Logics for OO information systems: a semantic study of object orientation from a categorial substructural perspective

de Haas, E.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: http://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 1

The object oriented development practice

When it comes down to it, the real point of software development is cutting code. Diagrams are, after all, just pretty pictures. No user is going to thank you for pretty pictures; what a user wants is software that executes.

Martin Fowler, UML Distilled; A brief guide to the Standard Object Modeling Language ([FowlerScott00])

"Here at our company we are doing business in Wuzzels. Wuzzels have Wazzels and this distinguishes them from the Buzzels.". When, in practice, an information modeler comes into a company to commence his task, he will need to capture information on objects he possibly does not know anything about. Nothing about the structure, the behavior, nor the interrelations between the objects. He will more or less start with a growing collection of labels, that gradually gets more structure and meaning. Moreover he will discover kinds of objects, relations between certain kinds of objects and constraints on the structure and relations of objects. Nevertheless the information modeler will need to immediately start writing down preliminary versions of the model of the world he is trying to capture. The preliminary model he writes down will be used to communicate with the experts and users that play an important role in the piece of the world he is modeling.

Languages that bear concepts from the object oriented methodology are used in the information capturing process and in analysis and design. Such practice imposes some requirements on the language in which information system models are written down. For example the language must be able to elegantly denote objects that have a partial nature or that have unknown structure and behavior,
and still we need to be able to interpret the language such that the expressions really denote some part of the information model. Moreover because of the additive way of working the language must be such that we can easily extend existing descriptions of the world, and that the interpretation of such an extended model is an 'elaboration' of the former model.

Object oriented languages have already been used much longer in programming and database practice. The most referred reason to use these kind of languages in the semi-formal world of computer coding was that the object oriented languages contain concepts that enable one to talk 'naturally' about the information that needs to be coded. This means that the idea is that the notions used in object orientation are founded on a natural intuition to talk about information. This also explains the popularity of object oriented languages in analysis and design.

In this chapter we will introduce the reader into the problem domain covered and researched in this thesis. We will discuss three important concepts that, in the general process of information capturing with object oriented languages, are important items in our analysis of the object oriented development practice. These are labels, partial descriptions of objects and partially or even non-wellfounded defined objects.

Moreover we will give a brief overview of the origins of the concepts of object orientation in object oriented information systems. We will present in more detail the Unified Modeling Language (UML), which is a standard in the object oriented design and analysis practice. This language is the most influential reference for the concepts of object oriented information systems. Consequently we will explain the connection between a design language, like UML, and the conceptual 'lower level' coding languages for databases and programming constructs.

In this chapter we will also briefly describe the information development practice. This practice covers the whole trajectory from analysis and design to implementation in programming and database coding languages. We do this for two completely different reasons. First of all we want to identify a number of requirements of the object oriented languages that are related to the use of these languages (especially for analysis and design languages). Secondly we want to make the reader acquainted with the field of application for which this research is done: Object Oriented modeling, design and development.

The main focus of this chapter will be on conceptual aspects of object orientation. These notions are most apparent in the analysis and design practice, but

\footnote{More precisely, it should not be the case that expressions that hold in the former model are not satisfied by the elaborated model, unless this was explicitly stated in the additive modeling step.}
1.1 Labels, partial descriptions and non-wellfoundedness

Notions that are key in this thesis are labels, partial descriptions and non-wellfoundedness. In this section we will explain these notions in more detail. Let us take a look again at the example from the beginning of this chapter:

"Here at our company we are doing business in Wuzzels. Wuzzels have Wazzels and this distinguishes them from the Buzzels."

Label. An information analyst who has never been introduced to the subject of wuzzels, wazzels and buzzels will need to start his model, based on the information from the above sentence, with a collection of labels. These labels here clearly denote types of objects. But even though the information modeler does not have a clue of what its interpretation should be, he can start to interpret that a particular wuzzel is some kind of object. This object does not have anything but a label.

