UNIVERSITÄT KOBLENZ · LANDAU

Institut für Wirtschaftsinformatik
Fachbereich Informatik
Universität Koblenz-Landau

ULRICH FRANK

TOWARDS A STANDARDIZATION OF OBJECT-ORIENTED MODELLING LANGUAGES?

Mai 1997

Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 3

UNIVERSITÄT
KOBLENZ · LANDAU

Institut für
Wirtschaftsinformatik

Fachbereich Informatik
Universität Koblenz-Landau

ULRICH FRANK

TOWARDS A STANDARDIZATION OF OBJECT-ORIENTED MODELLING LANGUAGES?

Mai 1997

Arbeitsberichte des Instituts für Wirtschaftsinformatik Nr. 3

Die Arbeitsberichte des Instituts für Wirtschaftsinformatik dienen der Darstellung vorläufiger Ergebnisse, die i.d.R. noch für spätere Veröffentlichungen überarbeitet werden. Die Autoren sind deshalb für kritische Hinweise dankbar.

The "Arbeitsberichte des Instituts für Wirtschaftsinformatik" comprise preliminary results which will usually be revised for subsequent publications. Critical comments would be appreciated by the authors.

Anschrift der Verfasser/
Address of the authors:

Prof. Dr. Ulrich Frank
Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
D-56075 Koblenz

Arbeitsberichte des Instituts für Wirtschaftsinformatik
Herausgegeben von / Edited by:

Prof. Dr. Ulrich Frank
Prof. Dr. J. Felix Hampe
Prof. Dr. Stefan Klein

Bezugsquelle / Source of Supply:

Institut für Wirtschaftsinformatik
Universität Koblenz-Landau
Rheinau 1
56075 Koblenz

Tel.: 0261-9119-480
Fax: 0261-9119-487
Email: iwi@uni-koblenz.de
WWW: http://www.uni-koblenz.de/~iwi

Abstract

Object-oriented modelling is used in a growing number of commercial software development projects. However, the plethora of approaches and corresponding CASE tools still prevents corporate users to migrate to object-oriented software development methods. Against this background the recent efforts of the Object Management Group (OMG) to standardize object-oriented modelling languages seem to promise substantial benefits: Not only will a standard allow to easily port a model from one CASE tool to another, it will also protect investment in training. In addition, it is a prerequisite for standardized business object models which - in the long run - may substantially improve the economics of developing and maintaining corporate information systems. Nevertheless there are objections against a standardization at present time: It is questionable, whether the state of the art in object-oriented modelling is mature enough to allow for standardization. Furthermore standardization holds the risk to discourage further innovations.

In order to analyze the state of the art, this report will first give an overview of previous approaches to object-oriented modelling. To get an idea of the present state of the art, the essential characteristics of the modelling languages currently under review at the OMG are briefly characterized - together with an additional approach that recently has gained remarkable attention. This will lead to a number of research challenges still to overcome. Finally the report offers a subtly differentiated answer to the question whether or not it is time for standardizing object-oriented modelling languages.

1. Motivation

During the last decade, object-oriented software development has been adopted in the academic world with remarkable enthusiasm. This is different with corporate software development: Although there is a tremendous marketing push caused by vendors, consultants, and specialist journals, many companies are hesitating to introduce object-oriented methods[1]. This is for comprehensible reasons: At present time, there are still serious inhibitors of object-oriented software development to overcome. There is lack of *mature technology*. While there are object-oriented programming languages that come with reliable compilers - sometimes embedded in rather convenient environments, today's object-oriented database management systems are usually not suited to replace relational database management systems within corporate information systems. Furthermore there is *lack of competence*. In order to exploit the benefits offered by the object-oriented paradigm, and to avoid its pitfalls, it is necessary for the developers to gain a deep understanding of the essential concepts. It is certainly not too daring to assume that most software developers do not have these skills today. From a managerial point of view it is risky to provide for the training that is required to develop these skills: Not only that training is expensive, and its success is hard to predict, furthermore there is still *lack of standards*. This is the case for object-oriented programming languages and database management systems, as well as for specialized development methods. For this reason protection of investments, both in professional training and in technology (CASE, compilers, etc.), is usually not satisfactory.

Although many companies are still hesitating to introduce object-oriented methods and technologies, there is no doubt that the future (that is at least the next ten years) of corporate software development will be more and more object-oriented - simply because there is no alternative paradigm of similar relevance. In other words: The already big market for object-oriented concepts, training, and technologies can still be expected to grow at a fast pace. However, in order to encourage corporate investments, it is not sufficient to develop mature technologies. In addition it is crucial to provide reliable standards that foster interoperability and reusability, and protect investments at the same time. Only recently it became apparent that the various dialects of object-oriented modelling will eventually be replaced by an industry standard modelling language. On the one hand, three of the best known authors of modelling methods decided to merge their efforts into one company in order to consolidate their methods into one "unified method" ([Rum96b]). On the other hand, the "Object Management Group" (OMG) issued a request for proposals for object analysis and design ([OMG96]). The submissions were due at January, 17th, 1997. The review process can be expected not to be completed before the end of 1997. The OMG's request for proposal was the cause for writing this report which will focus on the following questions:

- What is/could be subject of standardization in the area of object-oriented modelling?
- What are the prospects and risks that are related to standardization?
- What is the OMG's intention?
- What are the characteristics of the official proposals to the OMG?

1. To our knowledge there has not been a single representative study on an international scale. A recently conducted empirical study ([Sch97]) indicates that less than 10% of the german insurance companies use object-oriented software development methods.

- Is the current state of the art mature enough to allow for standardization?

This paper is intended to be followed by a later report ([FrPr97]) which will compare two modelling languages, the UML ([Rat97a] .. [Rat97j]) that has been proposed to the OMG, and the OML [FiHe96].

2. Object-Oriented Modelling: Need for Standardization?

During the last 10 years a remarkable number of approaches to support object-oriented software development on a conceptual level have been published. The majority of them evolved in an academic setting. Unfortunately, most of them are not documented in a comprehensive way. Furthermore the differences between some approaches seem to be marginal. For this reason, it would not make much sense to consider all of the approaches we have encountered. Instead we will give an overview that is restricted to a few characteristics. With the increasing popularity of object-oriented software development, object-oriented modelling has gained more and more commercial attention - resulting in a growing market for specialized services and tools. Against this background - a new paradigm, attractive both from an academic and a commercial point of view - it is not surprising that there is remarkable diversity in concepts and terminology. In order to discuss the benefits and risks of standardization we will define essential subjects of object-oriented modelling - such as method, model, metamodel, etc.

2.1 Overview of Previous Object-Oriented Modelling Methods

Object-oriented software development in general, object-oriented modelling in particular has been a rather popular topic both for conferences and specialized journals. This may indicate a vivid period of research - or simply the usual hyper-activity caused by the emergence of a new research paradigm (or at least what seems to be a new paradigm). To give an impression of the amount of research, we have gathered more than 40 approaches which focus on various aspects of object-oriented analysis and design. The following list is compiled from a previous survey ([Fra93]) and additional sources ([Big97], [Obj97]). It is not meant to be complete. Missing information about the name of an approach or about its tool support means that there was no information available. It does not necessarily imply that a name or tool does not exist.

Author/Institution	*Name*	*Refs*	*Type of Pub.*	*Tool Support*
Ackroyd/Daum		[AcDa91]	A	
Alabisco		[Ala88]	C	
Berard		[Ber93]	T	
Bailin		[Bai89]	A	
Booch		[Boo94]	T	Y
Buhr		[Bu84]	T	
Cherry	PAMELA 2	[Che87]	R	
Coad/Yourdon	OOA/OOD	[CoYo91], [CoYo90]	T T	Y
Coleman	Fusion	[Col94]	T	Y

Author/Institution	*Name*	*Refs*	*Type of Pub.*	*Tool Support*
Cunningham/Beck		[CuBe86]	C	
Desfray	Object Engi-neering	[Des94]	T	
Edwards	Ptech	[Edw89]	T	Y
Embley et al.		[EmKu92]	T	
ESA	HOOD	[ESA89]	M	
Felsinger		[Fel87]	R	
Ferstl/Sinz	SOM	[FeSi91], [FeSi90]	A A	Y
Firesmith		[Fir92]	T	
Goldberg/Rubin	OBA	[GoRu92]	A	Y
Graham		[Gra91]	T	
Halladay/Wiebel		[HaWi93]	T	
Henderson-Sellers/Edwards		[HeEd94], [Hen92]	T T	
IBM		[IBM90]	R	
Jacobson et al.	OOSE	[JaCh92]	T	Y
Johnson/Foote		[JoFo85]	A	
Kadie		[Kad86]	R	
Kappel/Schrefl		[KaSc91]	A	
Lee/Carver		[LeCa91]	A	
Liskov/Guttag		[LiGu86]	T	
Martin/Odell		[MaOd92], [MaOd95]	T T	
Masiero/Germano		[MaGe88]	A	
McGregor/Sykes		[McSy92]	T	
Meyer		[Mey88]	T	
Mullin		[Mul89]	T	
Nielsen		[Nie88]	R	
Odell		[Ode92]	A	
Page et al.		[PaBe89]	C	
Rajlich/Silva		[RaSi87]	R	
Robinson		[Rob92]	T	

Author/Institution	*Name*	*Refs*	*Type of Pub.*	*Tool Support*
Rumbaugh et al.	OMT	[Rum91]	T	Y
Seidewitz/Stark		[SeSt87]	A	
Shlaer/Mellor		[ShMe92], [ShMe88]	T T	Y
Velho/Carapuca	SOM	[VeCa92]	C	
Walden/Nerson	BON	[WaNe95]	T	Y
Wasserman et al.		[WaPi90]	A	
Wirfs-Brock/Wilkerson		[WiWi90]	T	

T Textbook
A Article in a Journal or Reader
M Manual
R Research Report
C Conference Paper

Fig. 1: Overview of past approaches to object-oriented modelling

2.2 Terminology

One of the shady sides of object-oriented software technology is the lack of a common, and consistent terminology. This fact may be contributed to various reasons:

- The specialized research is still in a rather early stage.
- There are many players - with different backgrounds and motives - who influence the evolving terminology: software-engineers, computer scientists, consultants, tool vendors, etc.
- There is an "inherent" ambiguity that results from the different levels of abstraction that are characteristic for the various phases of software development - resulting in subtle terminological traps that everybody in the field knows: Sometimes somebody, who is talking about an "object", actually means a "class", within another context he may think of a particular instance ...

The definition of a sound terminology is certainly one of the key contributions to be expected from a modelling approach. For this - and for some other reasons - we do not have to discuss the whole range of terminological problems at this point. However, we do need a number of terms to categorize the approaches to be evaluated: method, methodology, model, metamodel, modelling language, etc.. Since they are essential terms not only of computer science but of science and philosophy in general, we do not intend to provide a comprehensive definition. Instead, we will briefly describe the interpretations we favour within the context of this working paper.

A *method* in general is a systematic approach that helps with solving a class of problems. The

term "systematic" is to express that usually a class of problems is divided into classes of subproblems, together with techniques, or methods to solve them ([Lor84]). Furthermore a method includes a more or less rigid temporal order of the various problem solving activities together with the required skills and resources. In general, it is a key characteristic of scientific work to apply, to review and to revise methods. A software development method is a method that is supposed to support the systematic development of a class of software.

The term *methodology* in contrast denotes a study or a theory of methods in the sense that it includes a set of methods together with guidelines or principles to evaluate, select, apply, and develop them ([Mit84]). Many authors use "methodology" to denote a method. The methodology that is focusing on scientific methods in general, or within a particular discipline, is also called epistemology, or theory of science.

A *model* in general is a description of an existing or potential domain, where domain denotes either a part of the "real", perceivable world, an intellectual construction, or a mixture of both. It is essential for this description that it is an abstraction of the related domain. Certain aspects of the domain are not taken into account on purpose. For this reason a model allows to concentrate on relevant aspects, thereby helping to simplify the analysis of a complex matter. Relevance depends on the particular purpose a model is to serve - such as analysis, description, explanation, understanding, design. In the context of software development we call a model *conceptual* if it is focusing on relevant concepts of the application domain, neglecting implementation-related issues. A concept is a notion, an idea that allows to identify and describe classes or categories of real world entities by defining their essential features. According to Mylopoulos and Levesque ([MyLe84], p. 11), a conceptual model should provide "... descriptions of a world enterprise/slice of reality which correspond directly and naturally to our own conceptualizations of the object of these descriptions." Notice that this does not exclude models of technical systems, such as machines, processors, etc. Hence, a concept is an abstraction of a set of particular occurrences. In other words: Whether or not a model is regarded as being conceptual will also depend on the intention of its creators - and the perception of the observers. Usually, but not necessarily, a conceptual model includes a graphical representation.

A description requires the existence of a *language*. While in principle any language, in particular a natural language, can be used to describe a model, there are languages that were especially introduced for the purpose of describing models. We call these languages *modelling languages*. A language may be formalized or not. There may be various representations - such as text or graphics - to express propositions of a language. Modelling languages that are used for software development usually come with a graphical notation. Languages of that kind should be suited to fulfil the requirements of conceptual modelling (focus on "natural" concepts). At the same time they should allow for a straightforward transformation to a programming language. Object-Oriented modelling is an attempt to provide modelling languages which fulfil these divergent requirements.

A *metalanguage* is a language that serves to speak about a language, like its symbols, syntax and semantics. One inherent source of ambiguity in natural languages is the lack of a rigid separation of meta- and object language. In order to avoid ambiguity, formal languages require a clear separation of meta- and object level. That does not mean, however, that a formal language and its corresponding metalanguage need to be different in terms of their symbols or their syntax. Defining a language with a formal metalanguage promises a number of benefits. A metalanguage that is used to specify a number of object-level languages helps to *integrate* these lan-

guages: Information represented in the various object-level languages can be exchanged by referring to concepts defined in the common metalanguage - as long as the involved languages have those concepts in common. Furthermore a metalanguage can help with *reusability* and *flexibility*: Software that operates on a particular metalanguage, can be used to generate software that operates on the object-level language (like a compiler compiler, or Meta CASE, see for instance [EbSü96]). A metalanguage may be described by yet another metalanguage which is sometimes called meta metalanguage, and so forth. A meta metalanguage provides the same advantages for handling metalanguages as a metalanguage does for object-level languages.

A *metamodel* is a model representing a metalanguage of a class of modelling languages. It serves to describe the symbols, syntax, and semantics of these modelling languages in a more or less rigid way. Depending on the metalanguage it is based on a metamodel may be formal or not. A metamodel may also include instructions or recommendations of how to render a model for the viewer, in other words: the notation of the modelling language. A metamodel may be rendered in the same notation as its subordinated models. The benefits provided by metamodels or meta metamodels correspond to those of metalanguages and meta metalanguages respectively.

