On the purpose of Object-Oriented Analysis

Geir Magne Høydalsvik and Guttorm Sindre

University of Trondheim
Norwegian Institute of Technology
Department of Computer Systems and Telematics
UNIT-NTH/IDT, N-7034 Trondheim, NORWAY.
geirmag@idt.unit.no, guttorm@idt.unit.no

Abstract

The paper discusses the general purpose of analysis and evaluates OOA with respect to this, arguing that OOA does not deliver what it claims to do. The two major problems are that OOA often does not meet the full needs of the analysis phase, and that the transition to design is not always as easy as promised. The last point is illustrated by a solution to the OOPSLA conference registration problem.

Due to the mentioned shortcomings, OOA/OOD was not found sufficient for forming the basis of a common development methodology for three Norwegian software producers in a technology transfer project with our university. The suggestion made is that OOA should become problem-oriented rather than target-oriented.

Keywords: object-oriented analysis, problem-orientation, suitability, design

1 Introduction

The background of this work emerged in the NSR project[1]. NSR is a cooperation between three Norwegian software companies, all somehow relying on object technology and eager to strengthen their competence in this respect. One of the work packages in the project has addressed the possibility for establishing a common object oriented analysis and design methodology within the NSR project.

The partners are rather diverse, one occupied with office information systems, one with traditional administrative information systems, and one occupied with real world modeling. To give some background for the discussion made in following sections we will give a short description of existing methods in the three companies.

TASKON AS have developed an OOD method named OORAM [22, 33, 45]. This method can be characterized by being similar to RDD [44] and OBA [36] through its early focus on responsibilities and scenarios – formalized as *role models*.

SYSDECO AS have developed a method aimed at developing database applications. It is based on modeling of data and presentations, integrated by a

[1]The NSR (Norsk System Rammeverk = Norwegian System Framework) project is funded by NTNF, and the partners are three Norwegian software companies: Metis, Sysdeco, and Taskon. The project runs from 1991–94, and the total budget is 40 man-years, of which 4 man-years are delivered from the University of Trondheim, whose main task in the project is technology transfer.

4-GL. The product called SYSTEMATOR collect design information in a repository and generates applications for commercial databases. Our university has been involved in extending the modeling capabilities to include object-oriented features [26] and dynamic modeling capabilities [32].

METIS AS have developed a method aimed at enterprise modeling and engineering designs (organizations, products, processes). This is achieved in an object-oriented way by using a generic modeling tool named METIS SOFTWARE [2, 41]. At the model level, instances of objects and relationships are maintained, representing real-world entities. At the meta-model level, application and domain specific types are maintained. The model can be changed at any time by adding/deleting objects and relationships, or by changing associated types. Operations can be attached to both types and instances. The model is "alive" at all times and can thus be used as an information system. The lifecycle approach is in the "Adaptable development" category, defined in [14].

Within the NSR project the short term goal is to define a common methodology framework in which partner methods can serve as components. The long term goal is to use this framework as a road map for further development and tighter integration of methods and products. The role of our university is to evaluate research results, and to enable technology transfer to industry.

The decision to evaluate OOA methods was based on the following observations:

1. the need for early lifecycle support reported by the three companies.

2. the existence of object-oriented approaches and a positive attitude towards object-oriented technology in all companies.

3. the existence of both real-world modeling tools and software specification tools within the three companies.

4. the general goal of total lifecycle coverage and a smooth transition between phases.

By studying OOA methods we essentially found programming independent representations of objects and their interrelationships, largely what we previously had considered as design issues. Aspects that we had considered to be analysis – such as requirement acquisition and model validation – was neglected.

Because it is difficult to define exactly what OOA is, we choose to define it in terms of the methods that we evaluated, i.e. [11, 42, 24, 36, 38, 39]. We admit that to use a general term such as OOA is not fair to any individual method, but avoiding such a generalization will lead to an incomprehensible amount of details, blurring the issue of this paper. An important source for general information about the state of the art in OOA, is survey papers such as [6, 15, 29, 45]. With these considerations in mind, we now present what we found to be two general claims of OOA, namely:

1. OOA fulfills the purposes of analysis, and

2. OOA has a smooth transition to design.

Both of these claims is of high relevance and importance to the NSR project. On more detailed inspection, however, we found these claims to be false. OOA turns out to have several shortcomings and does not present more than a partial solution to our problem.

The rest of the paper is structured as follows: Section 2 discusses the different purposes of analysis and design as considered in general in the field of information systems engineering (i.e. not with particular emphasis on object-orientation). In sections 3–4 we go on with showing that usually at least one of these will fail: you may achieve a smooth transition to OOD, but then analysis has probably not fulfilled its purposes. On the other hand, if analysis does fulfill its purposes, the transition to OOD is likely to be hard. Section 5 explain the approach taken in the NSR project. Section 6 briefly outlines some positive trends in OOA. Finally, section 7 gives some concluding remarks.

2 The purposes of analysis and design

Analysis and design are common names for two phases of the software life-cycle. As observed in [14], the exact naming of phases may vary between life-cycle models. The traditional waterfall model [35] listed the phases *1) system requirements, 2) software requirements, 3) preliminary design, 4) detailed design, 5) coding,* and so on. Of these, 1–2 are commonly referred to as analysis, and 3–4 as design. Since the analysis activities concentrate on requirements, another common name for these is requirements engineering.

The more novel fountain model [21], which is specifically intended for object-oriented development, lists the phases *1) requirements analysis, 2) user requirements specification, 3) software requirements specification, 4) system design, 5) program design, 6) coding,* and so on. A difference from the traditional waterfall model is that some overlap between the phases is encouraged. Still, 1–3 in the fountain model clearly correspond to 1–2 in the waterfall model, and thus to analysis and 4–5 correspond similarly to 3–4 in the waterfall model and thus to design.