Partial descriptions. An important piece of information in the above sentence is that 'the fact of holding a wazzel' is a discerning fact. This means the following: To hold a wazzel can be a property of an object. And moreover, holding a wazzel is a characterization of certain types of objects, among which are wazzels. Also to hold a certain wazzel is a partial description of a wazzel. To make the example more concrete consider the following: To have a 'beard' is a property of 'ancient philosophers'. Having a 'beard' is characteristic for 'ancient philosophers'. And 'he having that white beard' is a partial description of Socrates.

Non-wellfoundedness. The information modeler is quite certain that he will learn a lot more about the wuzzels, wazzels and buzzels. He will get to know intrinsic properties and accidental properties of certain wuzzels, wazzels and buzzels. But he will never know that he grasped all that can be said about the wuzzels; even more drastically at certain stages in the iterative process he will be certain that he did not grasp everything he needs of the wuzzels, wazzels, and buzzels. This is part of the way he works. Every object can be extended by discovering more and more properties (descriptions), and the properties in turn (seen as objects themselves) can again be extended. The process of extending can possibly never end (either by cycles or by infinite chains). But that means that these objects will be not well founded. Moreover one never knows whether the objects are totally described in terms of it properties.
Chapter 1. The object oriented development practice

The above notions are only briefly introduced in this section. They will be elaborated in the semantic study later on in this thesis, and there the importance will become evident.

1.2 Object Oriented Information Systems

The numerous developments in information systems from the last two decades, both in practice and in theory, have contributed to the obtaining of proposals for so called Next Generation Information Systems ([Comm.ACM91nlO]). One of the most influential developments is object orientation. The concepts of object orientation are the subject of this chapter. Concepts from other conceptual worlds, however, have also contributed significantly to the arsenal of technologies used in current information systems. We will also briefly give attention to the relational paradigm and the logical paradigm. The relational paradigm is well known from the Entity Relationship design languages (ER) and the language for relational databases (SQL). The logical paradigm is used in knowledge based rule systems, deductive databases and logic programming.

1.2.1 An ontology of object oriented information systems

Among the recent developments in information systems, the most influential developments are probably those following the principles of the object-oriented (OO) programming paradigm. The OO programming paradigm has its origin in the SIMULA programming language ([Pooley87]), which was proposed in the late 1960s. The concepts underlying this paradigm became especially popular in the 1980s with the introduction of the programming languages SMALLTALK ([Smith95]), EIFFEL ([RistTerwilliger95]), and later C++ ([Stroustrup91]) and JAVA ([ArnoldEtAlii00]).

In the 1980s the paradigm of object orientation entered into the world of databases. Together with other developments in databases, in particular complex values and notions from semantic databases, this constituted the object oriented database (OODB) paradigm ([AtkinsonEtAlii89])

The concepts of the OODB paradigm entered the world of databases in several different disguises. Firstly, in OO programming there was a need for a persistent store for the objects that were created by the executions of the (OO) programs. In this context persistence means that the lifetime of the objects that occur in a program is longer than the lifetime of the program run that created the objects; this is in order to give other programs the opportunity to use these objects. In this disguise, some primitive database notions were incorporated into the world of OO programming languages. Another disguise of the concepts of the OODB paradigm was created the other way around. The new concepts of the OODB
paradigm were incorporated into existing database models. This gave rise to
types such as the 'object relational database model' and the 'deductive object
oriented database model' ([Abiteboul90]).