A *modelling method* is a method that supports the construction and use of a class of models. It will usually include some sort of metamodel or metalanguage that, however, does not have to be formalized. Furthermore it may include some of the following aspects (this list is not meant to be complete):

- more or less detailed decomposition into subproblems together with problem solving activities
- temporal/causal order of problem solving activities
- profiles of roles required to perform those activities
- information required, together with heuristics to gather it
- documents/models to be produced
- criteria or even metrics to evaluate models
- further support for project management
- examples of how to apply the method
- generic models for certain domains that can be reused and modified

A *modelling methodology* is a study or a theory about modelling methods within a certain area. Its subject is a set of modelling methods together with criteria/principles to use, analyze, and revise them. Computer science in general could be regarded as a modelling methodology, since one of its essential research goals is to provide appropriate abstractions for a class of problems - together with solutions to these problems.

Fig. 2 shows a semantic net that illustrates the terminology. It is not intended to provide a complete and precise definition.

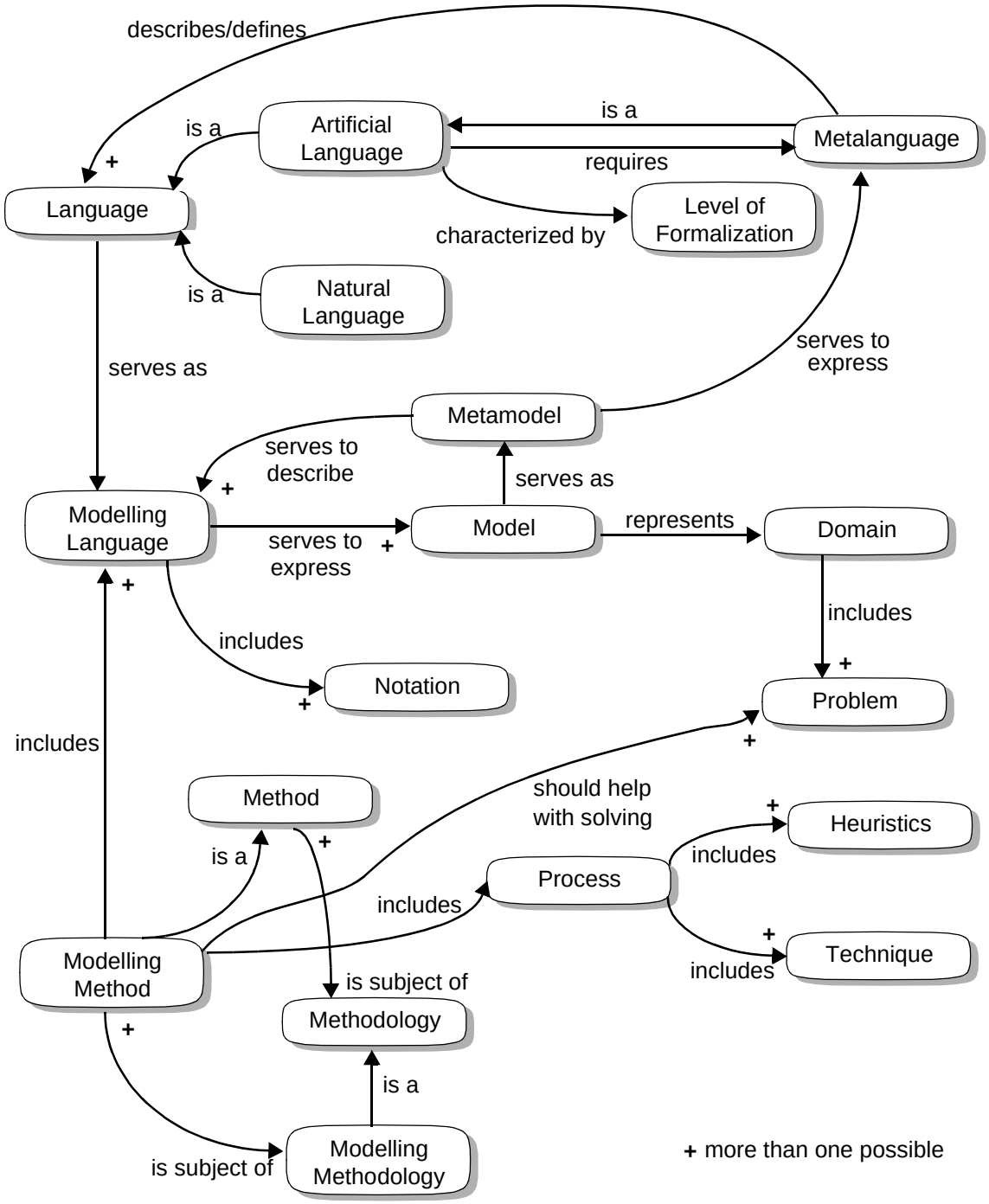

Fig. 2: Concepts within object-oriented modelling

2.3 Subjects, Benefits, and Pitfalls of Standardization

While conventions and standards have played an important role in information technology for long, standards have gained a still increasing attention in the nineties. It is remarkable that today's information managers are not only interested in suitable standards. Instead they are con-

centrating more than ever on the question which attempt will finally make it as the one and only global standard for a certain area - thereby almost paralyzing their ability to make decisions. We believe that this is both for economic and psychological reasons. The emergence of microcomputers and so called industry standards (which are sometimes in fact proprietary specifications) have demonstrated the tremendous impact of scale economies on the production costs per unit of hardware, and especially of software. This experience seems to have fostered a wide spread believe that in the end only one standard within a particular scope will survive. Not only vendors, but also (organizational) buyers are obviously impressed (not to say "obsessed") with this "the winner takes it all" mentality. From an economic point of view it makes a big difference, whether a "standard" results from a dominating proprietary technology, or whether it is "open" in the sense that its specification may - in principle - be accessed, and used by everybody. The specification of open standards has been the business of national or international standardization bodies for long. There have also been standardization initiatives which were organized by the industry itself, mainly for two reasons: to speed up the standardization process and to gain better control over it. Before we will have a look at one particular initiative that is of outstanding importance within the area of object-oriented technologies, namely the OMG, we will first analyze what could be subject of standardization in the area of object-oriented modelling.

Standardizing a (Modelling) Methodology

A methodology in the sense of how we introduced the term is by no means a subject suitable for standardization. It is common sense that a theory should be permanently evaluated against the domain it has been introduced for - and revised if necessary. This epistemological process would be compromised by "standard theories". Standardizing a study would be equally bizarre. There is, however, no doubt that a common terminology is rather important for the evolution of methodologies as well.

Standardizing a (Modelling) Method

Similar to methodology it is not obvious, why it would make sense to standardize "a systematic approach that helps with solving a class of problems". Instead it seems to be more appropriate that anybody, who has to solve a problem, may try the available methods and finally take the one that suits him best. From an academic point of view such a standardization seems to be bizarre: Scientific progress relies heavily on permanent reviews and - eventually - modifications of existing methods. Freezing a method by standardizing it would contradict this process. However, while this is certainly true from an epistemological point of view, things tend to be more subtle in practice. For one, we have to take into account that defining and prescribing stable methods - with more or less rigour - is an essential prerequisite of professional collaborative work ([Wei79], pp. 15). Organizing allows for task specific training, and is a prerequisite for core management functions such as planning and control. Does that mean, however, that there is need to standardize business processes? There is certainly a clear objection against standardizing the organisation of processes: Organizing its business should be part of the core competencies of almost any company, and therefore it is an essential part of its competitiveness. On the other hand we currently witness a tendency toward standard business processes, promoted by technologies such as integrated business software, or workflow management systems. Reorganising business processes by applying certain standard patterns facilitates the efficient introduction of corresponding software. The question of the appropriate level of standardization within business processes is a complex matter that we cannot analyze here in depth.

There is, however, another motivation for standardizing methods: Defining steps and guidelines for documenting their results may help with reconstructing a business process. Thereby customers may get an idea of the efficiency of a process and how the process contributes to the overall quality of a product or service. For the companies that apply the method it may help with increasing the awareness of organizational efficiency and thereby fostering more efficient processes. Recent examples of efforts to standardize methods or processes are ISO 9000 ([KeJa96], [Sch94]) and the "V-Modell" (a process model for software development, defined by the German forces "Bundeswehr" [BrDr93]). Notice that standard methods such as ISO 9000 or the "V-Modell" define a temporal order of documents and their corresponding (modelling) languages (which may be structured forms of the natural language) rather than providing a precise definition of the involved tasks and how to perform them.

We can summarize that standardizing a software development method in every detail is certainly no option. A method's potential depends on the problem it is used for, the skills and attitude of the method users, the available resources, etc. For this reason it is hard to imagine that there could be one best (object-oriented modelling) method at a point in time. Furthermore you have to take into account that methods will (and should) evolve over time. At last it would certainly be too expensive to enforce a detailed standard method, since that would cause expenses for monitoring which nobody could afford. Therefore it is definitely no good idea to standardize a method. However, in order to foster comparability, quality, and professional training, it can make sense to standardize the documents (models) that should be produced within a method - which in turn will certainly have an impact on the methods that use these documents.

Standardizing a (Modelling) Language

The notion of a language implies the idea of conventions which may be more or less rigid. Otherwise a language could not be used for the purpose of communication. Standardizing a language implies its formalization which in turn requires a formal metalanguage. It is evident that the metalanguage itself has to be precisely defined, too. In order to avoid a *regressum ad infinitum* there has to be at one point a reference to already existing well defined languages and concepts - like a logical calculus or standardized types/classes.

What are the benefits of standardizing modelling languages used within the process of software development? The potential benefits are obvious: Standardized modelling languages will allow for better protection of investments in technologies that depend on those languages. This is also the case for investments into training. Standardized modelling language will also improve the chance for exchanging models and interoperability in general, or - in other words - for system integration. Furthermore standardized modelling languages will foster reusability of existing models, for instance of generic or reference models developed for certain problem areas or domains. A modelling language will usually include a graphical notation which, however, does not have to be part of a standard: From a logical point of view the notation is irrelevant, since it does not contain any formal semantics. Nevertheless a common notation will facilitate the communication between all people using the modelling language. Therefore it might be reasonable to enhance a standard with recommendations for a notation.

Despite its potential benefits, standardizing modelling languages faces a severe challenge: What are the requirements a modelling language should fulfil in order to be useful within a longer time frame? We will analyze this question later (see 4.3, [FrPr97]). One thing, however, is for sure: There is no easy answer to it. This leads to another question we will also look at later in this report: Is it already the time to standardize object-oriented modelling languages?

Standardizing (Object) Models

During the last years an attractive vision has gained remarkable popularity: the emergence of a market for standardized (implemented) classes for corporate information systems, often referred to as "business objects". It is not possible to establish a market for business classes/objects that have been specified/implemented independent from each other: Usually objects, which are to be used in a certain domain, will interact with other objects/classes. Furthermore it is important to represent business objects in an illustrative way. For these reasons the vision of providing standardized business objects in the long run requires application level object models together with a corresponding implementation - either as a class library or as a framework (see [LeRo95]). Standardizing application level object models certainly implies the existence of a standardized modelling language. While standardized domain level object models offer fascinating prospects on reusability and highly integrated systems, the corresponding standardization process requires complex decisions to be made: about the scope of a model, about its level of abstraction and semantics, about the modelling languages to be used, etc.

3. The Current Situation: Between Demand Pull and Technology Push

Our overview of approaches to object-oriented modelling (see 2.1) gives an idea of the extensive research activities in this field. However, there has been a concentration on a small number of approaches. Some textbooks achieved an impressive circulation. Although we do not know of any sufficient empirical evidence, it seems that four methods by far received the most attention: OMT ([Rum91]), Booch ([Boo94]), OOSE ([JaCh92]), Coad/Yourdan ([CoYo91], [CoYo90]). All of these methods, as well as related services and tools, were marketed in companies the authors were affiliated with. From an economic point of view it is remarkable that even this small number of competing approaches turned out to be too big for the market: Many corporate users hesitated to introduce one of the methods as long as it was not clear which one would survive in the long run. In this situation, Rational Software, the company behind Booch, began an initiative to form a strategic alliance with some of the key players. First Rumbaugh became a partner. About a year later, in 1995, Jacobson's company, Objectory, joined Rational as well ([Rat97c]). Soon Rational announced to develop a "unified method", promising to make it a compilation of the best features offered by the original methods ([Rum96b]).

3.1 Background of Current Standardization Efforts

Back in 1992 the OMG had addressed a similar objective by founding an "Object Analysis and Design Special Interest Group" [KaCh92]. The OMG itself was founded in 1989 by vendors and IT users. It is committed to define reference specifications for object-oriented technologies. At present time the OMG has more than 700 members (http://www.omg.com). The activities of the OMG used to be focused mainly on specifying an architecture for distributed object systems, called "Object Management Architecture" (OMA), together with its components, such as the "Common Object Request Broker Architecture" (CORBA, [Ben93]) and a low level common object model. However, in order to foster the development of software that is OMA compliant, conceptual support is required as well. To provide for appropriate support, the OMG launched the "Object Analysis and Design Special Interest Group" mentioned above. But this group did not succeed to meet its central objective: "To formalise the definitions of the concepts used for analysis and design" ([KaCh92], p. 12). Apparently for this failure, the OMG issued a request for proposals for object analysis and design last year ([OMG96a], see 3.2).

While the OMG's primary focus is on CASE tool interoperability (see below), there is another reason for the OMG to standardize object-oriented modelling languages. It is related to the essential vision of the OMG: transparently communicating objects - distributed all over the world - no matter which platform they reside on, no matter which programming language they are implemented in. The OMA and its components specify an architecture of a system level infrastructure on top of distributed, heterogeneous environments. While this is certainly a prerequisite for distributed object systems (as operating systems are a prerequisite for local applications), it is not sufficient to allow for transparent communication on the application level. For instance: If you create an object representing an order, you cannot transmit it to another company that does not know the corresponding class.

Domain specific classes, however, are beyond the scope of the object model specified by the OMG ([OMG97]). Instead it is restricted to base level classes - similar to the types/classes usually provided with programming languages. Nevertheless the OMG is stressing the importance of domain level classes, namely business objects (see [OMG96b]). It defines business objects as "... representations of the nature and behavior of real world things or concepts in terms that are meaningful to the business. Customers, products, orders, employees, trades, financial instruments, shipping containers and vehicles are all examples of real-world concepts or things that could be represented as business objects. Business objects add value over other representations by providing a way of managing complexity, giving a higher level perspective, and packaging the essential characteristics of business concepts more completely. We can think of business objects as actors, role-players, or surrogates for the real world things or concepts that they represent." ([OMG95a], p. 5). In order to prepare the grounds for standardized business objects the OMG founded a "Business Object Management Special Interest Group" (BOMSIG). For this group the need for application level object models seems to be evident: "The business object model becomes the driving force of the information system, not the technology. The business object model is derived directly from the shape of the enterprise." ([OMG96b], p. 5) The OMG requires submitters to "... address the relationship between the meta-model constructs that represent OA&D objects and the metadata specifications of the Business Object Facility (BOF) which represent the business semantics of ... Business Objects." ([OMG96a], p. 15)

3.2 OMG's Request for Proposals: Subject and Ambiguities

Among the vendors of object-oriented modelling methods and tools Rational Software has gained an outstanding position. The strongest competition in the field seemed to be between the Booch method, OMT, and OOSE. Now that those competitors merged into one company, there is hardly any competitor left. Furthermore Rational has formed an alliance with Microsoft: While Rational has aquired a number of Microsoft's development and testing tools, Microsoft in turn will include Rational's modelling tools into future versions of its development environments (http://www.rational.com, [You97]). Taking all of those aspects together it is not surprising that analysts predict that the unified method announced by Rational will be the future industry standard: "... I think the battle for dominance of the object-oriented visual modeling marketplace is over: Rational wins." ([You97], p. 16). However, with the OMG's request for proposals ([OMG96a]) this situation has changed a little. Since Rational is a member of the OMG, and submitted a proposal as well, it should be committed to accept the standard finally proposed by the OMG. Therefore other submissions should have a chance to influence the future standard, too - although we do not believe that the OMG will ignore Rational's position at the marketplace.