As observed in [14], the exact boundary between analysis and design is hard to determine. In recent life-cycle models, such as those with evolutionary prototyping and incremental development, the two phases are heavily interleaved — but still, a difference in *purpose* between analysis and design is recognized in most life-cycle models.

> *Analysis concerns the description of the problem and the user requirements, whereas design concerns the construction of a solution, i.e. a software system which satisfies the previously recorded requirements,.*

This distinction is applicable whether the phases are strongly or weakly separated by the chosen life-cycle model.

In the following subsections, we will discuss in more detail the purpose of analysis, in terms of:

- the knowledge which is supposed to be represented,
- the activities which should be undertaken, and
- what requirements the above induces on the languages and methods used.

At the end of the section we will also discuss briefly the transition into the design phase, to contrast analysis and design for the purpose of the further discussion.

2.1 Knowledge to be captured

The motivation at the outset of a software engineering project is to solve some problem in the real world. The purpose of analysis is to model those aspects of the real world which are relevant to the problem [13]. Hence, the analysis should concentrate on the requirements, not the solution.

An nice overview of what an analysis model should cover, is given by [19]:

- *objectives:* these are the ultimate users' expectations towards the entire information system (both computerized and manual parts), i.e. the objectives which are to be fulfilled through the interplay between the computer system and the surrounding human organization.
- *application domain knowledge:* this defines the vocabulary of the application, its meaning and properties.
- *requirements on the environment:* this is a description of the behavior required from the human organization to meet the objectives.
- *requirements on the computer system:* this is a description of the behavior required from the computer system to meet the objectives.

The division of requirements into four parts does not imply that these four parts must be dealt with in any particular order during analysis. Usually they are worked on concurrently, and are mutually dependent: the fulfillment of the objectives must follow from the other three. By giving heed to all these four kinds of knowledge in the analysis phase, a better understanding of the interplay between the organization and the computer system is provided, and the computerized information system becomes easier to maintain. That is, if the application domain, objectives or environment has to change, the analysis documents can be consulted to reveal what system requirements must be changed to adapt to the new situation.

2.2 Activities to be undertaken

From the above, it can be seen that the activities of capturing and modeling domain knowledge, objectives and requirements belongs to the analysis phase. So does the determination of the automation border, which follows from the breakdown of responsibilities between the system and the environment. As noted by [19], this breakdown is an inventive activity – a compromise between needs, costs, and technical feasibility.

Quality assurance (QA) should be a major concern in analysis — errors in a finished system are generally more costly to correct the earlier in the life-cycle they were introduced. The major QA activities in the analysis phase are *verification* and *validation*. Verification means to check whether something is correct with respect to something else — typically: are all the requirements consistent, or do the requirements towards the system and environment guarantee that the overall objectives will be met? Validation means to check whether the stated knowledge and requirements are actually in correspondence with the users' needs. There are many possibilities for misunderstandings between the user and the analyst, which means that an analysis specification will be incomplete and partly invalid. Unlike verification, validation cannot be done by proof, only by close interaction with the users. In addition to discussing requirements directly with users, typical techniques for validation include prototyping, simulation, animation, and language transformations.

2.3 Requirements towards languages and methods

The main requirement towards languages and methods intended for the analysis phase is *problem-orientation*, meaning that:

- the modeling languages should primarily be suitable for describing real world problems rather than the computerized solutions, and
- the methods, accordingly, should guide the analysts towards paying attention to domain knowledge, system objectives, system and environment *requirements*, rather than the outline of a solution to the problem.

Problem-orientation does not only concern *what* a modeling language should be able to express, but also *how* the knowledge is expressed. To facilitate easy capture and validation, the knowledge should be expressed in a manner which is close to the users' perception of the problem domain, rather than being close to a computerized solution. I.e. the burden of translation from the problem domain to the computer system domain should not be put on analysis itself, but on the transition from analysis to design. This is indicated in figure 1 (adapted from [18]) where the upper alternative shows the recommended problem-oriented approach to analysis, with a small gap between the perceived problem and the specified requirements. The lower alternative shows a more solution-oriented approach, where the knowledge is structured in a manner which has got more in common with the following design than with the users' perception of the problem domain. This is commonly called *target-orientation*.

The target-orientation of traditional software development methods has been heavily criticized by researchers calling for a more problem-oriented approach [4, 7, 9, 19, 27, 34]. Typical objections are:

- A target-oriented analysis fails to uncover potential needs for changes in the human organization to be able to run the system — which is a common source for user dissatisfaction.

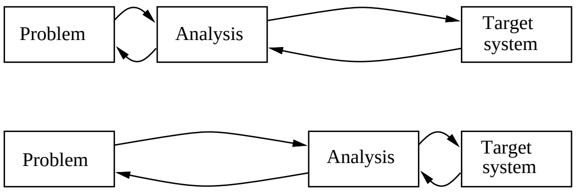

Figure 1: Problem-orientation and target-system orientation

- Target-orientation implies a design decision concerning the automation boundary — a decision which should not be made before the problem is properly understood.
- The system becomes more difficult to maintain in the light of organizational evolution, since the analysis models only describe the computerized information system, not the interplay between the system and the surrounding organization.

Target-orientation puts a heavy burden of translation on the analyst and also on the user, who needs to understand the analysis specification for validation purposes. Hence, the user's ability to take actively part in the QA of the analysis models is strongly reduced.