It has to be noted that the OODB paradigm rose from implementation efforts,
and is not based on a precise formal model. Since its appearance, several models
various degrees of formality have been developed. None of these models de-
veloped so far encompasses all the features that are associated with the OODB
paradigm, and also, a universally agreed upon model has not yet emerged. There
have been proposals for a standard model of OODB systems, of which the ODMG-
93 proposal ([Cattell94]) and its successors ODMG 2.0 ([Cattell97]) and ODMG
3.0 ([CattellEtAlii00]), from the ODMG group is the proposed standard. This
is the case, because a large number of influential providers of OODB systems
committed their efforts to the proposal. Unfortunately, the proposal suffers from
many conflicting compromises, and is very sloppy. It also completely lacks formal-
ity and rigor in the semantics², and is much criticized. Another very influential
proposal for standardizing the OODB concept is the SQL-3 standard ([SQL3]).
Although this standard is historically based on the relational database model,
it has incorporated many of the popular concepts of object oriented information
systems, and in particular the OODB concepts. Much work is done comparing
and synthesizing the two above mentioned proposals. The efforts to clarify the
different views on the concepts of the OODB paradigm have initiated three man-
ifests ([AtkinsonEtAlii89], [StonebrakerEtAlii90] and [DarwenDate95]). These
manifests list in an informal manner the required features of OODB systems.

Whereas the first manifest concentrated solely on the concepts of the OODB
paradigm, the second and third manifest stressed the importance of incorpo-
rating the fruitful notions of more traditional database systems, especially of
the relational database systems. Contrary to OODB systems, relational systems
([Codd70], [Ullman88]) evolved from a precise formal model, equipped with a
high level declarative language. The advantages of the theoretical clearness, the
ad hoc query mechanism and the declarativeness of the language, inspired the
implementation efforts of the relational database model for use in practice. The
relational model, enriched with a lot of features and provided with a standard³
query and data definition language (SQL), is considered one of the most successful
theoretically impaired languages in information systems.

Another mathematical model that found its way into information systems is
that of logic programming. The resulting paradigm of deductive databases is com-
mended for is declarativity in combination with its computational power. Fur-
thermore it enables one to incorporate rules of knowledge into the database, i.e. it

²syntax of the languages in the ODMG proposal are formally defined in BNF, although the
BNF syntax conflicts sometimes with the informal presentation of the syntax
³Although not everybody is happy with SQL, it is the unchallenged standard.
Chapter 1. The object oriented development practice

gives the possibility to bring in concepts of the world of knowledge bases. Combining notions from deductive databases and OO databases has become an important matter of research and controversy. The main problem here concerns the question whether the notions of the OODB paradigm and the deductive database paradigm are compatible; especially the matter of combining declarateness and the notion of an 'object' from the OODB paradigm ([Ullman91]). Although influential database experts refuted the combination of the deductive database model (declarateness) with the OODB model, many others proved this combination possible, with respect to their own interpretation of the concepts in question. Especially worth mentioning in the context of combining database paradigms is the language OORL [Rotterdam96] which combines in a declarative setting the relational model, notions from OODB systems and logical rules similar to the rules in deductive databases.

In the same era as OODB, from a different but maybe even more influential field of practice and research, notions relating to object technology originating from analysis and design made their advent in the conceptual world of information systems. The languages of analysis and design are tailored to describe at a high level of abstraction (conceptual level) the information structure of a system and its surroundings. For the languages supporting the object oriented notions in analysis and design in 1997 a standard language called UML has been accepted in industry. Although not formal, the models and languages of analysis and design contained appealing and well developed notions for information systems. Moreover the notions from analysis and design fit in and enrich nicely the concepts of the OO paradigms.

A very appealing feature of the languages of analysis and design is the ability to use sophisticated graphical schema techniques (UML, BOOCH, FUSION, OOSE, RDD, SYNTROPY, OMT, NIAM, EER etc.). Many of these schema techniques were gratefully adopted by implementation languages which included some practical database languages⁴ and (with some limitations) and programming environments⁵. Graphical syntax has also been introduced in scientific, and theoretically wellfounded database languages⁶. It is probably is impossible to imagine next generation information systems without means of graphical representation.

