannot give here a comprehensive overview of all the proposed approaches to requirements engineering. Especially for the traditional ones, the interested reader is referred to [1]. Recent OOA approaches (for an overview see [10]) challenge the traditional ones. RETH heavily builds on object-oriented ideas. However, most of today's OOA methods still ignore early development phases where important clarifications have to be made. It may even be argued that they are designed for a different phase. RETH specifically focuses on the early phase, and we propose to combine RETH with object-oriented analysis approaches.

The method by Jacobson *et al.* [11] and the tool supporting it (*Objectory*) bear some similarity to our approach. However, it does not apply object-oriented principles to the organization of the requirements, and consequently inheritance cannot be utilized (e.g., for templates).

Since RETH's internal representation is based on frames, it may be interesting to compare it with other approaches to requirements engineering using *artificial intelligence* (AI) technology. GIST [12] is an important early approach. RML [13] emphasizes the use of *knowledge representation* techniques of AI and domain modeling. Telos [14] is a derivative of RML. RA [15] shares with RETH the focus on a transition between informal and formal representations. The approach of ARIES [16] is quite similar to RA in being very knowledge-intensive. KBRA [17] utilizes hypertext ideas internally. While RETH's user interface for structuring text appears to be more

developed, some of KBRA's features of machine support could be very useful in RETH. However, KBRA lacks several important features of RETH.

CONCLUSION

In summary, our tool-supported method named RETH supports several activities in the course of requirements specification. Our approach of organizing the hypertext according to *object-oriented* principles has several advantages. Representing requirements as objects helps when structuring them via *classification*. *Inheritance* is provided by our tool already in the early phase of requirements specification, which helps to avoid redundant representation of information. In particular, it provides users automatically with templates of the internal structure of requirements, that depends on the kind of requirement. This way, the users are guided to fill in important information like the reason and priority of each requirement. While RETH is not intended to substitute useful existing techniques emphasizing more formal representations, it can be combined with them.

Since the advantages of such an approach to requirements engineering cannot be fully utilized without more elaborate *traceability* of the requirements, we also investigate how to best link requirements objects with *design* objects.

The usefulness of RETH to space projects is currently assessed using real-world requirements for the *Pastel Mission Planning System* at ESOC in Darmstadt. While it is too early for a final statement at the time of this writing, the preliminary results are encouraging. Since RETH is very general in terms of application areas, we could not find any reason why the application to space projects should be a problem.

ACKNOWLEDGMENTS

Stefan Kramer and Stefan Korner did very important work in building the tool. Peter Tippold gave useful comments on earlier drafts of this paper, and his willingness for in-depth discussion of practical issues in requirements engineering is highly appreciated. Also Holger Ziegeler participated in earlier discussions. Moreover, the partial funding of this work by ESA and the ITF (Innovation- und Technologiefonds) is acknowledged. Finally, we would like to thank CERN for giving us the opportunity to apply our new approach in one of their real-world projects.

References

[1] A. M. Davis. *Software Requirements: Objects, Functions, and States*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[2] M. Minsky. A framework for representing knowledge. In P. Winston, editor, *The Psychology of Computer Vision*, pages 211–277. McGraw-Hill, New York, 1975.

[3] H. Kaindl. Object-oriented approaches in software engineering and artificial intelligence. *Journal of Object-Oriented Programming*, 6(8):38–45, January 1994.

[4] H. Kaindl and S. Korner. Object-oriented languages supplemented with features of frame systems. In *Proceedings of the OOPSLA '92 Workshop on Object-Oriented Languages: The Next Generation*, Vancouver, Canada, October 1992.

[5] M. Snaprud and H. Kaindl. Knowledge acquisition using hypertext. *Expert Systems with Applications: An International Journal*, 5(3/4):369–375, 1992.

[6] M. Snaprud and H. Kaindl. Types and inheritance in hypertext. *International Journal of Human-Computer Studies (IJHCS)*, 1994. To appear.

[7] P. J. Brown. Do we need maps to navigate round hypertext documents? *Electronic Publishing—Origination, Dissemination, and Design*, 2(2):91–100, July 1989.

[8] H. Kaindl. The missing link in requirements engineering. *ACM Software Engineering Notes (SEN)*, 18(2):30–39, 1993.

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, and W. Lorensen. *Object-Oriented Modeling and Design*. Prentice Hall, Englewood Cliffs, NJ, 1991.

[10] D. de Champeaux and P. Faure. A comparative study of object-oriented analysis methods. *Journal of Object-Oriented Programming*, pages 21–33, March/April 1992.

[11] I. Jacobson, M. Christerson, P. Jonsson, and G. Övergaard. *Object-Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley, Reading, MA, 1992.

[12] R. Balzer. Final report on GIST. Technical report, USC/ISI, Marina del Rey, CA, 1981.

[13] A. Borgida, S. Greenspan, and J. Mylopoulos. Knowledge representation as the basis for requirements specification. *IEEE Computer*, 18(4):82–91, 1985.

[14] J. Mylopoulos, A. Borgida, M. Jarke, and M. Koubarakis. Telos: Representing knowledge about information systems. *ACM Transactions on Information Systems*, 8(4):325–362, 1990.

[15] H. B. Reubenstein and R. C. Waters. The requirements apprentice: automated assistance for requirements acquisition. *IEEE Transactions on Software Engineering*, 17(3):226–240, 1991.

[16] W. L. Johnson, M. S. Feather, and D. R. Harris. Integrating domain knowledge, requirements, and specifications. *Journal of Systems Integration*, 1:283–320, 1991.

[17] A. J. Czuchry and D. R. Harris. KBRA: a new paradigm for requirements engineering. *IEEE Expert*, pages 21–35, Winter 1988.

[0] PROKAPPA is a trademark of IntelliCorp, Inc.