GenGraph: A Multi-Grammar and Multi-Perspective Business Modeling Tool – Overview of Conceptualization and Implementation

Peter Fettke, Peter Loos, Kai Pastor

Johannes Gutenberg-Universität Mainz
ISYM – Information Systems & Management
Lehrstuhl Wirtschaftsinformatik und Betriebswirtschaftslehre
D-55099 Mainz, Germany
E-Mail: {fettke|loos|pastor}@isym.bwl.uni-mainz.de

Abstract: Within the information systems field, modeling of business systems results in complex information models which demand appropriate tool support. This paper discusses the motivation and development of a generic editor for information modeling. Our approach enables the user to configure the concepts of the utilized modeling grammars. So, this approach does not implement a specific set of modeling grammars. Instead it is based on a flexible meta-model which uses typed graph descriptions and allows multi-perspective modeling of business systems. Furthermore, we introduce the concepts of modules and links as means to deal with complex business models. Shortcomings and problems with graph-based descriptions of modeling grammars are discussed.

1 Introduction

1.1 Starting Point

Within the information systems field, information modeling is an important instrument to analyze, design, implement, and deploy business information systems [WW02; My98]. Information models enable or at least ease the management of complex business systems [Be95; SH92]. However, there is a large variety of existing modeling grammars, methods, and techniques which specify which concepts are to be applied to the universe of discourse and how to represent these concepts and their instances graphically.

At first, we have to clarify the terminological confusion in the modeling literature. For example, the term "model" is often used for different purposes. To avoid confusion, we use the following definitions: A *grammar* "provides a set of constructs and rules that show how to combine the constructs to model real-world domains" [WW02, p. 364]. In the remainder of this paper, we always refer to analysis grammars, e. g. the entity-relationship modeling grammar (ERM) or the Unified Modeling Language (UML). And while *modeling method* "provides procedures by which a grammar can be used" [WW02, p. 364], *scripts* are the product of the modeling process. "Each script is a statement in the language generated by the grammar" [WW02, p. 364]. A script is a representation of a real-world domain using a particular grammar. We use the term model for a set of scripts representing different aspects of a particular real-world domain. The visual respectively graphical representation of a script is called a diagram.

1.2 Problem

Different grammars are needed to describe different aspects of an information system. For example, known reference business models use several modeling grammars [FL03]. Frequently addressed aspects are data, functions, organization, and processes [Sc98]. These aspects describe different views of the same system. Furthermore, it is useful to model business systems from different perspectives [Fr02; RG02]. For instance, business models can be used for software development or business reengineering. These application areas have different modeling requirements [Be02]. Nevertheless such models from different perspectives should be smoothly integrated. In order to build well-structured, consistent scripts, the grammars which are used to describe various views of one system should be specified by a single integrated meta-model [SR98].

Most modeling tools offered by commercial software providers support a fixed set of modeling grammars. The level of integration of different grammars as well as the integrity of a set of scripts modeling a particular domain varies. The majority of tools lack sophisticated features for customizing existing grammars, for defining new grammars and for smoothly integrating custom grammars with predefined ones. Customization of modeling grammars is important for information systems research when investigating improved or even new modeling methods. Source code for commercial tools is usually not available. However this is a prerequisite for some efforts to provide better modeling tools.

All-purpose schema-drawing tools can be used to create diagrams in any language without being strictly directed by a grammar. Since these tools do not know the concept of a grammar, they provide little support to ensure the construction of syntactically valid scripts. Graphics tools do not offer means to maintain structure, overview and correctness in large complex models. Some tools (e.g. Microsoft Visio) have powerful programming interfaces which may be utilized to implement missing grammar and analysis features. However it is questionable if adding features which we regard as fundamental to an otherwise incomplete tool is the right way to tackle the problems with existing modeling software.