In addition to the requirement for problem-orientation, the need for verification and validation support implies that languages and methods used for analysis should be formal — to allow for deduction and execution [19]. Lack of formality is a major problem with most of the development methods in industrial use.

2.4 Transition to design

Whereas analysis is supposed to produce a model of a part of the real world, design is supposed to result in a model of a computerized information system. Thus, of the four parts of the total analysis knowledge presented above, only the requirements towards the computerized information system are of interest to the designers. For the interaction between analysis and design, the formal language used in analysis may have to be translated to another specification language more suited to describe the decisions taken during design [19].

The purpose of the design model is quite different from that of the analysis model, namely to:

- describe the structure and actions of the computerized information system (but without going into coding details) — what parts the system solution is going to consist of, and how these parts are supposed to interact,
- specification of each part,
- serve as a medium for communication between designers, and
- be a starting point for the coding phase.

Thus, the design phase receives as its input the requirements to the computerized information system and delivers as its output a system design which (hopefully) satisfies these requirements. For design, verification is still a major activity — i.e. showing that the design satisfies the requirements. Validation, on the other hand, is not so important — the user might not care whether one or the other design is chosen as long as the requirements are satisfied.

From figure 1 it is evident that target-orientation in the analysis phase will yield an easier transition between analysis and design. However, this is of little help if the analysis resulting from that approach is of inferior quality, meaning that the system is developed according to faulty requirements. Clearly, the burden of translation from the problem domain to the language of solutions should not be placed on the shoulders of the users involved in analysis, but on experienced software engineers. This calls for a problem-oriented approach to analysis. Although this means a more complicated transition to design, but this is more than outweighed by the advantages of a higher quality analysis phase in the first place.

3 OOA and the purpose of analysis

Evaluating OOA according to the purposes stated in section 2, one could ask the following questions:

- does OOA fit in with the traditional notion of analysis as a life-cycle phase?
- are OOA methods and languages problem-oriented?
- are OOA methods and languages formal, and do they prescribe steps for verification and validation?

These questions will be addressed in the following subsections.

3.1 OOA and the software life-cycle

Looking at OOA from the life-cycle perspective taken above, it can be observed that many OOA methods do not seem to fit with the traditional meaning of analysis. As observed in [36], many methods assume that the requirements have been established before analysis starts. With this perspective it seems awkward to call OOA "analysis" — what is there to analyze if the requirements have already been established? In this case, it seems that OOA would correspond to what is called preliminary design in the waterfall model.

For most OOA/OOD approaches, the difference between analysis and design is not recognized as the difference between the users requirements and the solution, but simply as the difference between "what" and "how". As stated in [15], which is rather mainstream in its definition of OOA: "Analysis is aimed at describing what a *target system* is supposed to do to obtain an agreement with a customer providing money, while design is aimed at describing *how* the desired system will work..." Unfortunately, the distinction between what and how can be found at any hierarchical level of a system model and in every development phase — every "how" is a solution to a higher level "what", and every "what" is again part of an even higher level "how". The statement that an organization needs to improve its information processing is itself a "how" which is a partial solution to an even higher level "what" (e.g. "the company must make money"). Instead of what/how, the real difference between analysis and design should be defined in terms of

- whether the constructs used in the model are part of the problem-domain (analysis) or part of the solution (design), and
- whether the method addresses communication with the user, including the need for verification and validation.

Clearly, although the distinction between what a stack does and how it does it may be valuable in many contexts, the stack will be a design object in both cases.

3.2 Problem-orientation

The observation that many OOA methods assume the requirements as a starting point indicate that OOA might be closer to preliminary design than analysis in the traditional sense. However, from the citation above, it can be seen that the result of analysis is intended for negotiation with the customer. Hence, it does seem natural to keep on considering OOA as analysis and evaluating it as such. The next question then is if OOA is problem-oriented.

A point often made by proponents of object-orientation, however, is that object-oriented modeling is *natural*, i.e. an object-oriented model is likely to be in accordance with how most humans naturally perceive the structure and behavior of a system. Should it not then be called problem-oriented?

Several objections can be made here. First of all, the argument is rather philosophical, and does not seem to be substantiated. But even assuming that it might be true, it can be observed that:

- You do not get good analysis simply by a model which is in accordance with the way humans think, if the model is of a design.

- An object-oriented representation might be good for some kinds of knowledge, but less suitable for other kinds of knowledge. Since an analysis specification has to contain many different kinds of knowledge, OOA will only present a partial solution.

- The analysis should be close to the user, not to the software engineer. Many users prefer other models than the object-oriented, and developers should adapt to the users in this respect, not the other way around. A dogma in the object-oriented camp seems to be that if the users requirements are not object-oriented, they are wrong. In many cases, such an arrogant attitude towards the way the users reason about their problem-domain is not likely to contribute positively to the development process.

- The main motivation for choosing OOA as an analysis technique often seems to be that the subsequent steps are object-oriented design and programming [5], i.e. to avoid a difficult transition from analysis to design. This is clearly target-orientation — the analysis technique is chosen to fit in with the following design technique rather than the problem at hand.

Two typical examples of knowledge which is hard to capture in a nice object-oriented structure, are:

- business rules [4], which tend to be global of nature and thus not easy to encapsulate within one specific class, and

- dynamics, e.g. processes — especially to give overviews of system dynamics. In an object-oriented specification dynamics are hidden within the object classes and hence statics need to be captured before dynamics [30].

Below, these two points will be looked at in more detail.

Rules

A business rule can be defined as a law or custom which guides the behavior or action of the agents connected to the organization [4]. The computerized information system will also be such an agent, and one of the main motivations for capturing business rules directly in the analysis phase is to make it easier to maintain the computer system, since it then becomes easier to make the necessary changes to the system when the business rules change — as they quite often do [4]. Examples of business rules may be:

R1: Product A should never be cheaper than product B.