The OMG's request for proposals, "Object Analysis&Design RFP-1" ([OMG96a]) was issued at June 6th, 1996. The initial submissions were due at January 17th, 1997. During the evaluation process, the submitters will get the chance to revise their proposals. These revised submissions were due at April 15th, 1997. Within the OMG's overall plan, object-oriented modelling is part of the OMA's so called "Common Facilities" ([OMG95b]). It is remarkable that - different from Rational's original approach - the OMG has never intended to standardize a method. Instead its primary focus is on the standardization of modelling languages. Therefore the submissions were to specify "the interfaces and semantics needed to support the creation and manipulation of a core set of OA&D models that define the structure, meaning, and behaviour of object applications within the OMG Object Management Architecture. These models may be

- static models (also known as structural models, e.g., class models, type models, instance models)
- dynamic models (also known as behavioural models, e.g., state-transition models, object interaction models, object dataflow models)
- usage models (e.g., use-case models)
- architectural models (also known as system composition models, e.g., subsystem models, modules-component models, process maps)

or other models of value during analysis and design." ([OMG96a], p. 3)

As already mentioned above, the OMG is primarily interested in CASE tool interoperability:

> "Users of a conforming tool will have confidence that they can import/export models from/to other conforming tools. Use of a common OA&D facility will help ensure that the content of an OA&D model can be combined with or imported into another OA&D model where it makes semantic sense, thereby improving consistency and traceability across the various OA&D models describing a system. Each of these models focuses on a particular facet of the system; together they describe the multiple facets of the system. The ability to integrate OA&D tools from multiple vendors should give users assurance that they can tailor their OA&D environments to meet their needs that their investments in OA&D tools can be well leveraged as their development environments grow and evolve, and that they can exercise freedom of choice in selection and deployment of OA&D tools." ([OMG96a], pp. 3)

Note, however, that the OMG is also thinking of models as a medium to foster communication between people: "While the primary focus of this RFP is OA&D tool interoperability, the OA&D facility also includes a set of notations that provide the foundation for enabling a core set of OA&D diagrams, each of which describes a model, to be communicated between people without semantic loss." ([OMG96a], p. 4) The OMG's statements about the purpose of the "OA&D facility" leave a number of questions:

What is the precise objective of the intended standard?

The explicit reference to tool interoperability indicates that the standard is to describe metamodels for those models managed by CASE tools. On the other hand the OMG stresses the conceptual level as well: "An object analysis and design model (hereafter called simply model) is defined as an object that represents the semantics of an analysis or design of a system. A model is used to define and specify the semantics of interest in a particular aspect of the domain." ([OMG96a], p. 13) Note that both types of specifications are usually different. While a

model that is managed by a tool should be suited to incorporate the semantics defined in the corresponding modelling languages, it will also include additional information, like for versioning, user management, etc.

What is the scope of the (meta) models to be specified?

While the request for proposals provides examples for a number of models that might be part of the standard ("class models", "instance models", "state-transition models", "use-case models", etc.), it does not tell the level of semantics they should incorporate. In other words: What are the stages of the software lifecycle to be covered (in part, or complete) by the modelling languages to be defined? There is evidence that the OMG does not plan a rigid standard anyway:

> "The OA&D facility should not constrain the processes and techniques that OA&D methods use to extract the information needed to determine the semantic content of OA&D models, to ensure the integrity and validity of the content of those models, or to evolve or optimize those models. Nor should the facility constrain OA&D methods from introducing models in addition to the core set." ([OMG96a], p. 4)

How does the intended standard relate to other standards?

There is no doubt that the standard should be compliant to existing OMG specifications (such as Interface Definition Language (IDL) - which, according to OMG, may imply an extension of the current IDL specification). However, there are at least two standardization efforts outside the OMG which are concerned with subjects that are obviously related to analysis and design.

Firstly, there is the CASE Date Interchange Format (CDIF), defined by the CDIF division within the Electronic Industries Association (EIA). CDIF is dedicated to support the exchange of models managed by CASE tools. CDIF does not specify the metamodel a tool has to use itself. Instead it is restricted to the external representations that a tool has to support with its interfaces. The following example illustrates that CDIF's emphasis is clearly on the information typically required by tools, not on the conceptual level ([Ern96]):

```
(Attribute ATT01
    (Name "Current Balance")
    (CreationDate "951102")
    (CreationTime "08:00:01.1234")
 )
```

With the ISO "Information Resource Dictionary System" (IRDS, [ISO90]) and the "Portable Common Tool Environment" (PCTE, [Wak93]), an initiative launched by tool vendors, two major standardization efforts are committed to using the CDIF metamodel ([CDI96]).

Secondly, there are the activities of the Workflow Management Coalition (WfMC). Similar to the OMG the WfMC is an industry consortium. It aims a a set of standards that are to allow for "open" workflow management systems. In order to achieve this goal the WfMC intends to define a set of interfaces for the components to be integrated (such as office application, DBMS, and so called workflow engines) as well as a "Workflow Process Definition Language" (WPDL, [WFM96]). The idea behind this approach is similar to entity relationship modelling and SQL respectively: A workflow type can be modelled using an appropriate modelling language provided by a corresponding tool. The modelling language will then translate into the WPDL

which can be interpreted by any workflow engine that has been certified by the WfMC (see fig. 3)

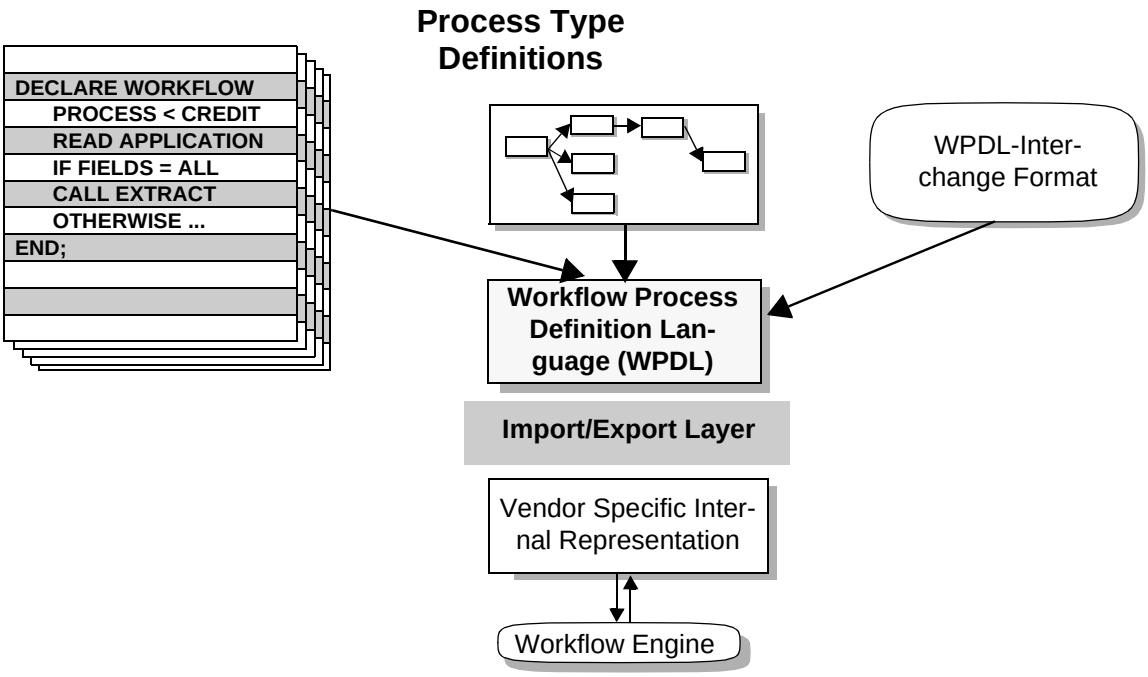

Fig. 3: Process Modelling and Specification within the WfMC Approach ([WfM96], p. 5)

There is a clear relationship to the mission of the OMG's "Object Analysis&Design Facility": In order to model a workflow, you need to describe the information required and produced within that workflow - for instance by referring to an object model. While both, the EIA ([CDI96]) and the WfMC ([And96], p. 21) refer to the OMG, to our knowledge the OMG itself does not explain its relationship to those approaches, nor does it even mention them.

4. Object-Oriented Modelling Languages: Is there a State of the Art?

Standardizing technologies or concepts requires a mature level of corresponding research activities. With object-oriented modelling, it is impossible to identify a unique scientific community that is dedicated to this subject. This is for a number of reasons. Firstly, object-oriented concepts have various roots, including programming languages, database design, and artificial intelligence. Secondly, there are contributions from authors with either a commercial or a more academic background. Therefore we will first focus on what you could call the "dominating" state of the art: recently published approaches which are either well known and/or promoted by powerful commercial consortia. For this purpose we will have a look at the languages submitted to the OMG early this year. While it is arguable that all of them are important enough to be part of the dominating state of the art, they have at least a potential impact on the standard, the OMG is going to specify. Our brief analysis is mainly motivated by the following question: Do the proposals give the impression of a convergent (and mature) field, or are the various ap-

proaches so different in nature that one can hardly speak of a coherent state of the art.

Since work on object-oriented modelling is certainly not restricted to the official proposals to the OMG, we will also have a look at alternative approaches, and discuss open research questions as well.

4.1 The Proposals to the OMG

Until the deadline at January 17th the following six proposals had been submitted to the OMG:

Company/Consortium	References	Extent (pages)
Taskon; Reich Technologies; Humans and Technology	[Tas97]	101
IBM; ObjecTime Limited	[IBM97]	196
Softeam	[Sof97]	37
Platinum Technology	[Pla97]	318
Ptech	[CeIb97]	58
Rational Software; Microsoft; Hewlett-Packard; Oracle; Texas Instruments; MCI Systemhouse; Unisys; ICON Computing; IntelliCorp.	[Rat97a] ... [Rat97j]	>550

Taskon; Reich Technologies; Humans and Technology

This submission does not claim to describe a complete set of object-oriented modelling languages. Instead its authors intend to provide an "extension of the UML, and the OML object models" ([Tas97], p. 1). It originates from a number of previous approaches (like [Ree95], [WiWi90]). Its main emphasis is on three additional basic concepts: *role model*, *role*, and *port*. Roles emphasize another level of abstraction than classes. A role represents a particular real world object's role within a certain activity. A role model describes the interactions of roles within an activity. Those interactions are invocations of operations in other roles/objects. Apparently role models are to be used mainly during analysis, in order to record systematically the features of classes within a corresponding object model: "Knowing all the roles that are to be played by an object, the interfaces of its class can be created as the union of all the relevant role model interaction interfaces." ([Tas97], p. 15) A port is described as "an abstraction on a variable, permitting the object represented by the adjoining role to execute operations in the object represented by OtherRole." ([Tas97], p. 50) This concept is not sufficiently explained. We assume that it is similar to - although not equivalent - to delegation, where an object may transparently access operations of its role object ([FrHa97]).

Figure 4 gives an impression of the core elements of the metamodel proposed by this consortium. The only comment of the authors is that it renders "a rough overview of the dependencies" (of model elements). Without offering directed arcs, the diagram is hard to understand. This lack of precision and formalization is characteristic for the whole submission. The metamodel itself is defined using the IDL standardized by the OMG. Although this has been encouraged by the OMG, there is no doubt that IDL is not suited to serve as a modelling or specification language, since it lacks relevant concepts (for instance: it is not possible to express cardinalities or more specific constraints).

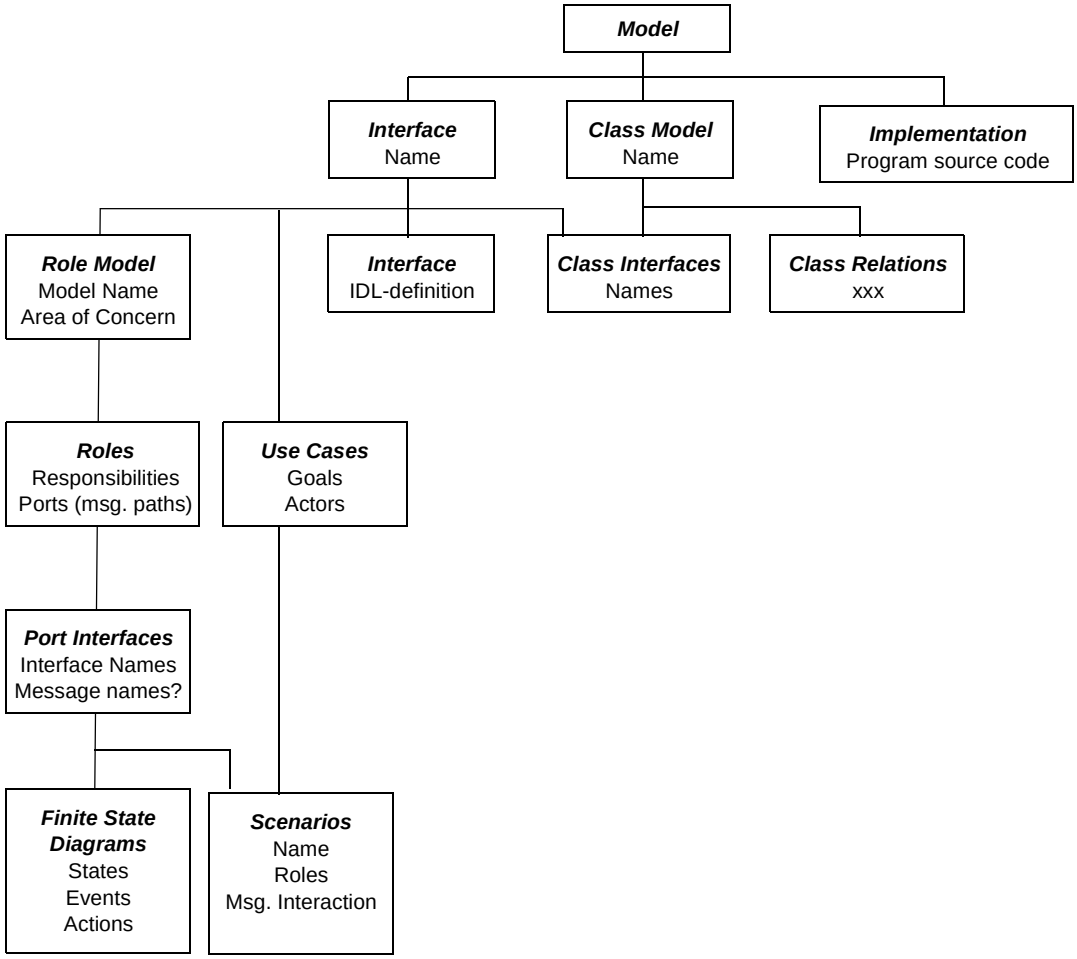

Fig. 4: "Dependency Hierarchy" of model elements ([Tas97], p. 17)

There is one other suggestion within this proposal that we would like to mention: "system inheritance" is to provide for specializing from "the overall system structure and behavior properties" ([Tas97], p. 14). As the authors state, such a concept would certainly be helpful for designing reusable frameworks. That, however, would require a precise definition of this type of inheritance - especially how to modify and enhance an inherited system. We could not find such a definition within the proposal.