⁴e.g. 'Gemstone', and 'O2' (pronounced 'O-deux').
⁵strictly the graphics are not part of the programming languages, but are abbreviations of programming language expressions in the integrated development environments (IDE's). Examples of such IDE's are Visual Age (IBM), Visual Studio (Microsoft), Forte (SUN).
⁶e.g. IFO ([AbiteboulHull87]), GOOD and HQL ([AndriesEngels94]).
1.2. Object Oriented Information Systems

The most important ingredients of the Object Oriented paradigm include\(^7\) the following concepts:

- object and object identifier,
- complex value, type and class,
- ISA hierarchy, subtyping and inheritance,
- operations and methods
- encapsulation.

Related concepts that are usually not mentioned under the object technology label are:

- Declarativeness
- Graphical syntax

Concepts that are important for the use of the languages with object orientation are:

- partial specifications, identity and the extendibility principle

1.2.2 The Unified Modeling Language (UML)

The need for a uniform and consistent visual language in which to express the results of rather numerous object oriented (design and analysis) methodologies extent in the early 1990s became very evident. During that period the authors of three influential object oriented methodologies began an effort to unify their methods, when they were 'recruited' around 1995 by the Rational Software Company, a company that had developed a number of software development tools and practices. These authors were Grady Booch (author of the Booch method [Booch94]), Ivar Jacobson (initiator of the use case driven approach [JacobsonEtAliii92]), and James Rumbaugh (principle developer of the Object Modeling Technique OMT [RumbaughEtAliii91]). They released a first version (version 0.9) of the Unified Modeling Language UML in 1996. The effort was expanded to include other methodologists and a variety of companies including IBM, HP, and Microsoft, each of which contributed to the evolving standard. The standardization process resulted in the release of UML version 1.1 under the authority of the Object Management Group (OMG) standard organization in November 1997. UML has

\(^7\)We note here that many concepts of the object oriented database paradigm already existed in earlier paradigms. We do not imply that the notions mentioned here originated in this OODB way of looking at databases, but merely that they are present in and characteristic for the OODB paradigm
now grown into the de facto visual language for writing down information system models for most (if not all) methodologies and tools that use the concepts of object technology (object orientation). Currently UML has evolved to version 1.3 (June 1999) with only minor revisions.

The Unified Modeling Language (UML) is a standard modeling language for software. It is a language for visualizing, specifying, constructing and documenting the artifacts of a software intensive system. Basically UML enables a information system modeler to visualize its work in standardized diagrams. For example the characteristic icon to write down an object is a layered rectangle with an underlined name in the upper layer. Such an icon is just a graphical notation. It is syntax. The icons of UML also have an intended meaning, a semantics. Below we will list an overview of the syntax of UML and a brief description of the informally defined meaning of graphical UML terms. For a thorough treatment of the UML language and its semantics we refer to [FowlerScott00] and [WarmerKleppe99]. There also exists the public documentation set of UML that was delivered when UML was released as a standard by the Object Management Group [UML99], but it is suited as a reference only. Note that the semantics of UML as defined in the books and the standard consist of brief English (natural language) sentences. It is not a formal semantics. In the evolving process of this standard the definition has become more consistent, but still no formal semantics is planned for future releases (UML 2.0 plans to contain a number of major enhancements, again driven by practical use). The formal semantics of the core concepts of UML is the subject of this thesis.

UML provides developers with a vocabulary that includes three meta categories: things, relationships, and diagrams. There are four kinds of things:

- **structural things:** these are building blocks that can specify structure of the world. These are: class, active class, use case, interface, component, collaboration and node, object\(^8\), attribute\(^9\) in vocabulary and operation\(^10\).

- **Behavioral things:** these are building blocks that specify behavior of the world. These are: Interaction and state machine.

- Grouping things: These are containers that organize the world. These are: package, model, subsystem, framework.

- **Annotational thing:** This is a construct for adding arbitrary information in natural language. This is: note.