R2: When the payment of a bill is two weeks overdue, it is required to send a reminder to the customer.

These rules are quite ordinary rules that almost any company might have, and which the customer would certainly want to see included in his systems requirements. However, with a strict encapsulation this cannot be done:

- **R1** would have been easy if it had said "Product A should never be cheaper than 1 dollar" or "The price of Product A should be 150% of its production cost", in which case it could have been specified as an integrity constraint on the product price attribute. However, as it stands, it involves two different object types, A and B. With a strict encapsulation this must be expressed in the operations for changing the prices of A and B, respectively — i.e. that the change of price for A is prohibited if the price becomes lower than B and the change for B is prohibited if the price becomes lower than A. This, however, is design, since we must specify *how* the requirement is ensured, rather than *what* is required.

- **R2** again involves several object types, and also an action. Moreover, it involves time, which will be particularly problematic to a object-oriented specification — to capture this rule we might have to introduce an object "time" whose "change" operation sends a message to every object which has a time dependent method — for instance "bill", which if unpaid will check if it is two weeks overdue, and if so send a message to a "reminder" object and so on and so forth. This is clearly implementation, not requirements analysis.

The examples indicate that an attempt to formalize general business rules within a object-oriented structure, will force the analyst into a jungle of design and implementation decisions. Since business rules tend to be of a global nature, it is difficult to attach them to specific classes. Although we did not find any OOA methods that incorporated general rules – we expect this to be so, as the technology matures. We believe that this could be obtained by relaxing the encapsulation and by attaching rules to sets of types.

Dynamic Modeling

Object-oriented models give good overview over the various objects in the system and the relations between these. However, in most methods the focus is very static. Dynamics are scattered — since operations must belong to specific objects, they are necessarily at a low level of detail and they rarely correspond to the tasks that the users perform in their work. This makes it difficult to get an overview of the system dynamics, because [5]:

- the sequences of operations execution will not be visible, or if made visible, they are usually unwieldy because the operations, in the first place, are only very low level.

- Higher level tasks involving operations from several objects, will not be visualized and can only be established by detailed reading of information from several parts of the model.

There are some OOA approaches which have improved the attention to dynamics, namely *use cases* [24] and *scenarios* [36]. These put the focus initially on dynamics rather than statics, investigating system behavior before classes are identified. However, the view on dynamics is rather low level, focusing on events, states, and transitions, not on aggregate dynamic concepts such as processes. Thus, the dynamic modelling proposed by these approaches are typically bottom-up rather than top-down, which might be useful in some cases, but it is hard to get overview pictures of the system dynamics that way.

As admitted in [5] the capturing of performance requirements, which may be essential in many applications, is very problematic with object-oriented specifications. And as indicated in the section on business rules, any requirement involving time will be difficult to capture without diving down into implementation detail. Since time is somehow important to most customers (otherwise, they might not need computers at all), this situation is clearly not satisfactory.

The criticism above does not mean that a strictly process- or rule-based analysis technique would be more appropriate than an object-oriented one. The point is rather that the real world should always be modeled in the way which is closest to the user's perception. If the user provides domain knowledge of objects, object-oriented modeling may be the most appropriate. On the other hand, if the user describes some domain processes, a process-oriented view might be better. And if the user provides business rules, it should be possible to model these directly, without turning into awkward design considerations. Thus, all specific techniques for analysis can only be considered as *partial* — an analyst should not be restricted to one specific structure or technique but always choose what is most appropriate.

3.3 Verification and validation

Currently, most languages and methods for OOA are informal. This means that

- verification is poorly supported, and

- useful validation techniques, such as prototyping, simulation, animation, language conversions, and paraphrasing through logical deductions, are not supported. Besides, manual validation will be particularly difficult if the user does not think in object-oriented terms.

The lack of formality is not a problem of object-orientation as such — it is possible to make just as formal methods and languages with this paradigm as with any other, and to some extent this has already been done [10, 12, 20]. The lack of formality is thus more a sign of the current immaturity of OOA.

More disappointingly, however, most OOA methods do not suggest any manual steps for verification and validation either. In general these problems seem to have been largely ignored by the researchers. Of course, traditional waterfall development with structured analysis is not very good at V&V either. But software engineering has progressed a lot since the 70's and it would be strange if the proponents of OOA should be satisfied with improving on a 20 year old method, corporating work in requirements engineering in the meantime [31].

4 OOA and the transition to design

While the previous sections discuss the purpose of OOA, we will now take a closer look at the transition to design. We do not doubt that in some cases it is legitimate and sufficient to do an object-oriented analysis. What we do doubt in this case, is the necessity of finding a smooth transition to design. The key issue is the objectiveness of OOA; although it is possible to construct analysis models that have a direct mapping into design, there are no guarantees that this will happen – except to intimately consider the target system while doing analysis.

Taking enterprise modeling as an example, using a technique to capture and understand the general objectives and constraints in an organization. Such a model can be stated in terms of objects and relationships, but since the purpose is to reflect the enterprise, they do not necessarily have a close resemblance with application objects that support the enterprise. Thus, we can not expect these models to directly map into some "unknown" information system. On the other hand they are proper analysis models, modeling both information systems and their external environments. Enterprise models are intended for understanding the organization, but at the same time they contain essential information for any information system that are intended to support it. There are no miracles – transition to design must transform analysis objects into design objects.