Although Reich claims to be a tool vendor, it seems that this consortium is focusing on the applications of modelling languages rather than on metamodels which would promote tool interoperability. The authors have rather elaborated ideas about the particular models to be used within a software development project. They suggest not less than nine partial views together with corresponding modelling notations. The models are described in an informal way ([Tas97], pp. 88):

"1. *Area of Concern view*, which is a textual description of the phenomenon modeled by the role model. (The Area Of Concern is recorded as the explanation in the RoleModel element).

2. *Stimulus-Response view*. A tabular view showing the system's environment objects (ac-

tors), the trigger messages that start system activities, and messages conveying results to the environment roles.

3. *Role List view*, a text view showing a list of all roles with their explanations and attributes. (We do not use a tabular form, because we want to encourage comprehensive explanations).
4. *Interface view*, a formal text view showing interface definitions in the user's preferred language. IDL is taken as default.
5. *Collaboration view*, a graphical view showing the pattern of roles and the message paths between them.
6. *Scenario view*, a graphical view showing example time sequences of messages flowing between the roles.
7. *State Diagram view*. A graphical view. There may be one state diagram for each role. It describes the possible states of the role, the signals that are acceptable in each state, the action taken as a result of each signal, and the next state attained after the action is completed. The only kind of signal possible in our model is the receipt of a message.
8. *Synthesis view*. A graphical view showing the system inheritance relations between role models.
9. *Synthesis table*. A tabular view showing the mapping of base model roles to derived model roles. Mapping of the corresponding ports is implicit, all base model ports and interfaces being mapped onto the derived model"

All in all, we have the impression that the extensions suggested in this submission may be helpful, however, they do not describe essentially new features. Furthermore these extensions are not defined in a precise way. When it comes to the application of modelling languages, this submission is rather interesting, since it includes substantial ideas of how to model a domain/ system from various perspectives.

IBM; ObjecTime Limited

This proposal, submitted by IBM and ObjecTime covers what may be called the "full range" of common object-oriented modelling: static properties, behaviour, and dynamics. Nevertheless the authors apparently do not only intend to define a complete set of languages for object-oriented modelling. While they completely abstract from modelling notations, they put strong emphasis on language semantics and pragmatics. It is characteristic for this outstanding submission that it first describes the requirements related to modelling languages in general. The authors start from the assumption that models may serve a wide range of purposes, which can hardly be determined in advance. To illustrate this point, they distinguish between "general purpose modelling languages", "domain specific modelling languages", and even "application specific modelling languages" (see fig. 5). Furthermore they state that modelling languages should take into account concepts which are appropriate for a particular purpose (and particular people) rather than reconstructing programming languages. On the other hand they argue that modelling applied to software development at some point requires formal definitions of the modelling languages. From these assumptions they conclude that there is "need for extensibility", "for language and method independence", and "for formality and interoperability" ([IBM97], pp. 12).

The consortium presents a "Core Meta-Model (CMM)" which is the foundation for the models that can be built within this approach. The CMM contains generic concepts which allow to express static, dynamic, and interaction semantics. It is characteristic for the level of abstraction

suggested by this proposal that the CMM does not explicitly contain core concepts of object-oriented modelling such as classes or objects. Instead the CMM provides abstractions which may be specialized further. One of the core abstractions is called "specification" (see fig. 5). Specification is an abstraction of concepts like type and class: "Specifications representing pure interfaces will have all Features unimplemented, those representing concrete classes will have all Features implemented, and further possibilities exist in between." ([IBM97], p. 37)

To allow for domain, or application specific modelling languages, they suggest "model schemes" which are to foster "modeling languages as formal extensions to the Core Meta-Model" ([IBM97], p. 11). Compared to other submissions, there is a stronger emphasis on formalization. For this purpose the authors introduce and define a formal language called "Object Constraint Language" (OCL) that is used to specify the CMM. Furthermore it helps with the specification of additional modelling languages, since the OCL in general is used to specify invariants within model schemes. Additionally, each "model scheme may define a Grammar for each type of Expression" ([IBM97], p. 72) - like invariants or conditions. The proposal includes a few examples of how to specify a model scheme. Smalltalk "design models", for instance, can be introduced by a scheme which includes - among other things - invariants to constrain generalization to single inheritance, or to express that every element within a class has exactly one name ([IBM97], p. 117):

SmalltalkClass
applies to Specification
invariant

- self.elementNames.collect(namedElement).forAll(el |
  el.names.select(nameSpace=self).size = 1);
  -- every element has just one name in self's namespace

- self.refinees.size <= 1;
  -- single inheritance

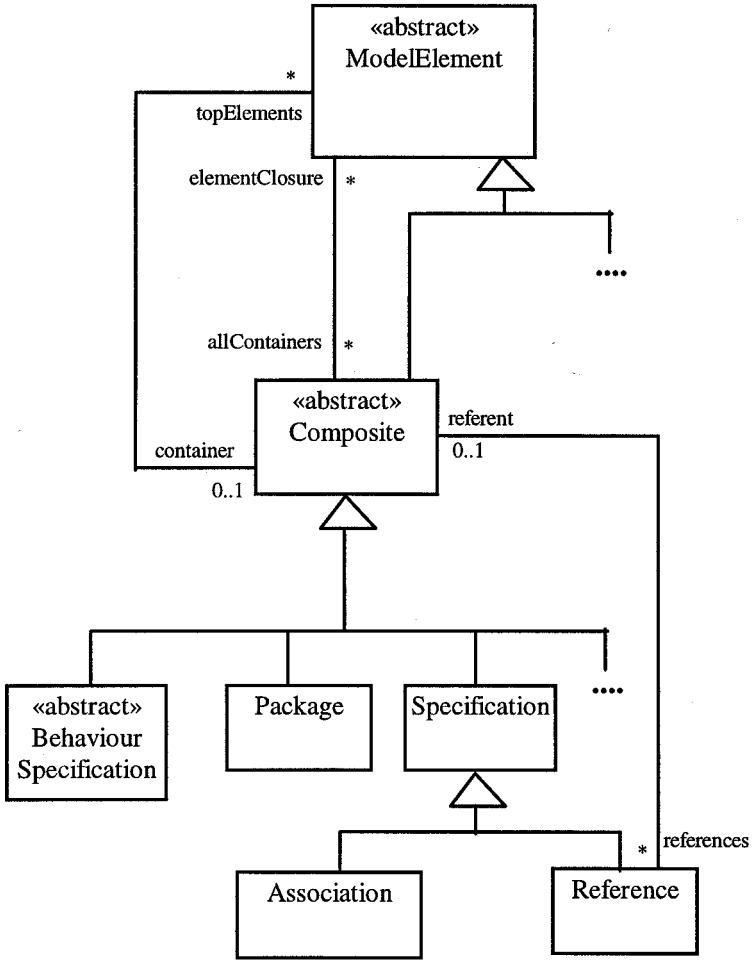

Fig. 5: Concepts within the Metamodel ([IBM97], p. 23)

The ability to introduce specialized languages which are defined by concepts in more general languages is a promising approach - last but not least to support the idea of standardized business object models (which correspond to the "domain specific modelling languages" in fig. 6). Notice, however, that extensibility - like the definition of new modelling languages with existing meta concepts - is not sufficient to allow for a straightforward interchange of models expressed in such a new language: Standardized meta concepts do not imply that the special concepts of a particular modelling language will be standardized, too. For instance: If you define a language scheme for a business modelling language, you may want to introduce a concept like "Organisational Unit". It is probably very hard to define such a concept in a way that everybody is willing to accept. Nevertheless a standardized metalanguage allows to determine precisely how a specialized concept was defined - even if you do not agree with it.

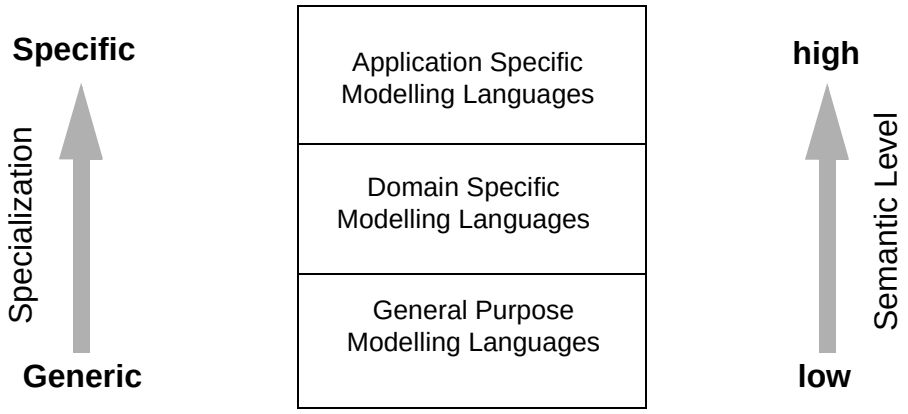

Fig. 6: Different levels of modelling languages ([IBM97], p. 14)

Although the authors point to the fact that modelling languages should suit a particular purpose, providing particular modelling languages is not their main concern. Instead they refer to approaches such as the UML or the OML ([IBM97], pp. 131) which they consider as possible applications of their specification. From our point of view, this is a very ambitious, yet substantial approach. To briefly summarize our impression, we think that the strength of this approach marks a possible weakness at the same time. Taking into account the realistic assumptions on requirements for modelling languages, it is certainly a good idea to provide for a high degree of extensibility: Thereby developers (and users) of specific modelling languages will not be restricted to concepts which do not fit their needs. However, by allowing for almost arbitrary specific modelling languages, standardization does not happen on the level of those languages.

Softeam

This submission is based on the "Class Relation method" which was introduced in 1989, and revised later ([Des94]). Softeam's proposal is certainly not as complete as for instance the previous one. In fact, it seems that the authors primarily intend to introduce a few concepts as alternatives to corresponding concepts in the UML. In particular they suggest an "object flow model" that is to provide a representation of operations behaviour. Most remarkable, from our point of view, is the introduction of a special state diagram that is based on the "Hygraph" principle suggested by Harel. It is motivated by the fact that the state charts often used in object-oriented modelling neither allow for a well defined integration with object models, nor do they allow to express generalization/specialization hierarchies. The suggested state diagrams can be decomposed into two different categories ([Sof97], pp. 7). "Control state diagrams" focus on the usage of a class, describing what is to be expected from an instance of this class from an external point of view. The authors state - referring to [Des94] - that control state diagrams can be completely translated into pre- and postconditions. "Trigger state diagrams" represent a particular implementation of a control state diagram. In other words: They describe the actual dynamics a class has to implement in order to satisfy the requirements specified in a corresponding control state diagram. Figure 7 gives an overview of the control state diagram metamodel.

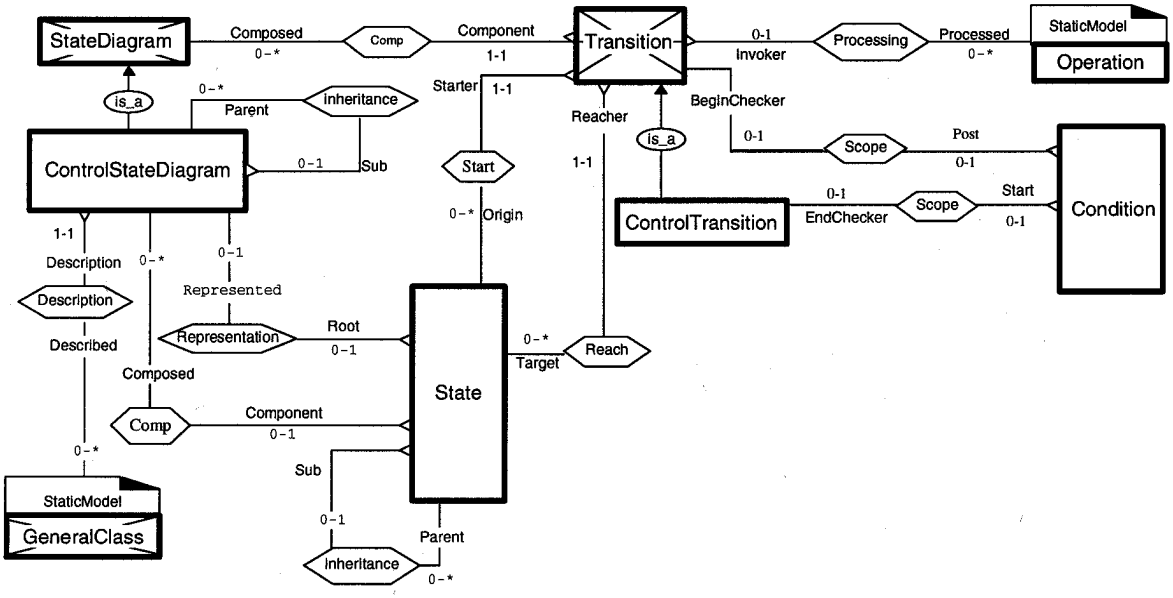

Fig. 7: Control state diagram metamodel ([Sof97], p. 20)

The specialized state charts suggested by Softeam promise both a better integration with other partial models, and to take advantage of generalization/specialisation. Notice, however, that the corresponding concepts are not defined in the proposal itself - which comprises only about 30 pages. Instead it is referred to [Des94].

Platinum Technology

Platinum used to be a provider of tools for the development and maintenance of relational DBMS. With the acquisition of Protosoft, the developer of an object-oriented modelling tool called "Paradigm Plus", Platinum became one of the major vendors in this market. "Paradigm Plus" is claimed to support various popular modelling approaches, such as OMT ([Rum91]), OOSE ([JaCh92]), and Booch ([Boo94)]. Against this background it is not surprising that Platinum's submission puts strong emphasis on tool interoperability. The proposal includes seven modelling languages, each of which is assigned to a so called "subject area". Each subject area represents a certain view on a system - such as object models, dynamic models, architecture model, etc. The subject areas are defined by seven corresponding metamodels which are tightly coupled. The concepts shared by all seven metamodels are assigned to an additional subject area called "Foundation and Common Subject Areas", which in turn is defined in a corresponding metamodel.