1.3 Approach

To address the problems, we designed and implemented a prototype software tool named GenGraph. This tool can be used for flexible enterprise modeling because it is not limited to a specific, predefined set of modeling grammars. Instead, it is possible to configure GenGraph for a wide range of different grammars by describing their concepts and combination rules. So it is possible to customize GenGraph for grammars such as ERM, EPC, UML, Petri-Nets, or other desirable grammars (e. g., this tool can also be configured with grammars representing problems from business application domains like bill of materials management and recipe management [LS94]). We call this feature multi-grammar support. To specify the concepts of a grammar, we use (typed) graphs as a meta-model. A special tool named GenGraph-Configurator supports the definition and modification of meta-models by specifying properties and restrictions of graph elements in a meta-model. With GenGraph, models can be created according to such definitions. Furthermore, GenGraph implements the concept of *views*, provides *modules* to structure large models, and has *links* to support navigation in complex models. Both meta-models and models are stored in a database (see Figure 1).

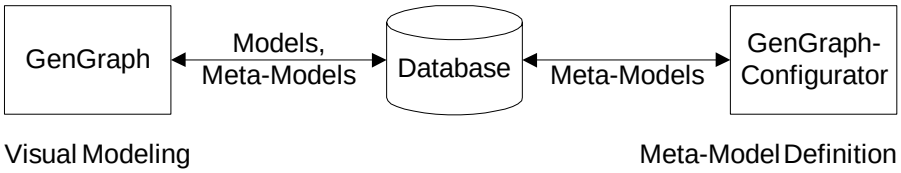

Figure 1: Overview of GenGraph

Once defined, meta-models can be reused for any number of modeling activities. Thus configuration is not a prerequisite for every single modeling project. If a generic modeling-tool is distributed together with meta-models for widely-used languages, it can be used without additional setup costs in comparison to non-generic products. Meta-models can be provided by third parties, too. Generic modeling tools help to provide tool support for new methods and languages without reinventing the wheel.

1.4 Structure

The following section discusses related work. Section 3 introduces the concept of models in GenGraph. Section 4 presents the structure of the meta-models that are used to describe modeling grammars. Section 5 illustrates these explanations by an example. Section 6 presents the design and implementation of GenGraph. Limitations of our approach are discussed in Section 7. The last section summarizes our findings and presents directions for further research.

2 Related Work

2.1 Business Modeling Tools

Business modeling tools support analysis and design of enterprise information systems. One of the best-known dedicated business modeling tool with over 14,000 licenses worldwide [GR00, p. 79] is the ARIS-toolset by German IDS Scheer AG. Originating from academic research, it is a commercial product today. However it does not provide mechanisms to customize and extend modeling grammars for domain-specific requirements [Da00].

A generic model editor based on the so-called E^3-Method has been developed at University of Technology Dresden (Germany) [An03b]. The E^3-Method engineering approach is described in [Gr03] (submitted PhD thesis). Because Greiffenberg's activity is parallel to our work, we were not able to take the results of his approach into consideration. For the moment, both approaches look similar and should be compared in detail in the future.

2.2 CASE Tools and Meta-CASE

Apart from business modeling tools in a narrower sense, related work can be found in CASE tools research. CASE tools are dedicated to the analysis and construction of software systems. Software is the core of the automated part of enterprise information systems.

State-of-the-art commercial CASE tools provide support for modeling using one or more grammars as well as powerful functions for deriving (parts of) an implementation from visual models. Such tools use normally different, integrated, but fixed grammars. However, most tools have some disadvantages for the purpose of designing enterprise information systems. All-purpose CASE tools do not provide sufficient support (in terms of modeling grammars) for modeling all relevant aspects of enterprise information systems. Modeling languages provided by CASE tools are focused on software engineering. For instance, support of the UML for modeling the organizational aspect and the business processes is missing or immature [LF01].

Customization and extension of modeling grammars for domain-specific requirements is difficult or impossible to do with standard CASE tools. The idea of a generic graph editor promised to solve this problem. The realization of generic graph editors can be seen as a major enabler for the development of meta-CASE-technology [Al91]. Meta-CASE tools are to support the construction of CASE tools.

A prominent commercial Meta-CASE toolset is MetaEdit+ [Me03] which originates from academic research by Smolander et al. [Sm91]. The MetaEdit project started with aims similar to ours. But the present result of their work is a complete, fixed and rather expensive commercial product, while we were looking for a tool to experiment with in a less restricted way.