\(^8\)not classified as such by the three UML founders [JacobsonEtAli99]. They somehow only mention 'object' under the diagram meta category

\(^9\)also not listed in [JacobsonEtAli99]

\(^10\)also not listed in [JacobsonEtAli99] in vocabulary
Within the second meta category, relationships we find three building blocks:

- a dependency denotes a dependency between things
- a association denotes an association (or relation) between things
- a generalization denotes an inheritance or isa relation between two things

In the last meta category diagrams we find 9 types of graphical containers:

- use case
- class
- object
- sequence
- collaboration
- state chart
- activity
- component
- deployment

The remaining figures in this section show an overview of the graphical notation of UML
Chapter 1. The object oriented development practice

Figure 1.2: The relevant graphical constructs of UML
1.2.3 Object oriented modeling languages, database languages, and programming languages

The Unified Modeling language (UML) has originally been defined for object oriented modeling and design, and as such is the official standard language in that field. A development process is more than analysis and design. In the end we will also need programs and databases. Programs and databases still require other languages. The important glue between these languages, the glue that makes a straightforward transformation from specification from analysis and design to implementation possible, are common concepts of object orientation. This makes it possible to transform a design model in UML to database and programming languages, writing down all the information in the detail level as specified in the design model. In the implementation activity then implementation specific coding needs to be added (more detail) in order to obtain an operational system. In current development tools the uniformity of concepts is often good enough to do some part of the transformation automatically. Moreover automatic transformation for implementation back to design is also possible to some extent in order to make the design model consistent with the implementation model in the event that in an implementation language something has been changed that is also prevalent on the design level. And this all without a rigorous semantics.

We will spend some time on two kinds of implementation languages: object oriented programming languages and object oriented database languages. In practice the most commonly used object oriented programming languages are Smalltalk, C++ and Java. Especially Java, chronologically the latest developed from the three, uses concepts very similarly used as UML (probably because these languages were defined at the same time). This has an advantage that the transitions are relatively smooth. Therefore there are quite some tools that do automated code generation from UML to Java, and vice versa re-engineering code to UML diagrams. For the object oriented programming languages it holds that they are well accepted in practice.

Object oriented database languages also exist in many flavors, but in this field standardization attempts are also being made. The leading standard is the already mentioned ODMG standard ([CattellEtAlii00]). Although the value of the use of object oriented database languages is recognized, they are hardly used. Even in the context of object oriented software development, the main stream database systems are based on a different, namely relational information

---

11 An example is the roundtrip tool for the Rational Rose UML modeling and design suite and IBM Visual Age for Java. A note to make here is that the translation is performed based on an intermediate 'meta language' for storing object design information called XMI. This language is based on the XML standard and is a proposed standard for interchanging object meta data.

12 i.e. database systems that support object oriented database languages, and thus provide the possibility to handle their content as objects in the sense of object technology.
model. Even though the relational model is a very elegant one, the transition from concept to implementation we mentioned above needs to leap over a completely different way of looking at information. This leap is normally bridged by an \textit{object to relational mapping}. Even though such a mapping works out well in many cases, it is not based on a formal semantics of the object world (while the relational world has a more or less formal semantics). Such a formal model could be a good vehicle to define such a mapping.

There are some things to be said about some object oriented concepts that play an important role in the transformation. The object oriented concepts on static models has matured to an extent that the structure is common to most object oriented languages. Things get complicated when \textit{constraints} start to play a role. Nevertheless constraints are getting more important in recent trends in software development with the focus on so-called 'business objects'. From the technological perspective business objects are persistent objects that can be shared by all the information systems in an enterprise or organization. Business objects model core information entities for the business of an enterprise or organization. Such business objects can have complicated structure and behavior, but particularly they have complex constraints in the form of \textit{business rules} defined on them. These rules should be expressed by the modeling and design languages and should be forced upon the objects by the realizing database and programming languages.