Notice that the COMMA ("Common Object-oriented Methodology Metamodel Architecture") metamodel, proposed by the OPEN consortium ([FiHe96), "had a major influence" ([Pla97], p. 13) on those metamodels. Furthermore the metamodels are defined within a common "meta meta model". It is based on the CDIF meta metamodel ([EIA93], in other words: it uses the concepts defined in this reference model). Therefore the metamodels can be exchanged between all tools which are capable of using the CDIF interchange standard formats.

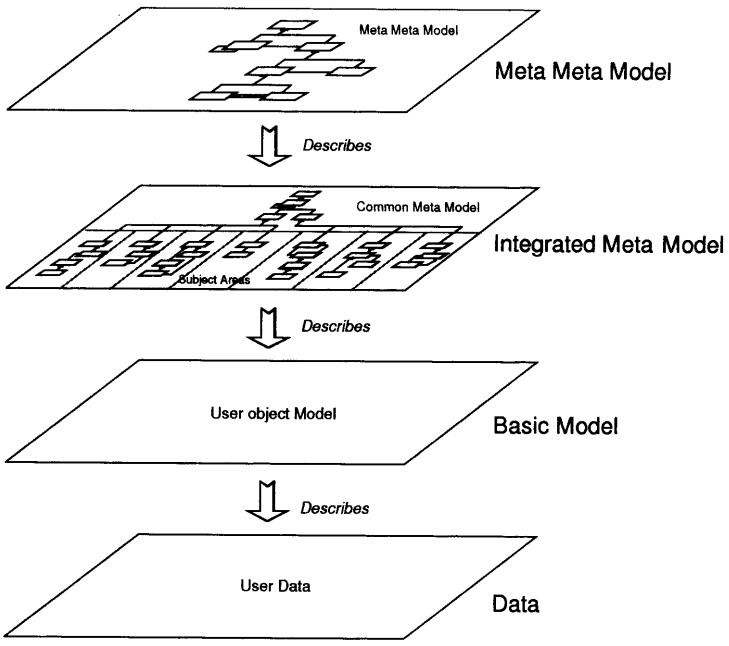

Fig. 8: Different levels of models ([Pla97], p. 14)

The meta metamodel (for an overview see fig. 8) provides a few core concepts which are "totally immutable" ([Pla97], p. 16). "Extensibility" of subject areas can be accomplished by modifying the corresponding metamodels. Notice, however, that extensibility is restricted to the expressive power of the concepts defined in the meta metamodel. Since those concepts are essentially restricted to data structures (see fig. 10) it seems that there is no way to specify constraints that go beyond the description provided by such structures - as it is possible in the metamodel suggested by IBM and ObjecTeam.

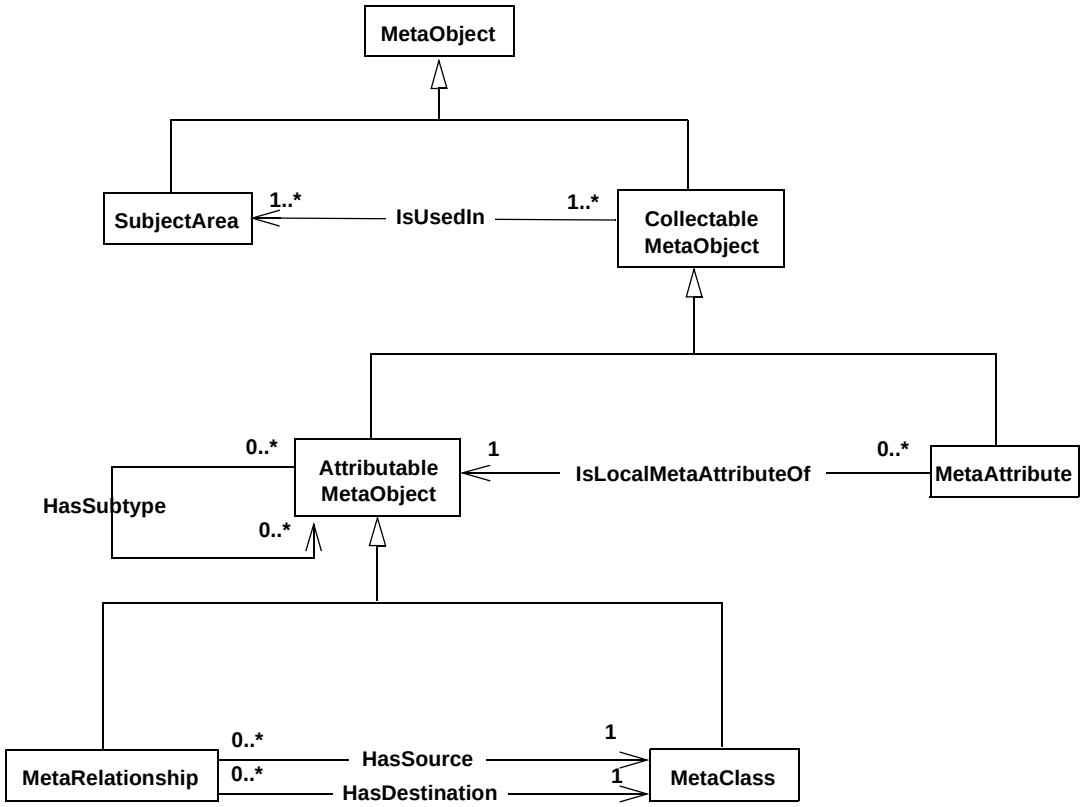

Fig. 9: Concepts of the meta metamodel ([Pla97], p. 23)

The submission includes an extensive description of the meta metamodel as well as of the metamodels (about 2/3 of the whole document), mainly on a level of detail illustrated in figure 9. By referring to an existing standard for exchanging information between CASE tools (CDIF), it is probably of special value for tool providers. However, from our point of view the proposal neglects to consider the requirements of conceptual modelling: A modelling language is not only thought to define models for the purpose of exchanging them between programs. Instead it is also a tool to support to an intellectual endeavour. In other words: It includes pragmatic aspects as well. It seems that these aspects are not Platinum's primary concern - on the contrary: In the past Platinum has gained its reputation as a tool vendor by supporting many modelling languages. In order to hold on to this tradition the proposal is restricted to metamodels which can be used to define a set of particular modelling languages. To emphasize this point of view, Platinum does not provide a specific notation. Instead it is argued that the proposed specifications "are fully compatible with the current state of the art in object modeling notations" - a statement, which is illustrated by brief examples that refer to other approaches which put more emphasis on notation ([Pla97], pp. 305). Although the "current state of the art" is hard to identify, such a statement is rather daring: Notations only make sense with associated concepts. We doubt that the meta metamodel allows to express the semantics of any concept that may be useful within object-oriented modelling. For instance: How would you express the specific semantics of delegation (see [FrHa97]).

<Name> - the name of the entity being described

ISINSTANCEOF

<entity which this entity is an instance of>

META-(X)-ATTRIBUTE-VALUES

<list of values for the attributes derived from the template from which this entity was instantiated>

META-(X)-RELATIONSHIP-INSTANCES

<list of relationship instances for the relationships derived from the template from which this entity was instantiated>

(X)-ATTRIBUTES

INHERITED:

<list of attribute templates which this object inherited from its supertype>

LOCAL:

<list of local attribute templates which this object introduced directly>

(X)-RELATIONSHIPS

INHERITED:

<list of relationship templates which this object inherited from supertypes>

IOCAL:

<list of local relationship templates which this object introduced directly>

"Basic Entity Schema"

Fig. 10: Template for Specifying a "Basic Entity Shema" ([Pla97], p. 19)

Ptech

The authors of this proposal first make a few assumptions on the purpose of modelling languages. They emphasize both, the need to offer a medium to foster effective communication between humans with different professional backgrounds, and the need for tool interoperability ([CeIb97], pp. 1). They provide metamodels for five types of models: "structural models", "architectural models", "behavioral models", "distributed processing models", and "usage models". It is remarkable that the authors seem to be very much inspired by relational theory. In order to define the concepts, they suggest for object-oriented modelling, they often refer to notions such as "set", "domain", "range", or "relation" - for instance: "The associations that implement the relation are named customer and order." ([CeIb97], pp. 3). The concepts rendered in fig. 11 give evidence for this assumption as well.

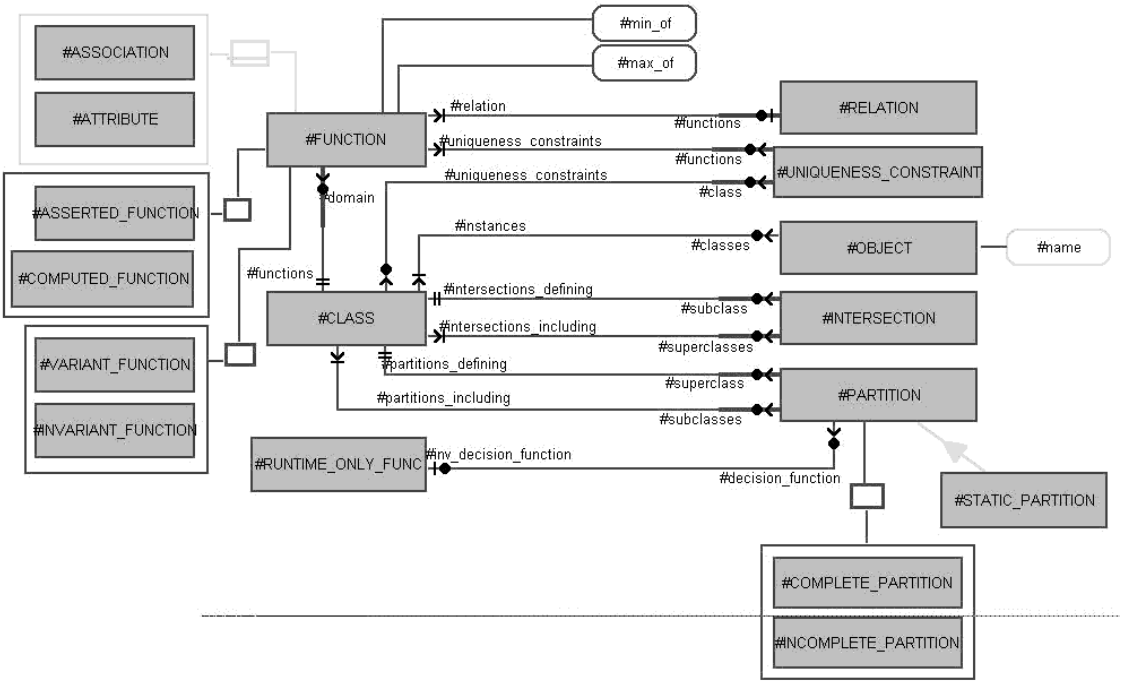

Fig. 11: Ptech Object Metamodel ([CeIb97], p. 6)

To some degree this proposal benefits from the precision of relational theory. It lacks, however, a specific object-oriented point of view - in other words: The authors seem not to have overcome the separation of function and data. For instance, they speak of "... an operation that can be performed on instances of a class." ([CeIb97], p. 14) Although the authors state that their metamodels "guarantee that all Ptech models are logically consistent." ([CeIb97], p. 1), the documentation of the metamodels does not seem to be complete. While the metamodels are said to be extensible ([CeIb97], p. 1), it is not demonstrated how an extension can be accomplished. Furthermore there is no language included that would allow to specify additional constraints. From our point of view, the most remarkable aspect of this proposal is the background of the Ptech method: It has been used for "understanding, analyzing, capturing, validating, and documenting business processes and systems." ([CeIb97], p. 1) It is surprising that the authors did not add specific requirements for object-oriented modelling languages to be applied in the area of process modelling.

Rational et al.

This submission is backed by a rather impressive consortium including Rational Software, Microsoft, Hewlett-Packard, Oracle, Texas Instruments, and Unisys. The proposed "Unified Modelling Language" (UML) has mainly been developed by Rational Software and has its roots in three well known object-oriented modelling approaches ([Boo94], [Jac92], [Rum91]). The material proposed by the consortium is relatively extensive: It includes 13 documents ([Rat97a] .. [Rat97j], the remaining three PDF documents provided by the OMG were apparently damaged) which cover more than 550 pages. Furthermore the background of the UML is explained in various textbooks ([Boo94], [Jac94], [Jac92], [Rum91], [Der95], [Whi94]), and

in numerous articles. But it is certainly not the sheer volume, why the UML is already regarded as a reference - often without discussing its qualities. Even within the competing proposals to the OMG the UML seems to have gained an outstanding position. This may be due to the fact that it is associated with three protagonists of the object-oriented modelling arena. Furthermore it seems that most of the other submitters have accepted the relevance of this proposal, since they describe how their metalanguages, or metamodels respectively would allow to specify the UML modelling languages (for instance [IBM97], [CeIb97]) - or they restrict their efforts to providing extensions to the UML (like [Sof97]).

Taking into account the obvious relevance of the UML as well as the extent of the available documents, we decided to analyze it in more detail in another report ([FrPr97]). Compared to the previous approaches, the UML is based on, the language description submitted to the OMG is much more comprehensive (fig. 12 gives an impression). Although some concepts of the original approaches, such as data flow diagrams, which are part of OMT, are not supported anymore, the UML almost appears like a superset of the languages it originates from. It supports nine types of diagrams. Static aspects can be rendered on various levels of abstraction. In addition to *class* or *object diagrams*, *component diagrams* serve to map existing software components and their interrelationships. *Deployment diagrams* can be used to render dependencies between runtime systems across various platforms. Collaboration diagrams allow to express message flows between objects. *Sequence diagrams*, *state diagrams*, and *activity diagrams* serve to render various dynamic aspects. Furthermore the UML provides *use case diagrams* as suggested in [Jac92]. While the variety provided with those diagrams improves the chances to find a diagram type which is appropriate for a particular view, it can be a burden at the same time, since some of the diagrams (like sequence diagrams, state diagrams, and activity diagrams) are overlapping in a subtle way.

The UML is defined by an object-oriented metamodel which features the same notation as the UML itself. The metamodel is supplemented by natural language descriptions. In order to allow for language extensions, the UML includes, among other things, a concept called "stereotype". A stereotype can be assigned to a set of modelling elements very much like a predicate. Notice, however, that the UML does not offer means to formalize the semantics of a stereotype. Hence, the semantics of a stereotype depends on human interpretation or on specialized tools. From our point of view the differentiation between *type*, *class*, and *interface* is not convincing: "Class is a subtype of Type, and therefore instances of Class have the same property as instances of Type. The fundamental difference being that Type instances specify interfaces, whereas Class instances specify the realization of these interfaces." ([Rat97a], p. 57) Reducing a type to an interface means to widely abstract from its semantics - which does not correspond to common ideas about types.