Sapia/Blaschka/Höfling present GraMMi which is a generic graphical modeling tool based on a standard (IRDS) repository management system [Sa00]. GraMMi uses typed graphs as model formalism. Layered graph grammars define model types. There is no special configuration program or language; model types are entered "directly" using the repository administration tools. This work addresses the domain of data warehousing.Ledeczi et al. describe a Generic Modeling Environment (GME) which allows the creation of domain-specific modeling and program synthesis environments [Le01]. GME is a quite advanced tool with various programming interfaces. It uses a meta-modeling approach which is based on the Unified Modeling Language. Static constraints may be specified using the Object Constraint Language (OCL). Meta-modeling is done with an accordingly customized GME. Specific needs of business modeling are not focused in that work.

3 Models in GenGraph

Basically, models in GenGraph are typed graphs, consisting of *vertices* and *edges*. (From here, we will use the term *elements* when we do not have to distinguish between vertices and edges.) Elements may have a number of *properties* (e.g. "Name" as string). **3.1 Views**

A *view* presents elements graphically according to a certain modeling grammar. The use of views is to support multi-perspective modeling. The user determines the arrangement of elements in a view by direct manipulation. He/she may add new elements to a view either by choosing elements which already exist in the model, or by newly creating them. All manipulations to a view are restricted by the content of the meta-model. Thus the risk of creating syntactically invalid models is reduced. An element may be reused in any number of views in the same model. It might have different graphical representations in views of different aspects.

3.2 Modules

The construct of *modules* is introduced in order to structure complex models. Modules are similar to folders in file systems, but they have additional properties. They may contain vertices, edges and views. In GenGraph, a module is a special vertex. This construction has two advantages. First, it allows to build a hierarchy of modules. Second, if the modeling grammar has an explicit module concept (e.g. packages in the UML), the user can create special views displaying the relationships of modules apart from the obvious hierarchical "containment" relationship. Pushing abstraction one step further, a model can now be regarded as a special module which does not have a parent module. In contrast to ordinary modules, models need special treatment with regard to administration of existing models in a database and association with a grammar.

Modules are to support understandability and maintainability of scripts. Therefore we favor loose coupling of modules. Coupling of modules arises from accessing (i.e. reusing) elements of a module M_A in a (view of) module M_B. In GenGraph, it is not allowed to reuse elements of a module M_B in views of module M_A, except if you state explicitly that module M_A shall include module M_B. In GenGraph views, vertices belonging to another module are annotated with an 'i' (for inclusion).

Figure 2 shows a part of the GenGraph class diagram illustrating major relationships of modules, models, views and elements.

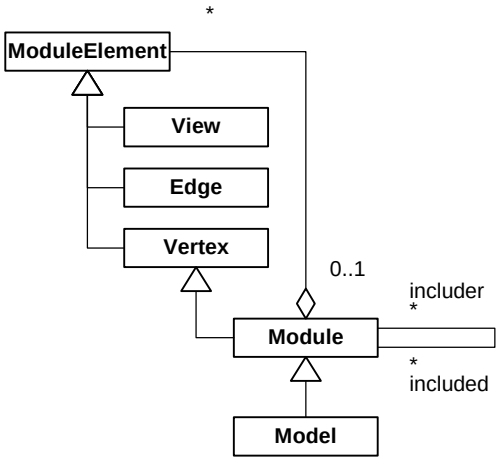

Figure 2: Class diagram of the module concept

3.3 Links

Another concept in GenGraph is the *link*. Links may be used to connect vertices (as anchor) with views (as target). Links are useful for different purposes. For example, a link can connect an activity vertex to a view which refines that activity. Like in hypertext documents, links in GenGraph are a powerful mean to navigate in a complex model. In GenGraph views, vertices that do have at least one link are annotated with a small arrow.

4 Describing Meta-Models

GenGraph enables the user to customize the modeling grammar to be used. This is done by specifying their meta-model. The meta-model defines the types of vertices, edges, properties, views, modules and links (i.e. the concepts), the elements' valid usage and graphical representation within views (i.e. the representation) and the vertex types which may legally be connected by an edge of a certain type (i.e. restrictions). Generally, there are type objects for every object which may occur in a model.