An excellent paper addressing these problems is written by McGinnes [28], in which he argue for more intuitive and flexible modeling constructs than those offered by OOA. A quote form this paper reflect our experience: *If we make no distinction between analysis and design, as proposed in many methods, and instead view the process as one of progressive refinement, then we are stuck with the idea that the objects in the system are equivalent to those of the real world. In fact this is not the case: it is often necessary to "bend" reality to fit into the object model.*

To illustrate the statements above, we will outline two alternative models based on the same problem statement, both using a single OOA notation. The only difference between the two models is their purpose, one describing the organization, the other describing a specific information system. We do not claim that these models are the only models that can be made, we only want to illustrate what can happen when the viewpoint is different. In section 4.4 the two models are compared, and a discussion of their differences are made.

4.1 The OOPSLA Conference Registration Problem

The example is motivated by a problem statement given by Rumbaugh in [37]. A smaller, alternative problem statement of the original problem is outlined below. The original requirement specification given by Rumbaugh is aimed at describing some "information system to be designed". The one given below is aimed at being a pure description of what is happening between authors and the program committee, i.e. we have no information system in mind.

1. The program committee announces "call for papers".
2. An author receives a call for papers and decides that she will submit some of her work. She writes a paper and sends it to the program committee. There may be several authors, but only one reply address.
3. The program committee registers the contributed papers upon receipt.
4. At a certain point in time the program committee distribute the papers among those undertaking the refereeing. Each paper is sent to three distinct referees.
5. The program committee continuously collect reports from the referees.
6. At a certain point in time the program committee select papers for inclusion in the program and notifies the authors about the selection.
7. The author receives the selection results.

We assume that this is an accurate description of the OOPSLA organization.

4.2 Describing the Organization

In this section our analyst looks at the OOPSLA organization for the purpose of expressing a limited part of it, armed by her OOA method and notation.

An object-oriented analysis would typically create a *static model* and a *dynamic model*. The static model would typically include types/classes and relationships between those, i.e specialization/ generalization, aggregation, and association. The dynamic model would typically include message passing between objects and state transitions inside objects. Message passing is often described by message sequence diagrams, and state transitions by

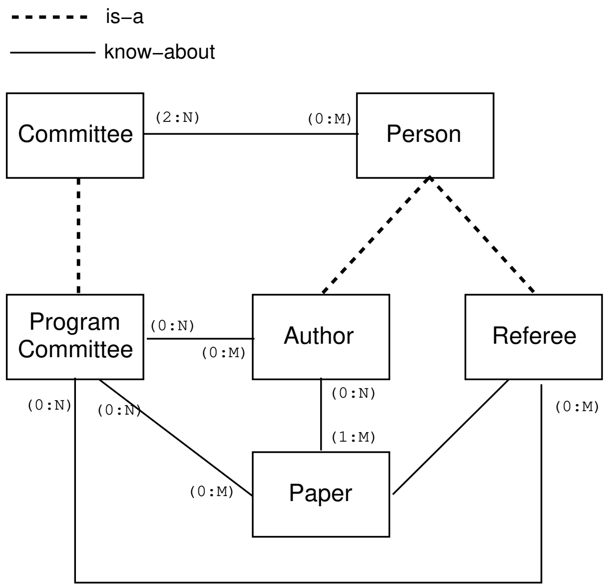

Figure 2: *Static model: The OOPSLA Conference Registration Problem*

Harel statecharts [16]. Although state transitions (state, event, action) are important for most OOA methods, we do not include these to simplify the example.

The static model is defined as depicted in Figure 2. This model is intended to be representative for available OOA methodologies, that is, representative in basic ideas, not in notation or amount of details.

A very simple dynamic model is depicted in Figure 3. The ellipses denote objects, and arrows denote message sends. The logical message sequence is described by numbering the messages. The data flow is indicated by using parameters, representing other objects.

A description of the resulting classes is shown in Figure 4. The convention used for the "know-about" relationship is an attribute with the same name as one of the classes. The cardinality is indicated by (min:max).

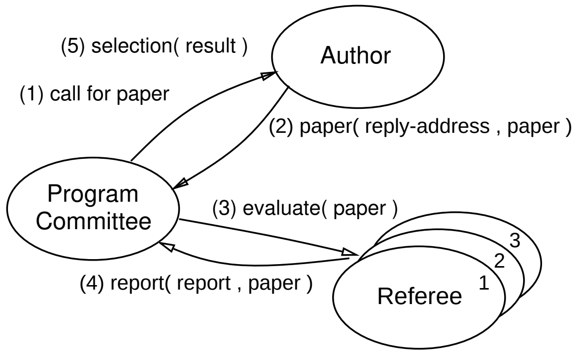

Figure 3: *Dynamic model: The OOPSLA Conference Registration Problem*

Class	Attributes	Association	Inheritance	Services
Papers	title content	Author(1:N)		read write copy
Person	name address sos.sec.#	Committee(0:N)		
Committee	purpose	Person(2:N)		
PC	deadline	Author(0:N) Referee(0:N) Paper(0:N)	Committee	paper report
Author	occupation	PC(0:N) Paper(0:N)	Person	call–for–paper selection
Referee	competence	PC(0:N) Paper(0:N)	Person	evaluate

Figure 4: *Real-world model*

4.3 Describing Software

In this section we look at the OOPSLA Conference Registration Problem from a different angle. Now, the purpose of our modeling efforts is to describe a system that support the Program Committee in its work, making a conference program.

With this goal in mind our analyst will set up an automation boundary. The goal will be to determine what is "inside" and what is "outside" the system, and to determine the users' interface. Note that the meaning of system has changed from being the organization to being the software system. Our analyst recognizes that communication between the program committee and external agents (authors and referees) is done by US-mail. The information system to be designed is a system used by the program committee for the purpose of keeping information about submitted papers up to date.