Without any doubt, the UML marks a clear progress compared to its direct predecessors. Nevertheless it is not completely convincing in the end. Although it comes with a semi-formal metamodel, some of its concepts still lack sufficiently precise definitions - use cases are one example. Furthermore, the UML lacks possibilities to specify additional constraints in a formal way. Different from IBM/ObjecTeam or Platinum, Rational et al. focus on the use of the language. However, it seems questionable whether the extent and variety of concepts provided with the UML will not confuse the prospective users. It is by no means easy to learn and use. For a more detailed evaluation see [FrPr97].

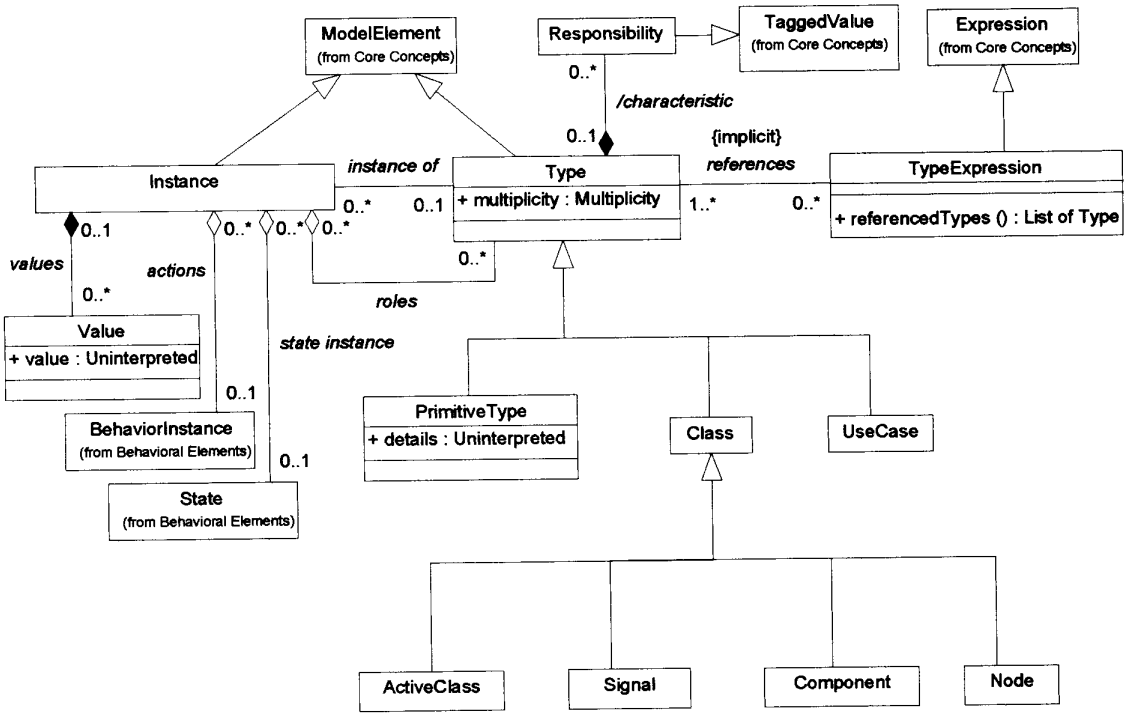

Fig. 12: Basis modelling concepts within UML ([Rat97a], p. 24)

4.2 Additional Approaches

Beside the official submissions to the OMG there are numerous other approaches that contribute to the state of the art as well - from various perspectives (for instance [Mey97], [Coa95], [GoRu95]). From our point of view, one of them is of outstanding importance since it is a very recent effort which is backed by a number of researchers, most of them with a remarkable reputation in the field of object-oriented software development. It has evolved from a joint research effort called OPEN ("Object-oriented Process, Environment, and Notation"), a part of which is the specification of the OPEN Modelling Language (OML, see fig. 13). Different from the companies that submitted their proposals to the OMG, OPEN is an initiative launched solely within an academic context. OPEN has been developed further from a number of previous approaches, like [Col89], [Des94], [Fir92], [Gra91], [HeEd94], [Hen92]. As it is indicated by its name, the members of the initiative clearly want the OPEN specifications to become standards ([FiHe96], p. 4). Nevertheless, they did not submit the already existing specifications to the OMG. A submitting organisation has to be a "contributing member" of the OMG. Furthermore it has to provide a statement about its willingness to make the proposed "technology ... commercially available within 12 months of adoption" ([OMG96a], p. 24) - whatever that means for modelling languages.

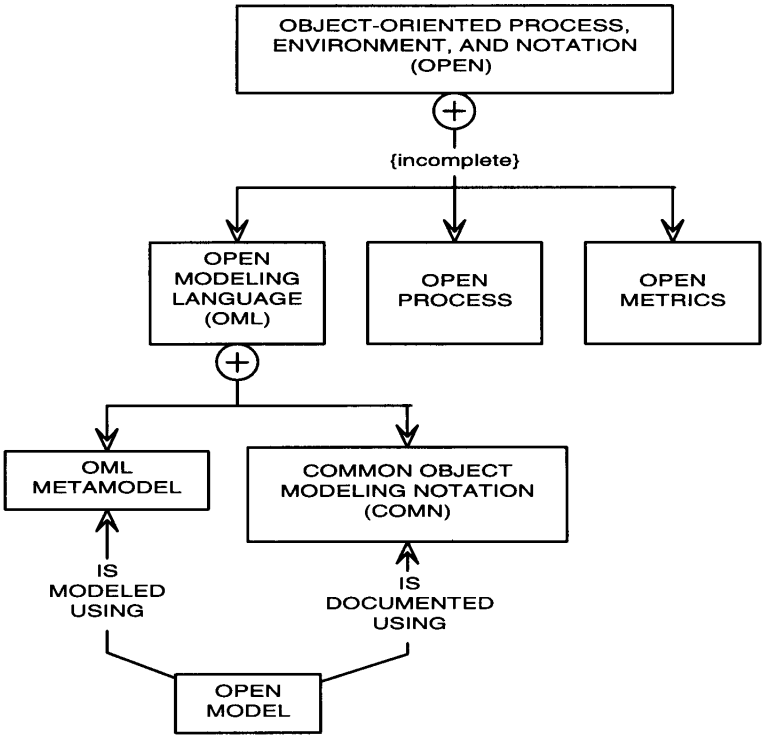

Fig. 13: Partial OML metamodel for objects and their classes and types ([FiHe96], p. 5)

The OML allows for 15 different diagrams ([FiHe96], pp. 131), including most of those offered by the UML. The diagram types are divided into four categories: "Semantic nets", which include *inheritance diagrams* and *cluster diagrams*, "scenario class diagrams", which are used to render various views on scenarios, such as use cases. "Interaction diagrams" are another category. They include *collaboration diagrams* that serve to render the message flow between objects. The fourth category consists of "state transition diagrams". Similar to the UML, the OML features stereotypes as a mechanism to extend the language. "Multi object" is a concept that serves to render "... any homogeneous collection of objects that are instances of the same class or type" ([FiHe96], pp. 226). While class diagrams usually do not require such a concept, since one may use cardinalities for the same purpose, there are other abstractions - like interaction diagrams - where it may be helpful to explicitly render a collection. Similar to other approaches, the OML provides a concept called "cluster" that serves to group a set of objects, classes, or parts of non-object-oriented software into cohesive collections. Clusters may include clusters. A cluster may have a class and a type. Furthermore a cluster class may be specialized from another cluster class ([FiHe96], pp. 47). Unfortunately the authors neglect to specify precisely what they mean by "cluster inheritance".

While the notation of a modelling language in general is hard to judge, the OPEN notation offers one feature that may foster a more convenient use of the language for many people: Beside the full range notation, there is a "light" version that includes a fairly small set of the most relevant concepts ([FiHe96], pp. 199). "Drop-down" boxes are another concept which allows to adapt the level of detail to be rendered. Each modelling element can be associated with a drop-down box that serves to display a selection of the elements' traits (specific information about

the modelling element, like an attribute of a class). While the OML includes some interesting features and offers a number of advantages over the UML, we can hardly agree with the authors who suggest that their approach offers the "best available notation" and defines the state of the art of object-oriented modelling languages ([FiHe96], p. 200). Like the UML, the OML does not allow for a formal specification of user defined constraints. Furthermore some of the concepts suggested with the OML are not sufficiently formalized. Comparing the OML against the UML reveals that both have their specific strengths and shortcomings (see [FrPr97]). Hence, one may conclude that none of them defines the state of the art on its own.

4.3 Open Research Questions

Assuming that modelling languages are of pivotal importance for the way people perceive and conceptualize real world domains, and how they design software, the design of a general modelling language is a challenging task. This is for various reasons. Firstly, there is *variety* - both in domains and in software systems. For instance, there are certainly essential differences between traffic control systems, vending machines, or marketing information systems. This is the case for the corresponding terminologies as well as for abstractions used to render system features. Additionally, there is an immense variety of people who use a modelling language - both as readers and writers. Since it can be expected that this variety is accompanied by a wide range of individual perceptions, conceptualizations, and preferences, it is almost impossible to design a language that fits the needs of all potential users. Secondly there are *trade-offs*. A modelling language should be easy to use. Its notation should be intuitive, which implies that it corresponds to the conceptualizations its users prefer. At the same time it should support the design and implementation of software. At some point, that requires to introduce formal concepts which are suited to be mapped to implementation languages. Furthermore there is the well known trade-off between quality, cost, and time. While it is difficult to tell how a modelling language relates to this trade-off, we cannot assume that quality, cost, and time are independent of the modelling languages used within a project. Thirdly there is *arbitrariness*. Like any artificial language, a modelling language should be designed to fit its purpose. However, it cannot be deduced logically from this purpose. This thought alone implies that arbitrary decisions can hardly be avoided. Furthermore, dealing with trade-offs will usually require compromises that will, to a certain degree, reflect subjective preferences rather than substantial reasons.

Comparing the current state of the art of object-oriented modelling against those challenges reveals a remarkable lack of research. Regarding a modelling language as an instrument for software development would suggest to start with a thorough requirements analysis, which includes a number of questions, for instance:

- What are the purposes the language is to be used for?
- Who are the prospective users of the modelling language?
- What are the concepts a language should provide in order to fit their cognitive styles?
- How does an intuitive notation look like?

There is no doubt that answering these questions requires some sort of empirical research. The authors of object-oriented modelling methods or languages sometimes emphasize the experience they have gathered with applying their approach in practice (for instance [FiHe96], p. 9, or [CeIb97], p. 1). Vendors and consultants have certainly received some kind of feedback from their customers. Additionally, there are forums on the internet dedicated to object-orient-

ed modelling (like comp.obj). However, it is not too daring to state that none of those activities has resulted in a representative and detailed analysis of user requirements. It is remarkable that up to now there has been no sophisticated study of how people perceive and deal with concepts and notations of object-oriented modelling languages (at least we do not now of any). There have been a few studies on the use of entity relationship (ER) modelling ([Hit95], [GoSt90]). They indicate that ER models are not intuitive at all for many people. Our experience, as well as the similarity between object models and ER models suggest that this is also the case for object-oriented modelling. Approaches like the UML or the OML, which offer a wide range of different modelling languages, can be expected to be even less intuitive for many prospective users. Furthermore some concepts featured by those languages, such as use cases, are difficult to understand. For this reason, applying them can easily result in bad design (for a detailed analysis of pitfalls see [Ber97]). In other words: Without substantial knowledge about the way how people perceive and apply modelling concepts, it is hard to tell whether those concepts contribute to software quality - or to "disaster" ([Ber97]).

There are modelling approaches and studies which put more emphasis on user involvement, like [Che81], [Mum95], [DaLe91]. They, however, originate in the social sciences and lack concepts to support later stages of the software development process. It is not surprising that those approaches have evolved within a scientific community that apparently has no links to the field of object-oriented modelling.

Beside the lack of research on user perceptions and preferences, there is a wide range of domains and purposes that suggest the use of modelling languages. To give a few examples: business process (re-) design, workflow management, organisational design, enterprise modelling, design of document management systems, design of integrated circuits, design in the field of computer telephony integration (CTI). During the last years a lot of special purpose modelling approaches have emerged in those fields (for instance: [Fra97], [Fra94], [IsSt95], [Oul95], [Sch95], [Tay95]). While they usually require specialized concepts, many of them are closely interrelated with object-oriented modelling. For instance: Modelling a business process usually requires to refer to information specified within an object model. In order to foster integration, it is helpful to regard documents as objects which require special concepts. None of the modelling languages discussed in 4.1, 4.2 includes special concepts to cover one of the example fields mentioned above. Furthermore some of those fields are still in a virgin state, although they are of increasing relevance - like business process modelling or enterprise modelling. Against the background of standardization, it is important to notice that additional requirements for modelling languages can be expected from various evolving areas.

While reusability has been a research topic for long, there are two recent approaches that have gained remarkable attention: design patterns ([GaHe95]) and frameworks ([LeRo95]). They promise to deliver reusable artifacts which are rather flexible and/or convenient to use. Nevertheless, there is only little experience about how to integrate them in the process of software development. This is especially the case for design patterns since they are - by definition - less formal in nature than frameworks. It seems characteristic that, despite these problems, some of the approaches discussed in 4.1, and 4.2 already feature concepts to describe design patterns (for instance: [Rat97a], [FiHe96]) - however, without specifying them in an adequate way (for instance: [Rat97a] defines a design pattern as "a template collaboration", p. 66).

In order to specify/standardize modelling languages you need meta models/languages. Among other things, they should allow for convenient and safe extensions of the corresponding object

level languages. At the same time a meta language perspective, as emphasized in [IBM97] or [Pla97], allows to abstract widely from some problems occurring on the level of a particular modelling language - like user perceptions or preferences. For this reason it may appear that an approach like the one suggested by [IBM97] provides a satisfactory solution at least for the metalanguage level. However, this is not the case. Within computer science there is a wide range of alternatives. Beside general, well known approaches like predicate calculus, or algebraic specification, there are many special approaches that define representations for metalanguages to be used for CASE tools or Meta CASE tools respectively (for instance: [EbWi96], [KeSm96]). While there are requirements for metalanguages, like completeness, simplicity, and correctness (see [SüEb97], pp. 2), it is still difficult to compare them in an objective sense: Similar to modelling languages to be used on an object level, metalanguages are artificial. Therefore they are necessarily arbitrary to some degree. This is an important aspect for another reason as well: Even with metalanguages you cannot completely neglect cognitive styles of prospective users: An extensible metalanguage/metamodel is also used by people who have their own ideas about language concepts and notation - although the group of people who work on a meta level is much smaller than the group of those who use a modelling language on an object level.

5. Concluding Remarks

Our overview of the current state of the art has shown that object-oriented modelling is a still evolving field with relevant problems and challenges still to be overcome. To summarize, we can state that there is lack of knowledge about how users perceive and apply object-oriented modelling languages. There is no substantial knowledge about how the various views, offered by some approaches, relate to development cost and software quality. Evolving new modelling areas, like business or enterprise modelling, have not been sufficiently taken into account by most object-oriented modelling languages. Furthermore the field of object-oriented modelling lacks a common focus: There is work on metamodels with more or less formal rigour, while other approaches focus on the application of modelling languages. There is also variety in the backgrounds of people working on various aspects of object-oriented modelling; to name a few: programming languages, database design, artificial intelligence. Furthermore the motives range from purely academic to greatly commercial.