GenGraph uses a dedicated tool named Configurator for entering meta-model descriptions. (Using GenGraph with a special meta-modeling customization was considered as well, but some conceptual and practical problems need further analysis.) The tool has to support creating new definitions from scratch, modifying existing meta-models and defining new model types based on existing definitions of modeling grammars. Each configuration task may need a different order of activities. Therefore the GenGraph Configurator does not prescribe a particular order of steps although dependencies do exist. Figure 3 depicts the user interface of the Configurator.

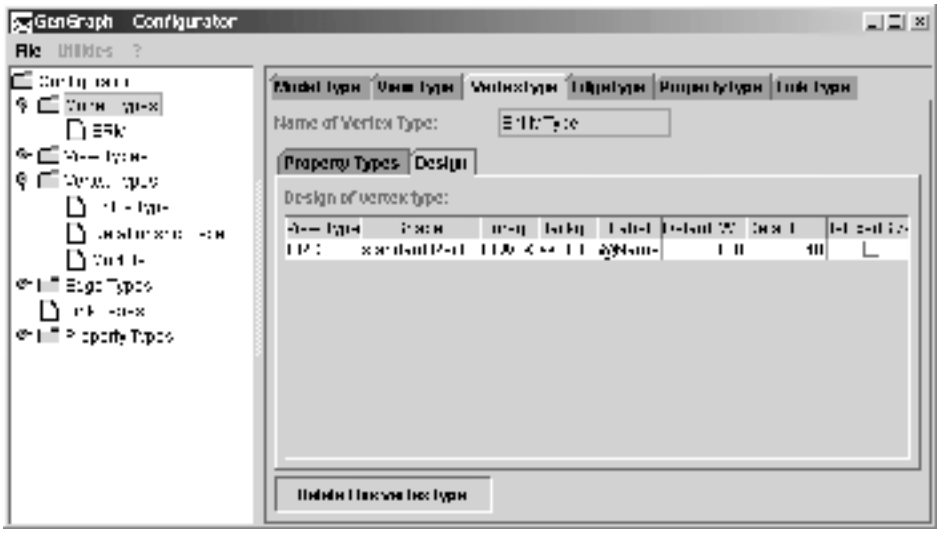

Figure 3: Sample Configurator session. This mask is used for defining the graphical representation of a vertex type in different view types.

5 Example

The entity-relationship diagram [Ch76] has three types of vertices, “Entity Type”, “Relationship Type” and “Attribute”, and two types of edges, “[Entity Type] takes part in [Relation Type]” and “[Entity Type] has [Attribute]” (constructs of the grammar are embraced with square brackets). Each vertex type has a property “Name” which is a string. “Entity Type” is represented by a rectangle (see the Configurator example in Figure 3). “Relationship Type” is drawn as a diamond. “Attribute” is displayed as an ellipse. All are labeled with their name. The first type of edge is a connection only between an “Entity Type” and a “Relationship Type”. It is drawn as a straight line, labeled with its properties “Role” and “Multiplicity”. The second type of edge is drawn as a straight line, too, but has no attributes.

To be able to create a GenGraph model using the ER approach, we must define a model type (e.g. “ERM”) based on a module type (e.g. “Module”) with an appropriate view type (e.g. “ERD”). The view type is configured to display the elements as described above.

After the meta-model has been specified, it can be applied for modeling. The user may create a model of type “ERM”, add a view of type “ERD” and start adding elements to the model. Figure 4 shows GenGraph when editing an ERD “Campaigns” in module “CRM” of model “Requirements”. Entity type “Customer” has two annotations. The letter ‘i’ indicates that this element is taken from another module which is included by “CRM” explicitly (“Customer” is defined in the top-level module “Requirements”). The small arrow indicates that this vertex is start of a link. The target of the link is a view which refines entity type “customer” by showing its attributes.