In the modeling language UML there is a constraint language, called OCL ([WarmerKleppe99]), to express constraints, which enables a declarative way to write down constraints on objects. In the various object oriented databases there are constraint languages but there is not a constraint language in the proposed standard for object database languages by the ODMG. In the standard, constraints are to be forced by the operations on the objects, exactly like it is done in a programming language. This means that there is not a general mechanism that enforces the constraints, but every operation on the objects should make sure, in the end, that the constraints are satisfied. This practice imposes quite a gap between the design, in which we can declare the constraints, and the implementation, in which we need to enforce them. Most of the time custom mechanisms need to be defined to enforce the constraints. Alternatively, some constraints may be forced by the relational database system through an object to relational mapping. We see an important role in this area for a formal model for object orientation. The large complexity of the matter needs a thorough formalism for (better understanding of) the object oriented concepts in order to capture the constraints and perform the transformation. A lot can be gained if one could provide a general mechanism for solving this problem.
1.3 Object Oriented Software Development

A software development process defines who is doing what, when, and how to reach the goal of building or enhancing a software product. A very important tool in such a process is the language that formulates the accomplishments during the development process. This language is used in the descriptions of requirements, documents containing analysis of the universe of discourse for which one builds IT support, design models of the system to be built, and even the actual source code of the programs that denote in detail the working of parts of the system.

1.3.1 A very brief history of the OO software development process

In the late 1960s when software products became more and more complex, the need to present software architecture by means similar to engineering blueprints became apparent, in order to be able to communicate the content of the software and to guide the development of the software product to a successful end ([Jacobson85]). A significant milestone in the streamlining of software development processes was the issuance in 1976 by CCITT, the international body for standardization in the telecommunications field, of the Specification and Description Language (SDL) for the functional behavior of telecommunication systems. SDL was the first specialized object modeling language. Periodically updated it is still in use by a large number of developers. In the same context many other (non-specifically object technology based) languages with their companion methodologies were developed, of which the most influential is the language SA (Structured Analysis) with its methodology SADT (Structured Analysis and Design Technology) ([Ross77], [Ross85]).

Where the SA technology kept evolving in a steady way, the object technology inspired methodologies and accompanying languages became a real hype when object orientation became very popular (and more mature) in the late 1980s. Many object oriented development methodologies and design languages were introduced. Well known examples are the Booch method ([Booch94]), OOA/OOD (Object Oriented Analysis/Object Oriented Design) by Coad and Yourdan [CoadYourdan91a] [CoadYourdan91b], OMT (Object Modeling Technique) by Rumbaugh et alii ([RumbaughEtAliii91]), OL (Object Lifecycles) by Shlaer and Mellor ([ShlaerMellor88]), OOAD (Object Oriented Analysis and Design) by Martin and Odell ([MartinOdell92]), FUSION by Coleman et al. ([ColemanEtAliii94]), OOSE (Object Oriented Software Engineering) by Jacobson et al. ([JacobsonEtAliii92]), OOSD (Object Oriented System Development) by de Champeaux et al. ([deChampaeuxEtAliii93]), and MOSES by Henderson-Sellers and Edwards ([HendersonEdwards90]). The enormous amount of design languages provided a problem in communication of designs and automation of the development process. This fact initiated a considerable reduction in object
oriented design languages by the standardization effort of the leading forces in object oriented software development, resulting in a standard language for denoting information using concepts from object orientation. This language is the already mentioned the unified modeling language.

Now the industry has a standard object design language for use in object oriented software development, the trend to get to a unified process is also ongoing. This process is (of course) called the unified software development process, often abbreviated with 'UP' ([JacobsonEtAlii99]). We will use UP as a reference to give an overview on the object oriented software development process in the following section.