Against this background one can hardly speak of a mature state of the art. While this thought would recommend further research in object-oriented modelling, there is yet another, more essential, reason to beware of premature standards: Although object-orientation offers substantial benefits and seems to be the dominating paradigm in software engineering, it is questionable whether or not it is satisfactory in the long run. Firstly, you have to take into account that there are alternative modelling approaches which offer particular benefits (for instance: Petri nets, rule based systems). There is need for thoroughly analyzing how object-oriented languages relate to these approaches, both in terms of semantics, and pragmatics (when to use a particular approach). Secondly, there is no doubt that object-oriented modelling does not cover the whole range of challenges to be dealt with in software development. Beside being an engineering task, the development of software can also be seen as a process that depends on social and psychological aspects: Different, and competing, goals are involved, as well as a wide range of interpretations and preferences. There are a few approaches which emphasize these aspects. They have a strong epistemological background and try to go beyond a mere engineering perspective. The authors refer to constructivism [Floy92] as well as to speech act theory

and hermeneutics ([Win95], [WiFl94], [BuZü90]). They argue that the traditional engineering perspective is not adequate for system design, since system design both depends on and has impact on organisational change. For this reason those approaches focus on natural language and social interactions in order to gain a deep understanding of the current application domain and its potential for change. While some of those approaches remain rather abstract and lack the rigour that is required in software development, they emphasize an important aspect: Software development can hardly be reduced to the use of object-oriented modelling languages.

This thought is raising yet another objection: Like any other language, modelling languages are tools for thinking. They may help with analyzing problems and finding solutions. However, at the same time they may bias perception and conceptualization of those who use them. While we tend to assume that object-oriented modelling allows for "natural" abstractions of the real world, it can hardly be denied that object-orientation has its roots in software engineering. In other words: Since object-oriented concepts are definitely influenced by ideas about system design, it is hardly possible to totally abstract from the system level during object-oriented analysis - not to talk about design. (Notice that there is hope: Maybe both software systems and the "real" world can be described in a "natural" way using the same, or at least corresponding concepts.) For this reason it may be an alternative to concentrate on more "natural" languages which allow to (re-) construct a domain without the bias imposed by an object-oriented modelling language - an approach that is, for instance, suggested by [OrSc96]. However, such an approach does not come without pitfalls either. Firstly, there is good reason to assume that any language will influence the way we perceive and conceptualize reality ([Who59]). Secondly, a modelling language which does not integrate well with concepts needed for system design and implementation will increase the chance of friction.

From an academic point of view software development in general, conceptual modelling in particular has not reached a level of maturity that would recommend to freeze a certain paradigm by standardizing corresponding modelling languages. Although there has been considerable progress during the last years, we still agree with the authors of an open letter to the OMG ([MeSh93]) who, in 1993, emphatically advised against standardization of object-oriented modelling methods (which includes the standardization of modelling languages)[1]:

> "Standardization of this rapidly developing technology will be out of date almost immediately. Not only is standardization futile, but, to the extent that it succeeds, positively dangerous. Standardization will discourage the innovation required to advance and mature the methods."

However, things look different from an economic point of view: There is no doubt that standards are of crucial importance for establishing integrated information systems. Furthermore they foster reusability, help to protect investments, and decrease transaction cost. Hence, in the end it comes down to the trade-off between the benefits of standardized representations and the pitfalls of investing into premature concepts. This decision is beyond the capabilities of IS research: It finally happens in market-like settings - only if enough players see a chance for an equilibrium between the incentives and the cost of standardization, a standard may be established.

1. In the meantime some of them, like Booch, Rumbaugh, or Wirfs-Brock, have apparently changed their mind, since they participated in the preparation of proposals to the OMG.

References

[AcDa91] Ackroyd, M.; Daum, D.: Graphical notation for object-oriented design and programming. In: JOOP, Vol. 3, No. 5, 1991, pp. 18-28

[And96] Anderson, Michael J.: Draft Workflow Standard - Interoperability. Abstract Specification WFMC-TC-1012, 3-June, 1996

[Ala88] Alabisco, B.: Transformation of data flow analysis models to object-oriented design. In: Meyrowitz, N. (Ed.): OOPSLA ’88 - conference proceedings, San Diego, Sept. New York, 1988, pp. 335-353

[ArBo91] Arnold, P.; Bodoff, S.; Coleman, D.; Gilchrist, H.; Hayes, F.: An Evaluation of Five Object-Oriented Development Methods. Report No. HPL-91-52, June 1991 Bristol 1991

[Bai89] Bailin, S.C.: An object-oriented requirements specification method. In: Communications of the ACM, Vol. 32, No. 5, pp. 608-623

[Ben93] Ben-Natan, R., CORBA: A Guide to Common Object Request Broker Architecture. McGraw-Hill, New York et al. 1995

[Ber93] Berard, E.V.: Essays on Object-Oriented Software Engineering. Vol. I, Englewood Cliffs, NJ: Prentice Hall 1993

[Ber97] Berard, E.V.: Be Careful With "Use Cases". 1997. Obtained via http://www.toa.com/pub/html/use_case.html

[Ber86] Berard, E.V.: An Object-Oriented Design Handbook. Rockville, MD 1986

[Big97] Biggs, P.: A Survey of Object-Oriented Methods. Obtained via http://www.dur.ac.uk/~dcs3pjb/survey.html

[Boo94] Booch, G.: Object-oriented Design with Applications. 2nd ed., RedwoodCity, Ca.: Benjamin Cummings 1994

[Bou91] Bourbaki, N.: Toward a definition of object-oriented languages, part 1. In: Journal of Object-Oriented Programming, March/April 1991, pp. 62-65

[BrDr93] Bröhl, A.-P.; Dröschel, W. (Eds.): Das V-Modell: Der Standard für die Softwareentwicklung mit Praxisleitfaden. München, Wien: Oldenbourg 1993

[BuZü90] Budde, R.; Züllighoven, H.: Software-Werkzeuge in einer Programmierwerkstatt - Ansätze eines hermeneutisch fundierten Werkzeug- und Maschinenbegriffs. München: Oldenbourg 1990

[Bu84] Buhr, R.: System Design with Ada. Englewood Cliffs, NJ: Prentice Hall 1984

[CDI96] CDIF: Harmonization of CDIF with other Standards Bodies, 96-07-26, 1996. Obtained via http://www.cdif.org/intro.html

[CeIb97] Cerrato, J.; Ibrahim, H.: The Ptech Method for Object-Oriented Development Version 1.0, 1997. Obtained via http://www.omg.org/library/schedule/AD_RFP1.html

[Che87] Cherry, G.: PAMELA 2: An Ada-Based Object-Oriented Design Method. Reston, Va., 1987

[Che81] Checkland, P.: Systems thinking, systems practice. Chichester et al.: Wiley 198

[Coa95] Coad, P.: Object Models: Strategies, Patterns, and Applications. Englewood Cliffs, NJ: Prentice Hall 1995

[Col89] Colbert, E.: The Object-Oriented Software Development Method: A Practical Approach to Object-Oriented Development. In: Proceedings of TRI-Ada ’89 - Ada Technology in Context: Application, Development, and Deployment. New York: ACM Press 1989, pp. 400-415

[Col94] Coleman, D. et al.: Object-Oriented Development. The Fusion Method. Englewood Cliffs, NJ: Prentice-Hall 1994

[CoYo91] Coad, P.; Yourdon, E.: Object-Oriented Design. Englewood Cliffs, NJ: Prentice Hall 1991

[CoYo90] Coad, P.; Yourdon, E.: Object-oriented analysis. Englewood Cliffs, NJ: Prentice Hall 1990

[CrRo92] Cribbs, J.; Roe, C.; Moon, S.: An Evaluation of Object-Oriented Analysis and Design Methodologies. New York: SIGS Books 1992

[CuBe86] Cunningham, W.; Beck, K.: A diagram for object-oriented programs. In: Meyrowitz, N. (Ed.): OOPSLA ’86 - conference proceedings. New York 1986, pp. 39-43

[DaLe91] Davies, L.; Ledington, P.: Information in Action. Soft Systems Methodology. Houndmills et al.: Macmillan 1991

[DeFa92] De Champeaux, D.; Faure, P.: A comparative study of object-oriented analysis methods. In: JOOP, No. 1, Vol. 5, 1992, pp. 21-33

[Der95] Derr, K.W.: Applying OMT: A practical step-by-step guide to using the object modeling technique. New York: SIGS Books 1995

[Des94] Desfray, P.: Object Engineering - The Fourth Dimension. Reading, Mass.: Addison-Wesley 1994

[EbWi96] Ebert, J.; Winter, A.; Dahm, P.; Franzke, A.; Süttenbach, R.: Graph Based Modeling and Implementation with EER/GRAL. Thalheim, B. (Ed.): Proceedings of the 15th International Conference on Conceptual Modeling. Berlin et al.: Springer 1996, pp. 163-178

[EbSü96] Ebert, J.; Süttenbach, R.; Uhe, I.: Meta-CASE in Practice: A Case for KOGGE. Informatik Fachberichte No. 22, Universität Koblenz-Landau 1996

[Edw89] Edwards, J.: Lessons learned in more than ten years of practical application of the Object-Oriented Paradigm. London 1989

[EIA93] CDIF Framework for Modeling and Extensibility, EIA/IS-107, Electronic Industries Association, November 1993

[EmKu92] Embley, D.W.; Kurtz, B.D.; Woodfield, S.N.: Object-Oriented Systems Analysis. A Model-Driven Aproach. Englewood-Cliffs, NJ: Prentice Hall 1992

[Ern97] Ernst, J.: Introduction to CDIF. 1997, obtained via http://www.cdif.org/

[ESA89] European Space Agency (ESA): HOOD Reference Manual. Issue 3.0. Nordwijk 1989

[Fel87] Felsinger, R.: Object-Oriented Design, Structured Analysis/Structured Design, and Ada for Real-time Systems. Mt. Pleasant, SC 1987

[FeSi91] Ferstl, O.K.; Sinz, E.J.: Ein Vorgehensmodell zur Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik, Vol. 33., No. 6, 1991, pp. 477-491

[FeSi90] Ferstl, O.K.; Sinz, E.J.: Objektmodellierung betrieblicher Informationssysteme im Semantischen Objektmodell (SOM). In: Wirtschaftsinformatik, Vol. 32., No. 6, pp. 566-581

[FiKe91] Fichman, R.G.; Kemerer, C.F.: Object-Oriented and Conventional Analysis and Development Methodologies: Comparison and Critique Boston, Ma. 1991

[Fir92] Firesmith, D.: Object-oriented requirements analysis and logical design. Chichester 1992

[FiHe96] Firesmith, D.; Henderson-Sellers, B.; Graham, I.; Page-Jones, M.: OPEN Modeling Language (OML). Reference Manual. Version 1.0. 8 December 1996. Obtained via http://www.csse.swin.edu.au/OPEN/comn.html

[Flo92] Floyd, Ch. (Ed.): Software Development and Reality Construction. Berlin et al.: Springer 1992

[FrPr97] Frank, U.; Prasse, M.: A Comparison of the Unified Modelling Language (UML) and the Open Modelling Language (OML). Arbeitsberichte des Instituts für Wirtschaftsinformatk, No. 4, Koblenz 1997

[FrHa97] Frank, U.; Halter, S.: Enhancing Object-Oriented Software Development with Delegation. Arbeitsberichte des Instituts für Wirtschaftsinformatk, No. 2, Koblenz 1997

[Fra97] Frank, U.: Enhancing Object-Oriented Modeling with Concepts to Integrate Electronic Documents. In: Proceedings of the 30th HICSS, vol. VI, ed. by R. H. Sprague, Los Alamitos, Ca.: IEEE Computer Society Press 1997, pp. 127-136

[Fra94] Frank, U.: MEMO: A Tool Supported Methodology for Analyzing and (Re-) Designing Business Information Systems. In: Ege, R.; Singh, M.; Meyer, B. (Ed.): Technology of Object-Oriented Languages ans Systems. Englewood Cliffs: Prentice Hall 1994, pp. 367-380

[Fra93] Frank, U.: A Comparison of two Outstanding Methodologies for Object-Oriented Design. Arbeitspapiere der GMD, No. 779, Sankt Augustin 1993

[GaHe95] Gamma, E.; Helm, R.; Johnson, R.; Vlissides, J.: Design Patterns. Elements of Reusable Object-Oriented Software. Reading/Mass. et al.: Addison-Wesley 1995

[GoRu95] Goldberg, A.; Rubin, K.S.: Succeeding with Objects: Decision Frameworks for Project Management. Reading, Mass.: Addison-Wesley 1995

[GoRu92] Goldberg, A.; Rubin, K.S.: Object Behaviour Analysis. In: Communications of the ACM. Vol. 35, No. 9, 1992, pp. 48-62

[GoSt90] Goldstein, R.C.; Storey, V.C.: Some Findings on the Intuitiveness of Entity Relationship Constructs. In: Lochovsky, F.H. (Ed.): Entity Relationship Approach to Database Design and Query. Amsterdam: Elsevier 1990

[Gra91] Graham, I.: Object-Oriented Methods. Wokingham et al.: Addison-Wesley 1991

[HaWi93] Halladay,S.; Wiebel, M.: Object-Oriented Software Engineering. Englewood Cliffs, NJ: Prentice Hall 1993

[HeEd94] Henderson-Sellers, B.; Edwards, J.M.: Book Two of Object-Oriented Knowledge: The Working Object. Object-Oriented SoftwareEngineering: Methods and Management. Sidney et al.: Prentice Hall 1994

[Hen92] Henderson-Sellers, B.: A Book of Object-Oriented Knowledge: Object-Oriented Analysis, Design and Implementation. A new Approach to Software Engineering. Englewood Cliffs, NJ: Prentice Hall 1992

[HeCo91] Henderson-Sellers, B.; Constantine, L.L.: Object-oriented development and functional decomposition. In: JOOP, Vol. 3, No. 5, 1991, pp. 11-16

[Hew91] Hewlett Packard: An Evaluation of Five Object-Oriented Development Methods. Software Engineering Department, HP Laboritories. Bristol 1991

[Hit95] Hitchman, S.: Practitioner Perceptions on the Use of some Semantic Concepts in the Entity Relationship Model In: European Journal of Information Systems, Vol. 4, 1995, pp. 31-40

[HoGo93] Hong, S.; Goor, G.: A Formal Approach to the Comparison of Object-Oriented Analysis and Design Methodologies. In: Nunamaker, J.F.; Sprague, R.H. (Ed.): Information Systems: Collaboration Technology, Organizational Systems, and Technolgoy. Proceedings of the 26th International Hawaii International Conferenc on System Sciences. Los Alamitos 1993, pp. 689-698

[Hsi92] Hsieh, D.: Survey of object-oriented analysis/design methodologies and future CASE frameworks. Menlo Park, Ca. 1992