The user interface of the modeling tool (Figure 4) shows a standard menu bar and status line. The biggest part of the window is used by the diagram drawing component (“drawing area”). On the left side, there is a navigation tree with folders for modules, views, vertices and edges, and an editor for element properties. Concrete items on the navigation tree may be selected, edited or deleted.

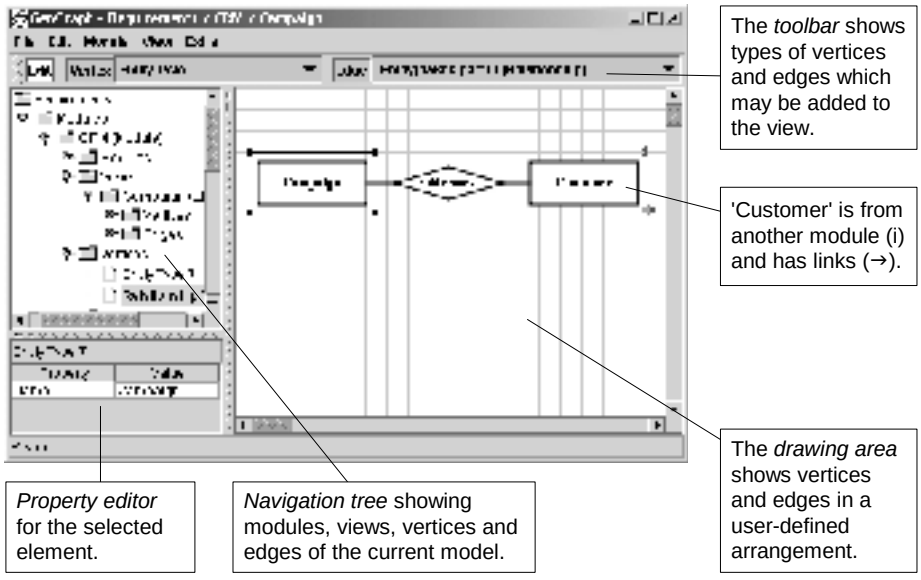

Figure 4: GenGraph showing an ER diagram

6 Design and Implementation

GenGraph's architecture has three layers. The quite independent middle layer (core) defines interfaces and classes for describing, accessing and manipulating models. The upper layer realizes the user interface. It accesses services of the middle layer. The lower layer deals with persistence. It implements interfaces which are defined in the middle layer.

The design of the interfaces for model access in the core layer is the most critical part: Every other layer depends on these interfaces. Changes to the core layer affect many other components.

The communication between core and user interface follows the model-view-controller (MVC) or observer pattern [Ga95]. Manipulations done by the user via controllers are sent to the model which validates them. If the manipulation is legal, the actual change of the model takes place, and an event is sent to registered model observers (usually views). Finally the views will update the display, and the manipulation is complete. The MVC pattern allows loose coupling between the user interface components: They do not need to tell each other about model changes, they only must listen to the model. A change of a "Name" property propagates to the navigation tree, to the drawing area and to any other component which may even be unknown at development time.

The model interface can be used by further components. We implemented several test cases as well as model-type-specific algorithms. Such algorithms can be understood as *plug-ins* which implement method-specific extensions to GenGraph. For the ERM example, a plug-in might generate a basic schema definition for a relational database if you add the attributes to the model.

The implementation of GenGraph was done in Java (JDK 1.3). Initial platform was Windows NT with Microsoft Access as database (via JDBC/ODBC). Meanwhile GenGraph was tested for and adapted to Linux, PostgreSQL and MySQL.

7 Limitations

The configuration of graphical representations is restricted to defining the name of a Java class which implements the shape and add-on functions (accessible from the context menu). While this approach is powerful with regard to adding type-specific functions like specialized property editors and extra constraints, it makes it impossible to add new graphical shapes without programming. The solution to this problem might be textual descriptions of shapes, for example using the Scalable Vector Graphics (SVG) format [An03a].

Several problems arise from the direct mapping between elements in the model and elements on the screen. In some modeling grammars, relationships between objects in certain contexts do not map to graphical symbols but restrict the way these objects may be arranged in a diagram. One example of this situation is the swim lane in UML diagrams. A semantic relationship of type "is-responsible-for" is shown by arranging activities in a column below the (name of the) actor that bears the responsibility. The arrangement has relationship semantics; there is an "invisible" edge. This cannot be configured in GenGraph thus far.