1.3.2 An overview of the OO Software Development Process

The aim of software development is to build a software system. A software development process is the set of activities needed to transform a user's need or requirement into a software system. The need or requirement that is to be transformed into a software system can vary from a simple processing demand of well understood entities to a request for sophisticated computations on various kinds of complex information to serve unintelligible processes. The sprouted software system, in the end, will have some purpose in the (more or less abstract) world in which the requirements make sense. In order to achieve this result an understanding of the world is needed, as well as an accurate description of the information that is processed. In order to get to this understanding and description (and eventually program code) a development process defines workflows and steps to gradually build the understanding (a model of the world). In this process languages are needed to write down the gained knowledge of the world. Here we get to the object technology in the object oriented development process. We will use languages that bear concepts of object orientation to write down the achievements in several stages of the process. These notions are important in the process because they are developed based on an intuition to talk about the worlds for which we build the software systems. The concepts are described in detail in the next chapter. The process itself also imposes some requirements on the languages as we shall see.

The general activities in software management are usually categorized by the following terms: requirements, analysis, design, implementation and test. Each of these activities (called core workflows in UP) are comprised of several tasks and have several deliverables. We give a short description the core workflows here:

- **Requirements.** The here goal is to find out what the purpose or the need of the users for the system is. A result of this activity is a list of requirements.
1.3. Object Oriented Software Development

- **Analysis.** Here we gain a conceptual understanding of the world in which the system shall live, and what its function shall be. We make a conceptual model of the system.

- **Design.** Here the conceptual model of the analysis phase is transformed to a technical description in terms that relate closer to what can be implemented with the current information technology. We expect from this activity a technical model and a systems architecture description.

- **Implementation.** Here we build the system with actual information technology. We code programs to be compiled and executed on operating systems and middleware platforms, and code database schemas and procedures for storage of information on database platforms. The result should be a fully operating information system.

- **Test.** Here the operational system is tested. We verify that its performance satisfies the requirements set, assess that it serves its purpose in its world and make sure that it does not malfunction technically. This activity will result in a number of defect descriptions and additional requirements for the software system.

In a software development process these activities are organized in phases and steps. Traditional software development processes like the waterfall process for software development typically organize these activities in a strict sequence, where at the end of each activity one aims to have a completely finished deliverable for the whole system. Deficiencies in either of the deliverables force one to step back to the activity in which this deliverable was constructed, aiming again to fully complete the deliverable. This process has some drawbacks, because in order to fully deliver the requirements or the conceptual model, one already needs full understanding of the world. In practice this is hardly ever the case.

The notions of object orientation enable an object oriented software development process to organize the activities differently. In an object oriented development processes the basic language constructs to build the deliverables are *objects*. These objects have a 'generality' that enables one to talk about the objects on an arbitrary abstract level. Objects can be referred to as meaningless labels, or as complex structures with sophisticated behavior. This feature enables one to let the objects that make up the deliverables evolve from an abstract indefinite version to a version that carries enough meat to realize the software system. Taking advantage of this the software development process organizes the activities in iterations. In an iteration artifacts of requirements, analysis, design, implementation and test evolve in parallel. In the early iterations most of the emphasis will

---

13Because we want to emphasize the notion introduced by OO software development we only mention this drawback, and do not go further into other fruits or drawbacks of the waterfall process.
be on requirements and analysis activities and only little on design, implementation and testing. In later iterations the most of the effort will go into design, implementation and testing, and less in requirements and analysis. In order to steer the software development process these iterations are organized in phases. For UP (but similar for the other OO development processes) these phases are:

- **Inception.** The primary goal of this phase is to establish the business case. After this phase one needs to be able to judge feasibility of the software system and validate its purpose.

- **Elaboration.** This phase focuses on do-ability. Here we need to establish the main part of the conceptual model and a basis for the architecture.

- **Construction.** Here we refine the conceptual and technical artifacts and do most of the building. This phase should deliver an initial system that operates and has all the main functions.

- **Transition.** Here the system and the artifacts are finalized and we validate its integral correctness.

### 1.4 Summary

In this chapter we described the practical context which is subject to scientific analysis and formalization in the coming chapters. The results presented in this thesis have their applications in precisely this context, and strive to contribute to the scientific fundamentals of the domains of 'information processing' and 'software development'.