The program committee knows about a set of referees, and as submissions occur, the program committee knows about a set of papers and a set of authors. The static model is depicted in Figure 5.

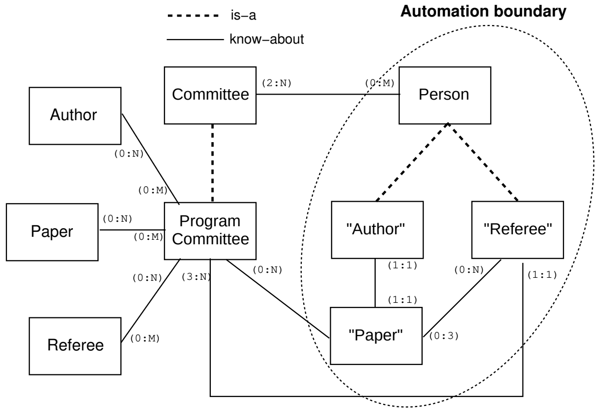

Figure 5: *Static model with automation boundary.*

The important thing to notice is that the classes inside the automation boundary changes their nature. As a result we need a different set of objects to represent "Author", "Referee", and "Paper", which now becomes information objects reflecting corresponding real world objects. The different cardinalities on objects and information objects indicate that the semantics are different.

The program committee wish to see a computerized information system that gives them access to various address, paper, and status information. For each referee they must know the address and the papers allocated to that referee. For each paper they must know about the following:

- Reply address.
- Title, subject, author (for program description).
- Referee allocation.

- Referee process: Paper sent, report received.
- Selection results: None, accepted, rejected.
- Author notification: None, date.

An overview of information objects and corresponding properties is given in Figure 6. The program committee is considered to be the user of the information system – and represents the user. The author object is considered as the reply address, so the paper has only one author.

Class	Attributes	Association	Inheritance	Services
Paper	title subject reply–address referee–allocation paper–sent report–received selection–results report–received author–notification	Author(1:1) Paper(0:3)		read copy
Person	name address soc.seq.#			
PC	deadline	Paper(0:N) Referee(3:N)	Committee	paper report
Author		Paper(1:1)	Person	
Referee	competence	PC(1:1) Paper(0:N)	Person	

Figure 6: *Software model*

4.4 Comparing the Models

The above analysis was done with the objective of making the two models as similar as possible, i.e. to use the same names. In reality there are no agreement on the names or properties of classes, this depends on the purpose of the model and on the background of the analysts. Techniques such as Glossaries [36] must be used to ensure a consistent terminology.

Below we have listed some problems related to the transition between analysis and design. We claim that this is not due to bad analysis or design, but is due to differences in purpose and usage of the model.

- **The relative importance of classes may change.** By setting up an automation boundary the analyst will change his viewpoint – and the resulting analysis will look different. Generally, new classes may appear and disappear. As pointed out in [36]: *"Things" well suited to the paper world may not be well suited to the electronic world.*
- **Concurrency may change.** Real world objects usually act concurrently. This is often reflected in an analysis model. In transition to design it is often necessary to serialize the objects. Some methods do mention this, but few give any guidelines in how to do it.
- **Behavior may change.** In the real world model, the author object is active (writing a paper, sending a paper). In the software model the author is reduced to a passive record (name, address). We find that a more detailed analysis creates more differences, e.g. by describing behavior in terms of a state transition diagram. In our model this results in having two objects, one representing the author, the other representing the information about the author.
- **Services may change.** In the real world model, the author object must respond to a "call for paper". In the software model this is totally irrelevant, what matters is that a paper is received by the program committee. Alternatively, we could move the "call for paper" service to some computer object that would write address lists, and thus change the semantics of the service.
- **Attributes and relationships may change.** In the real world model the program committee know about authors, in the software model it is the "responsibility" of a paper to "know about" its authors and its reply address. This may result in different structure both with respect to specialization/generalization and association relationships.

What we learned from the OOPSLA case is: It is not that we use the object abstaction that is important for a smooth transition to design, but how we use it.

Our experiences with analysis and design makes us believe that the available object-oriented methodologies have a too simplistic view on analysis, and thus ignoring the gap between analysis and design. If we by analysis mean *creating real world models*, it is generally not possible to talk about isomorphism between analysis and design. The smooth transition towards an implemented software system can not be obtained before we only consider design objects. The OOA view that OOD simply adds implementation and presentation specific classes, is simply not true.

5 Consequences for the NSR project

Being problem-oriented, analysis should be appropriate for modeling any real-world activity, rather than just the information processing activities to be automated. In this section we will outline some work, within the context of the NSR project, aimed at acheiving this goal.

5.1 Roles

TASKON has developed the concept of role models to describe their software [33]. A role is an object abstraction, describing a particular purpose or viewpoint on the object. A role model is a structural abstraction that describe the context in which the object play a certain role. Simple role models can be combined by synthesis operations, thus forming complex structures. In [23] we shows that the technique has promising properties with respect to modeling organizations.

Another use of roles can be found in [8]. In this model, roles are used for describing object behaviour as evolution. A class is divided into roles, each role has properties, states, operations, as well as transition rules – which define when to change roles. E.g. a class Car may have different roles such as production, transportation, insurance, accidents, etc. The paper argues for using a role approach to requirements specification.

Other references to roles include [43], a philosophical and formal treatment, and [40] a social science approach. At the University of Trondheim, we are currently investigating the possibility for integrating these concepts. The idea is to model the world in terms of objects that travels between contexts, a context is governed by rules and described in terms of role models.