Some grammars use visual constructs where a line starts at some point of another line. For example, in the UML class diagram, a binary association is drawn as a line. If a binary association is an association class, there is a class symbol which is connected to the association by a dashed line [OM01]. This cannot be realized in GenGraph. If the binary association is to be represented as a line, it must be modeled as an edge in GenGraph. But when associations are edges, nothing can be attached to them. The user of GenGraph may choose to use an n-ary association when he wants to model an association class. The n-ary association has to be modeled as a vertex because it is shown as a diamond symbol. Therefore it is possible to draw a line between the n-ary association and the association class. But this is a deviation from the style of binary associations as defined by the UML.

GenGraph does not have a sophisticated concept of constraints. In GenGraph metamodels, restrictions can be defined for the types of vertices which may be connected by a certain type of edge, for the types of edges and vertices which may be added to a certain type of view and for the types of views which exist in a model type. You cannot express limitations to the multiplicity of edges. If you allow creating an edge of a given type between vertices of certain types, the user may create any number of edges. In a UML class diagram, the GenGraph user might connect several different association classes to an n-ary association. This is obviously not correct. You also cannot express complex restrictions which involve more than the type of one edge and its associated vertex types. The concept of constraints in GenGraph is not sufficient to prevent the user from creating scripts which are syntactically invalid according to the underlying grammar.

8 Conclusions

We suggest a module concept which structures models similar to folders in a file system but additionally promotes loose coupling between different structural units of a complex model. We introduced a concept of links which helps to visualize relationships and to navigate between different parts of a model.

We presented GenGraph, which is a generic graph editor for visual enterprise modeling. GenGraph may be configured for different modeling grammars. Different views are integrated by a single meta-model description for each type of model. A dedicated tool supports definition and modification of model type definitions. GenGraph implements the concepts of views to enables multi-perspective modeling. Furthermore, modules are used to "componentize" large models and links allow an easy navigation between different views and model aspects.

We identified several problems with a graph-based approach to describing modeling grammars and with the GenGraph implementation. Thus future work will have to address the means to describe modeling grammars. The concept of modules and links needs deeper practical evaluation.

References

[Al91] Alderson, A.: Meta-CASE Technology. In: A. Endres; H. Weber (Eds.): Software Development Environments and CASE Technology. Berlin et al. 1991, pp. 81-91.

[An03a] Andersson, O. et al.: Scalable Vector Graphics (SVG) 1.1 Specification. W3C Recommendation 14 January 2003. http://www.w3c.org/TR/SVG11/, visited on 2003-12-23.

[An03b] Anonymous: Generischer Modelleditor. http://wiseweb.wiwi.tu-dresden.de/gme/, visited on 2003-12-23.

[Be02] Becker, J.; Delfmann, P.; Knackstedt, R.; Kuropka, D.: Konfigurative Referenzmodellierung. In: J. Becker; R. Knackstedt (Eds.): Wissensmanagement mit Referenzmodellen. Konzepte für die Anwendungssystem- und Organisationsgestaltung. Berlin et al. 2002, pp. 25-144.

[Be95] Becker, J.: Strukturanalogien in Informationsmodellen - Ihre Definition, ihr Nutzen und ihr Einfluß auf die Bildung der Grundsätze ordnungsmäßiger Modellierung (GoM). In: W. König (Eds.): Wirtschaftsinformatik '95 - Wettbewerbsfähigkeit, Innovation, Wirtschaftlichkeit. Heidelberg 1995, pp. 133-150.

[Ch76] Chen, P. P.-S.: The Entity-Relationship Model - Toward a Unified View of Data. In: ACM Transactions on Database Systems 1 (1976) 1, pp. 9-36.

[Da00] Davis, R.: Business Process Modelling With ARIS: A Practical Guide. 2000.

[FL03] Fettke, P.; Loos, P.: Classification of reference models - a methodology and its application. In: Information Systems and e-Business Management 1 (2003) 1, pp. 35-53.