[IBM97] IBM; ObjecTime Limited: OMG OA&D RFP Response Version 1.0. 1997. Obtained via http://www.omg.org/library/schedule/AD_RFP1.html

[IBM90] IBM: Object-Oriented Design: A Preliminary Approach. Doc. GG24-3647-00. IBM International Technical Support Center, Raleigh, NC 1990

[ISO90] ISO/IEC1990 IRDS Framework. ISO/IEC-Standard 10027. 1990

[IsSt95] Isakowitz, T., Stohr, E.A., Balasubramanian, P.: RMM: A Methodology for Structured Hypermedia Design. In: Communications of the ACM, Vol. 38, No. 8, 1995, pp. 34-44

[Jac94] Jacobson, I.; Ericsson, M.; Jacobson, A.: The Object Advantage. Business Process Reengineering with Object Technology. Wokingham et al.: Addison-Wesley 1994

[Jac92] Jacobson, I.; Christerson, M; Jonsson, P; Overgaard, G.: Object-Oriented Engineering. A Use Case Driven Approach. Reading, Mass.: Addison-Wesley 1992

[JoFo85] Johnson, R.E.; Foote, B.: Designing Reusable Classes. In: JOOP, Vol. 1, No. 2, 1985, pp. 22-35

[KaCH92] Kain, J.B.; Christopherson, M. et al.: Object Analysis and Design. OMG Document 92-10-01.PDF, draft 7.0, 1992. Obtained via http://www.omg.org/library/public-doclist.html

[Kad86] Kadie, C.: Refinement Through Classes: A Development Methodology for Object-Oriented Languages. Urbana, Il. 1986

[KaSc91] Kappel, G.; Schrefl, M.: Using an Object-Oriented Diagram Technique for the Design of Information Systems. In: Sol, H.G.; Van Hee, K.M. (Ed.): Dynamic Modelling of Information Systems. Amsterdam, New York et al. 1991, pp. 121-164

[KeJa96] Kehoe, R.; Jarvis, A.: ISO 9000-3: A Tool for Software Product and Process Improvement. New York et al.: Springer 1996

[KeSm96] Kelly, S.; Smolander, K.: Evolution and issues in metaCASE. In: Information and Software Technology. Vol. 38 (Special Issue: Method engineering and metamodelling), No. 4, 1996, pp. 261-266

[Kin89] King, R.: My Cat is Object-Oriented. In: Kim, W.; Lochovsky, F.H. (Ed.): Object-Oriented Concepts, Databases, and Applications. New York 1989, pp. 23-30

[LeCa91] Lee, S.; Carver, D.L.: Object-oriented analysis and specification: a knowledge base approach. In: JOOP, Vol. 3, No. 5, 1991, pp. 35-43

[LeRo95] Lewis, T.; Rosenstein, L. et al. (Eds.): Object Oriented Application Frameworks. Greenwich, CT: Manning 1995

[LiGu86] Liskov, B.; Guttag, J.: Abstraction and specification in program development. Cambridge, Mass.: MIT Press 1986

[Lor84] Lorenz, K.: Methode. In: Mittelstraß, J. (Ed.): Enzyklopädie Philosophie und Wissenschaftstheorie. Vol. 2, Mannheim: BI Wissenschaftsverlag 1984, pp. 876-879

[Man87] Mannino, P.: A Presentation and Comparison of Four Information System Development Methodologies. In: Software Engine ering Notes, Vol. 12, No. 2, 1987, pp. 26-27

[MaGe88] Masiero, P.; Germano, F.: JSD as an Object-Oriented Design Method. In: Software Engineering Notes. Vol. 13, No. 3, 1988, pp. 22-23

[MaOd92] Martin, J.; Odell, JJ.: Object-Oriented Analysis and Design. Englewood Cliffs, NJ: Prentice Hall 1992

[MaOd95] Martin, J.; Odell, JJ.: Object-Oriented Methods: A Foundation. Englewood Cliffs, NJ: Prentice Hall 1995

[McSy92] McGregor, J.D.; Sykes, D.A.: Object-Oriented Software Development. Engineering for Reuse. New York 1992

[MeSh97] Mellor, S.J.; Shlaer, S.: Recursive Design. Englewood Cliffs, N.J.: Prentice Hall 1997

[MeSh93] Mellor, S.J.; Shlaer, S.; Booch, G.; Rumbaugh, J.; Salmons, J.; Babitsky, T.; Adams, S.; Wirfs-Brock, R.J.: Premature methods standardization considered

harmful Open Letter to the Industry In: JOOP, Vol. 6, 1993, No. 4, pp. 8-9

[Mey97] Meyer, B.: Object-Oriented Software Construction. 2nd Ed., Englewood Cliffs: Prentice Hall 1997

[Mey88] Meyer, B.: Object-Oriented Software Construction. Englewood Cliffs: Prentice Hall 1988

[Mit84] Mittelstraß, J.: Methodologie. In: Mittelstraß, J. (Ed.): Enzyklopädie Philosophie und Wissenschaftstheorie. Vol. 2, Mannheim: BI Wissenschaftsverlag 1984, p. 887

[MoPu92] Monarchi, D.E.; Puhr, G.: A Research Typology for Object-Oriented Analysis and Design. In: Communications of the ACM, Vol. 35, No. 9, 1992, pp. 35-47

[Mul89] Mullin, M.: Object-Oriented Design with Examplex in C++. Reading, Mass.: Addison-Wesley 1989

[Mum95] Mumford, E.: Effective Systems Design and Requirements Analysis. The ETHICS Approach. Houndmills et al.: Macmillan 1995

[MyLe84] Mylopoulus, J.; Levesque, H.J.: An Overview of Knowledge Representation. In: Brodie,M.L.; Mylopoulos,J.; Schmidt,J. (Ed.): On Conceptual Modelling. Perspectives from Artificial Intelligence, Databases and Programming. Berlin/Heidelberg: Springer 1984, pp. 3-17

[Nie88] Nielsen, K.: An Object-Oriented Design Methodology for Real-Time System in Ada. San Diego, Ca. 1988

[Obj97] Object Agency: A Comparison of Object-Oriented Development Methodologies. Obtained via http://www.toa.com/pub/html/mcr.html

[Ode92] Odell, J.J.: Modelling objects using binary-and-entity-relationship approaches. In: JOOP, June 1992, Vol. 5, No. 3, 1992, pp. 12-18

[OMG96a] Object Management Group: Object Analysis & Design RFP-1, ad/96-05-01, 1996. Obtained via http://www.omg.org/library/public-doclist.html

[OMG96b] Object Management Group: Common Facilities RFP-4: Common Business Objects and Business Object Facility. OMG, TC 13CF/96-01-04, 1996. Obtained via http://www.omg.org/library/public-doclist.html

[OMG95a] Object Management Group: OMG Business Object Survey. Doc. 95-6-4, 1995. Obtained via http://www.omg.org/library/public-doclist.html

[OMG95b] Object Management Group: Common Facilities Architecture Draft 4.0. 1995/95-01-02, 1995. Obtained via http://www.omg.org/library/public-doclist.html

[OMG92] Object Management Group: OMG Architecture Guide 4. The OMG Object Model. Draft July, Framingham, Mass. 1992

[OMG97] Object Management Group: A Discussion of the Object Management Architecture. 1997. Obtained via www.omg.org/library/omaa.html

[OrSc96] Ortner, E; Schienmann, Br.: Normative Language Approach - A Framework for Understanding. In: Thalheim, Bernhard (Ed.): Conceptual Modeling - ER'96 Proceedings of the 15th International Conference on Conceptual Modeling. Berlin,

Heidelberg etc.: Springer 1996, pp. 261-276

[Oul95] Ould, M.A.: Business Processes: Modelling and Analysis for Re-Engineering and Improvement. Chichester et al.: Wiley 1995

[PaBe89] Page, T.W.; Berson, S.E.; Cheng, W.C.; Muntz, R.R.: An Object-Oriented Modeling Environment. In: Meyrowitz, N. (Ed.): Object-Oriented Programming: Systems, Languages and Applications. New York, 1989, pp. 287-296

[Pet92] Petrie, Ch.J. (Hrsg.): Proceedings of the First International Conference on Enterprise Integration Modeling Cambridge, Mass.: MIT Press 1992

[Pla97] Platinum: Object Analysis and Design Facility Response to OMG/OA&D RFP-1. Obtained via http://www.omg.org/library/schedule/AD_RFP1.html

[RaSi87] Rajlich, V.; Silva, J.: Two Object-Oriented Decomposition Methods. Detroit 1987

[Rat97a] Rational: UML Semantics. Version 1.0, 02-13-97, 1997. Obtained via http://www.rational.com

[Rat97b] Rational: UML Notation Guide. Version 1.0, 02-13-97, 1997. Obtained via http://www.rational.com

[Rat97c] Rational: UML Summary.0, 02-13-97, 1997. Obtained via http://www.rational.com

[Rat97d] Rational: UML Semantics. Appendix M1 - UML Glossary. Version 1.0, 02-13-97, 1997. Obtained via http://www.rational.com

[Rat97e] Rational: Appendix M3: UML Meta-Metamodel Alignment with MOF and CDIF. Version 1.0, 02-13-97. 1997, Obtained via http://www.rational.com

[Rat97f] Rational: UML Semantics Appendix M4 -Relationship to OMG Technologies. Version 1.0, 02-13-97. 1997. Obtained via http://www.rational.com

[Rat97g] Rational: UML-Compliant Interchange Format. Version 1.0, 02-13-97, 1997. Obtained via http://www.rational.com

[Rat97h] Rational: UML Semantics Appendix M5 -Relationship to Reference Model of Open Distributed Computing. Version 1.0, 02-13-97, 1997. Obtained via http://www.rational.com

[Rat97i] Rational: UML-Compliant Object Analysis & Design Facility. Version 1.0 beta 1, 02-13-97. 1997, Obtained via http://www.rational.com

[Rat97j] Rational: Mapping of UML to CORBA IDL. Version 1.0 beta 1, 02-13-97. 1997. Obtained via http://www.rational.com

[Ree95] Reenskaug, T.: Working with Objects: The OORAM Software Engineering Method. Englewood Cliffs: Prentice Hall 1995

[Ren82] Rentsch, T.: Object-Oriented Programming. In: SIGPLAN Notices, Vol. 17, No. 12, 1982

[Rob92] Robinson, P.: Hierarchical Object-Oriented Design. London: Prentice Hall 1992

[Rum96a] Rumbaugh, J.: Notation notes: Principles for choosing notation In: Journal of Object-Oriented Programming, Vol. 8, No. 10, 1996, pp. 11-14

[Rum96b] Rumbaugh, J.: To form a more perfect union: Unifying the OMT and Booch methods In: JOOP, Vol. 8, No. 8, 1996, pp. 14-18

[Rum91] Rumbaugh, J. et al.: Object-oriented Modelling and Design. Englewood Cliffs, N.J.: Prentice Hall 1991

[Sch94] Schmauch, Ch.H.: ISO 9000 for Software Developers. Milwaukee, Wis.: ASQC Quality Press 1994

[Sch97] Schnur, B.: Objektorientierung in Versicherungsunternehmen. Die Branche gibt sich bislang noch zurückhaltend. In: Informatik Spektrum,Vol. 20, No. 1, 1997, pp. 52-53

[Sch95] Schwabe, D., Ross, G.: The Object-Oriented Hypermedia Design Model. In: Communications of the ACM, Vol. 38, No. 8, 1995, pp. 45-48

[SeSt87] Seidewitz, E.; Stark, M.: Towards a General Object-Oriented Software Development Methodology. In: Peterson, G.E. (Ed.): Object-Oriented Computing - Tutorial. Vol. 2: Implementations. Washington, DC. 1987, pp. 16-29

[ShMe92] Shlaer, S.; Mellor, S.J.: Object Lifecycles - Modeling theWorld in States. Englewood Cliffs, N.J.: Prentice Hall 1992

[ShMe88] Shlaer, S.; Mellor, S.J.: Object-Oriented Systems Analysis - Modeling the World in Data. Englewood Cliffs, N.J.: Prentice Hall 1988

[Sof97] Softeam: Submission of the specification of Object Analysis & Design Facility OMG RFP response, 1997. Obtained via http://www.omg.org/library/schedule/AD_RFP1.html

[SüEb97] Süttenbach, R.; Ebert, J.: A Booch Metamodel. Fachberichte Informatik, 5/97, Universität Koblenz 1997

[Tas97] Taskon: The OOram Meta-Model - combining role models, interfaces, and classes to support system centric and program centric modeling. A proposal in response to OMG OA&D RFP-1, 1997. Obtained via http://www.omg.org/library/schedule/AD_RFP1.html

[Tay95] Taylor, D.A.: Business Engineering with Object Technology. New York et al.: Wiley 1995

[VeCa92] Velho, A.V.; Carapuca, R.: SOM: A Semantic Object Model - Towards an Abstract, Complete and Unifying Way to Model the Real World. In: Sol. H. (Ed.): Proceedings of the Third International Working Conference on Dynamic Modelling of Information Systems. Delft 1992, pp. 65-93

[Wak93] Wakeman, L.: PCTE: The Standard for Open Repositories. Foundation for Software Engineering Environment. New York et al.: Prentice Hall 1993

[WaNe95] Walden, K. ; Nerson, J.-M.: Seamless Object-Oriented Software Architecture: Analysis and Design of Reliable Systems. New York et al.: Prentice Hall: 1995

[WaPi90] Wasserman, A.I.; Pircher, P.A.; Muller, R.J.: The Object-Oriented Structured Design Notation for Software Representation. In: Computer 23, No. 3, 1993, pp. 50-63

[Wei79] Weick, K.E.: The Social Psychology of Organizing. 2nd. ed., Reading, Mass.: Addison-Wesley 1979

[WFM96] WfMC (Workgroup 1): Interface 1: Process Definition Interchange WfMC TC-1016, Version 1.0 Beta, May 29, 1996. Obtained via http://www.aiai.ed.ac.uk/WfMC/ 1996

[Whi94] White, I.: Using the Booch Method - A Rational Approach. New York et al.: Benjamin Cummungs 1994

[Who59] Whorf, B.L.: Language, Thought, and Reality. New York: Wiley 1959

[Win95] Winograd, T.: From Programming Environments to Environments for Designing. Communications of the ACM, Vol. 38, No. 6, 1995, pp. 65-74

[WiFl94] Winograd, T.; Flores, F.: Understanding computers and cognition - a new foundation for design. 9th print, Reading, Mass.: Addison-Wesley 1994

[Win96] Winograd, T. (Ed.): Bringing design to software. Reading, Mass.: Addison-Wesley 1996

[WiWi90] Wirfs-Brock, R.J.; Wilkerson, B.; Wiener, L.: Designing Object-Oriented Software. Englewood Cliffs, NJ.: Prentice Hall 1990

[You97] Yourdon, E.: Rational Software: The first Billion-Dollar CASE Vendor? In: Application Development Strategies, No. 1, 1997, pp. 1-16