5.2 Data Flow Diagrams

METIS, with about 10 years experience in modeling engineering designs in terms of objects and relationships [41], have recently added a process model similar to DFD's to represent work processes [2]. This should indicate that message sequence diagrams, originating from the telecommunication area [25] , is not the only valid paradigm for modeling system dynamics.

In the article [30], we describe an intuitive and yet powerful language to model the real-world. The language is based on some small, but essential modifications to the DFD-diagram. The processes, flows, and stores of conventional dataflow diagrams are here interpreted as transformations, transportations, and preservations of items. These divides the representations of real-world dynamics in three orthogonal aspects: matter, location, and time.

This approach distinguishes between ideal- and real- representations. An *ideal process* represents a change of matter while keeping time and location constant. An *ideal flow* changes the location of items, while keeping matter and time constant. An *ideal store* changes time while keeping the matter of items and their locations constant. An ideal representation is *intra-decomposable*, e.g. processes can only be decomposed into subprocesses. Their real counterpart do not have this restrictions, thus being *inter-decomposable*, e.g. a store can be decomposed into processes, flows, and stores.

These concepts – together with some additional concepts called item, link, and port – are shown to have an object-oriented interpretation. Thus, choosing a highly process-oriented modeling paradigm during analysis does not restrict us to choosing a similar structuring principle for design.

5.3 Further Implications

All concepts should have well defined semantics, thus formality is required. However, simplicity is of equal importance, both needed for validation purposes. Thus, our objective is to find simple and yet well defined modeling constructs. With respect to validation we focus on dynamic aspects because this is by far the most difficult part to validate. The techniques considered are symbolic execution and simulation, coupled with visualization techniques. Work in this area can be found in [1, 17].

6 Positive trends in OOA

In this section we will briefly indicate what we consider to be positive trends in OOA. First, OOA has evolved into putting focus on system dynamics. This has been adopted by most methods in the form of object-iteraction models and state transition diagrams.

Second, some methods have recognized the difference between analysis and design. OBA [36] can be taken as an example. The basic technique in this method is to collect information into scenarios (scripts) that describe sequences of actions. Novel features of this method include:

1. it does not assume that there exists a previously written requirement specification.

2. it put focus on the analysis context, including goals and objectives.

3. it considers external objects – as initators of a scenario.

4. attention to requirement elictation is given by creating scenarioes from a structured interview process.

5. symbolic execution can be obtained, because scripts and state transitions are coupled through pre- and post- conditions.

OBA is quite similar to OORAM [3, 22, 33], adding analysis context and lacking the flexibility obtained by synthesis of role models.

Third, formal approaches have started to appear, OSA [42] being the most notable example. The models include an object-relationship model and object behaviour models, i.e. message sequences and state transitions. Besides its formality, novel features include: views, high-level abstractions, constraints, and model integration.

In the future we expect OOA methods to become more problem-oriented and put focus on the surrounding organization.

7 Conclusion

The article has presented some general views on the purpose of analysis and design, and based on this it has been shown that OOA has several shortcomings, most importantly in being target-oriented rather than problem-oriented. An awkward assumption of most OOA methods is that the requirements exists prior to OOA, which makes OOA more a preliminary design.

We believe that the purpose of OOA must be found in the general objectives of analysis, not in its closeness to design. Furthermore, given the existence of a difference between OOA and OOD, we have argued that a smooth transition to design is highly questionable. We have also argued that objects are not sufficient for a adequate representation of real-world concepts. We need a wider range of modeling constructs, flexible capturing of concepts, experimentation facilities, validation, views and high level abstractions. We consider OOA largely to be a collection of design methods, offering little support for analysis – thus finding OOA more harmless, than useless.

8 Acknowledgements

Thanks to John Dodd (Texas Instruments) and George Wilkie (University of Ulster) for very critical comments. Thanks to Professor Reidar Conradi, John Krogstie, Sivert Sørumgård, and Sigurd Thunem – for comments and discussions.

References

[1] V.De Antonellis and L. Vandoni. Validation of Object-Oriented Dynamic Specifications. In *Proceedings of 26'th HICSS*, volume 4, pages 399–408, January 1993.

[2] METIS A/S. METIS Information and Process Modeling Handbook. Technical report, METIS A/S, February 1992. Preliminary Edition.

[3] TASKON AS. OOram, Object Oriented Role Analysis and Modelling. Technical report, TASKON Work Environments, January 1993. Handbook – Preliminary version.

[4] F. Van Assche et al. Information systems development: a rule-based approach. *Knowledge-Based Systems*, 1(4):227–234, 1988.

[5] S. C. Bailin. An object-oriented requirements specification method. *Communications of the ACM*, 32(5):608–623, May 1989.

[6] Mehmet Askit & Lodewijk Bergmans. Obstacles in Object-Oriented Software Development. In *In OOPSLA'92 Proceedings*, 1992.

[7] A. Borgida et al. Knowledge representation as the basis for requirements specification. *IEEE Computer*, 18(4):82–91, April 1985.

[8] B.Pernici. *Object-Oriented Development*, pages 75–100. Universite de Geneve, editor D.Tsichritzis, 1989.

[9] J. A. Bubenko jr. On concepts and strategies for requirements and information analysis. In *Information Modelling*, pages 125–169. Chartwell-Bratt Ltd., 1983.

[10] Stephen W. Clyde, David W. Embley, and Scott N. Woodfield. Tunable Formalism in Object-Oriented Analysis: Meeting the Needs of both Theoreticians and Practitioners. *OOPSLA'92 Proceedings*, 1992.

[11] Peter Coad and Edward Yourdon. *Object-Oriented Analysis*. Prentice Hall, 1990.

[12] Derek Coleman, Fiona Hayes, and Stephen Bear. Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design. *IEEE Transactions on Software Engineering*, 18(1):9–18, January 1992.