[Fr02] Frank, U.: Multi-perspective Enterprise Modeling (MEMO) - Conceptual Framework and Modeling Languages. Proceedings of the 35th Hawaii International Conference on Systems Science (CD-ROM). Hawaii 2002

[Ga95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns - Elements of Reusable Object-Oriented Software. Reading, MA, et al. 1995.

[GR00] Green, P.; Rosemann, M.: Integrated Process Modeling: An Ontological Evaluation. In: Information Systems 25 (2000) 2, pp. 73-87.

[Gr03] Greiffenberg, S.: Methodenentwicklung. Dissertation, Technische Universität Dresden, Dresden 2003 (submitted on 2003-01-17).

[Le01] Ledeczi, A.; Maroti, M.; Bakay, A.; Karsai, G.; Garrett, J.; Thomason, C.; Nordstrom, G.; Sprinkle, J.; Volgyesi, P.: The Generic Modeling Environment. IEEE International Workshop on Intelligent Signal Processing, 24-25 May 2001. Budapest, Hungary 2001

[LF01] Loos, P.; Fettke, P.: Towards an Integration of Business Process Modeling and Object-Oriented Software Development. In: I. Ivan; I. G. Rosca (Eds.): Information Society - The Proceedings of the Fifth International Symposium on Economic Informatics - IE 2001, Bucharest, May 9-12, 2001, Proceedings. Bucharest 2001, pp. 835-843.

[LS94] Loos, P.; Scheer, A.-W.: Graphical Recipe Management and Scheduling for Process Industries. In: T. O. Boucher; M. A. Jafari; E. A. Elsayed (Eds.): Rutgers' Conference on Computer Integrated Manufacturing in the Process Industries (Proceedings CIMPRO '94, April 25-26, 1994). Piscataway 1994, pp. 426-440.

[Me03] MetaCase: Company website. http://www.metacase.com/, visited on 2003-12-23.

[My98] Mylopoulos, J.: Information Modeling in the Time of the Revolution. In: Information Systems 23 (1998) 3/4, pp. 127-155.

[OM01] OMG: Unified Modeling Language Specification: Version 1.4. Needham 2001.

[RG02] Rosemann, M.; Green, P.: Integration Multi-Perspective Views into Ontological Analysis. In: L. Applegate; R. Galliers; J. I. DeGross (Eds.): Twenty-Third International Conference on Information Systems. Barcelona, Spain 2002, pp. 618-627.

[Sa00] Sapia, C.; Blaschka, M.; Höfling, G.: GraMMi: Using a Standard Repository Management System to Build a Generic Graphical Modeling Tool. Proceedings of the 33rd Hawaii International Conference on Systems Science (CD-ROM). Hawaii 2000

[Sc98] Scheer, A.-W.: ARIS - Business Process Frameworks. 2. ed., Berlin et al. 1998.

[SH92] Scheer, A.-W.; Hars, A.: Extending Data Modeling to Cover the Whole Enterprise. In: Communications of the ACM 35 (1992) 9, pp. 166-172.

[Sm91] Smolander, K.; Lyytinen, K.; Tahvanainen, V.-P.; Marttiin, P.: MetaEdit - A Flexible Graphical Environment for Methodology Modeling. In: R. Anderson; J. A. Bubenko jr.; A. Sølvberg (Eds.): Advanced Information Systems Engineering. Third International Conference CAiSE '91, Trondheim, Norway, May 1991, Proceedings. Berlin et al. 1991, pp. 168-193.

[SR98] Schuette, R.; Rotthowe, T.: The Guidelines of Modeling - An Approach to Enhance the Quality in Information Models. In: T. W. Ling; S. Ram; M. L. Lee (Eds.): Conceptual Modeling - ER '98 - 17th International Conference on Conceptual Modeling, Singapore, November 16-19, 1998, Proceedings. Berlin et al. 1998, pp. 240-254.

[WW02] Wand, Y.; Weber, R.: Research Commentary: Information Systems and Conceptual Modelling - A Research Agenda. In: Information Systems Research 13 (2002), pp. 363-377.