[13] A. M. Davis. *Software Requirements Analysis & Specification*. Prentice-Hall, 1990.

[14] A. M. Davis et al. A strategy for comparing alternative software development life cycle models. *IEEE Transactions on Software Engineering*, 14(8):1453–1461, October 1988.

[15] Dennis de Champeaux and Penelope Faure. A Comparative study of object-oriented analysis methods. *Journal of Object-Oriented Programming*, pages 21–33, March 1992.

[16] D.Harel. Statecharts: A Visual Formalism for Complex Systems. *Sience of Computer Programming*, 8:231–274, 1987.

[17] Dr.Folkert W.Wierda. Playing With Toy Trains or an Empirical Investigation into the Feasibility of Dynamic Modelling of Information Systems. In *Proceedings of 26'th HICSS*, volume 4, pages 429–437, January 1993.

[18] J. Hagelstein. Problem-oriented requirements engineering. In G. Shoenmakers, editor, *Colloquium Software Specificatie Technieken*. Academic Services, Schoonhoven, 1987.

[19] J. Hagelstein. Declarative approach to information systems requirements. *Knowledge-Based Systems*, 1(4):211–220, 1988.

[20] Richard Helm, Ian M. Holland, and Dipayan Gangopadhyay. Contracts: Specifying Behavioral Compositions in Object-Oriented Systems. *OOPSLA'90 Proceedings*, 1990.

[21] Brian Henderson-Sellers and Julian M. Edwards. The Object-Oriented Systems Life Cycle. *Communications of the ACM*, 33(9):142–159, September 1990.

[22] Jon Harald Holm. Dynamic Modeling in OORAM. Technical report, Norwegian Institute of Technology, December 1992. MSc thesis, REBOOT-STP TR 94.

[23] Geir Magne Høydalsvik and Guttorm Sindre. Object-Oriented Role Modeling for the Analysis and Design of Organizational Information

Systems. In *Proceedings of 26'th HICSS*, volume 3, pages 159–168, January 1993.

[24] Ivar Jacobson et al. *Object-Oriented Software Engineering – A Use case Driven Approach.* Addison-Wesley, 1992.

[25] Rolv Bræk, Geir Hasnes, and Øystein Haugen. Engineering Real Time System with An Object Oriented Methodology based on SDL. Technical Report SISU-90001, March 1990.

[26] Petter Lowzow. Object Oriented Interface for a Relational Database. Technical report, Norwegian Institute of Technology, December 1992. MSc thesis, REBOOT-STP TR 87.

[27] R. N. Mathur. Methodology for business systems development. *IEEE Transactions on Software Engineering*, 13(5):593–601, May 1987.

[28] Simon McGinnes. How Objective is Object-Oriented Analysis? In *Proc. CAiSE'92, Manchester*, 1992.

[29] David E. Monarchi and Gretchen I. Puhr. A Research Topology for Object-Oriented Analysis and Design. *Communications of the ACM*, 35(9):35–47, September 1992.

[30] A. L. Opdahl and G. Sindre. Concepts for real-world modelling. In C. Rolland et al., editor, *Advanced Information Systems Engineering, Proc. CAiSE'93, Paris*, pages 309–327. Springer Verlag (LNCS 685), 1993.

[31] Proceedings. *IEEE International Symposium on Requirement Engineering.* IEEE Computer Society Press, 1993.

[32] Student project. DYNAMOO: Dynamisk modellering av objekter. Technical report, Norwegian Institute of Technology, November 1992. REBOOT-STP TR 85 (in Norwegian).

[33] Trygve Reenskaug et al. OORASS: seamless support for the creation and maintenance of object oriented systems. *Journal of Object-Oriented Programming*, 5(6):27–41, October 1992.

[34] G.-C. Roman. A taxonomy of current issues in requirements engineering. *IEEE Computer*, 18(4):14–24, April 1985.

[35] W. W. Royce. Managing the Development of Large Software Systems: Concept and Techniques. In *Proceedings of WesCon*, August 1970.

[36] Kenneth S. Rubin and Adele Goldberg. Object Behaviour Analysis. *Communications of the ACM*, 35(9):48–62, September 1992.

[37] James Rumbaugh. Designing bugs and dueling methodologies. *Journal of Object-Oriented Programming*, January 1992.

[38] James Rumbaugh, et al. *Object-Oriented Modeling and Design.* Prentice Hall, 1991.

[39] Shlaer and Mellor. *Object Lifecycles, Modeling the World in States.* Yourdon Press, 1991.

[40] Baldev Singh and Gail L. Rein. Role Interaction Nets (RINs): A Process Description Formalism. Technical report, MCC Technical Report Number CT-083-92, July 1990.

[41] Tor G. Syvertsen, Frank Lillehagen, and Morten Løvstad. A Generic Object Model for Engineering Design. *In proceedings TOOLS-EUROPE'92*, March 1992.

[42] David W.Embley, Barry D.Kurtz, and Scott N.Woodfield. *Object-Oriented Systems Analysis – A Model-Driven Approach.* Yourdon Press, 1992.

[43] R.J. Wierenga. Algebraic Foundations for Dynamic Conceptual Models. Technical report, Vrije Universiteit te Amsterdam, 1990. PhD thesis.

[44] Rebecca Wirfs-Brock, Brian Wilkerson, and Lauren Wiener. *Designing Object-Oriented Software.* Prentice Hall, 1990.

[45] Rebecca J. Wirfs-Brock and Ralph E. Johnson. Surveying Current Research in Object-Oriented Design. *Communications of the ACM*, 33(9):105–124, September 1990.