Limits of Seamlessness in Object Oriented Software Development

Dorothea Beringer
GfAI, Gruppe für Angewandte Informatik AG
CH - 3037 Herrenschwanden
and
Swiss Federal Institute of Technology
Software Engineering Laboratory
EPFL-DI-LGL
CH - 1015 Lausanne
dorothea.beringer@di.epfl.ch

Abstract

Seamlessness is a promise often used to promote object orientation, and the use of object orientation in the different domains and in the different stages of the software development process has brought great improvements in seamlessness.

Where can we find seamlessness? And are there different kinds of seamlessness? In this paper we look at seamlessness starting with the different models found in large software developments. To give a common base for the discussion a model architecture is presented and different aspects of seamlessness are defined. We then show on a general level, where object orientation has already brought some seamlessness, where it will hopefully bring a higher degree of seamlessness in the future and where it cannot bring seamlessness due to restrictions inherent to the model architecture. The emphasis of the discussion is not to present a complete theoretical treatment but rather to help software developers to judge the many claims about seamlessness.

1 Introduction

In software development the most evident seams are found between analysis and design and between design and implementation. Such seams have been blamed for many delays and failures in software development when structured methods like Structured Analysis and Structured Design were used, but could never be bridged in a convincing way. Another seam is found between database models and application models. In applications where procedural languages are used together with relational databases, this last seam is very evident. The developer has to master two different paradigms and uses special tools like embedded SQL to bridge the gap. There are still other seams that are not so evident, e.g. between the external view of a user interface and its internal design, or between the conceptual data models and the logical database design.

With the penetration of object orientation into design and analysis as well as databases, the promise of seamlessness is often emphasized: using the object model in all of these areas would allow to have the same syntax and the same semantics in all models, thus avoiding any seams between the models or at least making the transition from one model into another very smooth.

Discussions about seamlessness often end in misunderstandings due to different presumptions about the development process. Therefore, before the degree and the places of seamlessness are discussed, we need to define our view of the development process. We do this by describing a model architecture. This will be the basis for the actual goal of the paper: to show on a general level where object orientation has already brought some seamlessness, where it will hopefully bring a higher degree of seamlessness in the future and where it cannot bring seamlessness, due to restrictions inherent to the model architecture. Chapter 2 presents the model architecture, in chapter 3 we define several aspects of seamlessness, chapter 4 then investigates seamlessness according to the criteria of chapter 2 and 3. Finally, some often heard misconceptions about seamlessness are briefly mentioned in chapter 5.

This paper is written from a developers point of view. Its purpose is to shed light onto some practical aspects encountered in rather large development projects, but it does neither aim at resolving the open questions nor at discussing in depth all aspects of the field.

2 The model architecture

The reason for presenting the frame of a model architecture here is that our discussion of seamlessness will be based on it.[1] The model architecture defines the most important models of a software system that are produced during its development. For each model it describes the purpose of the model, its syntax and its semantics. As part of the metamodel of the development process it only defines the most important results of the process. The model architecture shows the static side of the methods used (the models that are developed), but it does not show the process side of the methods (how the models are developed).

A single model architecture may serve as a base for several life cycle models. It only fixes which models are to be developed and what their static relationships are, but it does not make any constraints on the order in which the models are developed or on the development process itself. A model architecture thus enables sequential and iterative approaches. Prototyping is no contradiction to having a model architecture. Prototyping is an aid in the process to attain certain models (e.g. experimental prototyping for the models of the system architecture) and a prototype can even be part of a certain model (e.g. part of the user interface specification can be modelled by a prototype). Figure 1 shows the different abstraction levels of the model architecture that is the base for this paper.

Different methods with different kinds of models can be fitted into this frame. Many of the current OO-methods do not provide a complete model architecture that fills the whole frame presented here. Most of them cover the models of the detailed design and to some extent the system analysis. Other parts like the external specification of large user interfaces or the domain analysis are still very immature. Depending on the method used, each level can contain more than one model. As an example, the analysis model of OMT is made up of the object model, the dynamic model and the functional model [Rumbaugh 91].

In the following description of the different levels of the frame of a model architecture we concentrate on the purpose of the models. But we do not describe any possible syntax or semantics of them, which would then make up a concrete definition of a model architecture.

Models of the Domain and Problem Analysis:[2]

These models describe the universe of discourse on a very coarse level of granularity. The universe of discourse may contain one or more existent or planned computer systems but is in any case larger than the one defined by the system boundaries in system analysis.

The purpose of the domain analysis model is the following:

- To identify concepts and models which span multiple applications in a given domain.
- To describe the universe of discourse in the language and in the way of thinking of the

1. There are some more reasons to emphasize the model architecture and to define it explicitly, namely:
   - The model architecture defines the static view of methods in a very condensed way.
   - The same model architecture may be valid throughout projects with different life cycle models, whereas the process has to be adapted.
   - The model architecture also serves as the basis for quality assurance, project control and metrics.
   - Those models that are to be maintained throughout the lifespan of a system can be defined easily based on the model architecture.

2. D'Souza has proposed a similar frame of a model architecture in [D'Souza 93].

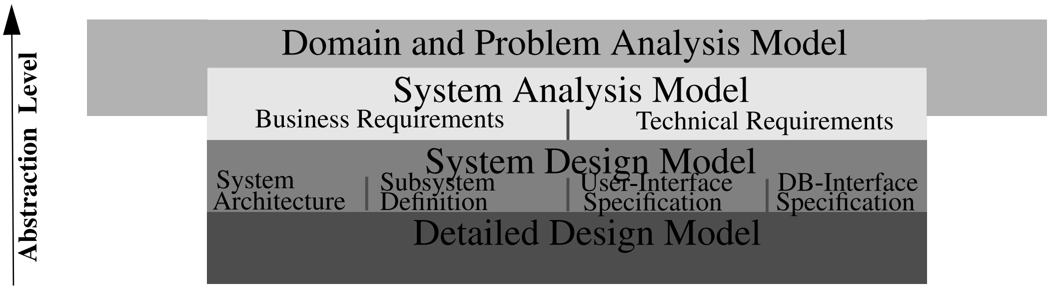

Figure 1

domain experts and users.

The purpose of the problem analysis model is:

- To describe one real-world problem in its environment.
- To model the problem in the language and in the way of thinking of the domain experts and users.

Models of the System Analysis:

The system analysis (also called conceptual modelling or functional requirements) focuses only on the part of a problem that will be implemented in a software system and differs from the problem analysis model mainly in the level of granularity and in the exact definition of the system boundaries. Before we can describe the outside view of the system, we need to know the exact system boundaries on a conceptual level. This must not to be mixed up with the user-interface specification, which is based on the specific technical constraints, goes into much more details, and is a model of the system design and not of the system analysis. The purpose of the system analysis model is:

- The system is modelled in the world of the user, in the way he is used to think. This may include business rules or business transactions that affect many different objects as well as objects that do not take any actions in the real world.
- The system is viewed and described from the outside. The overall features of the system must be determined. The main focus is on the external events acting on the system as a whole, together with their input and output characteristics.
- In order to support the definition of the business requirements (external view) it may be helpful to model the internal view on a very high level for structuring purposes. But the main focus lies on modelling what the business is (external view), not how it is solved by a certain system (internal view).

Examples of the semantics and syntax of system analysis models as defined above may be found in the requirements analysis of OOSE [Jacobson 92], the analysis model of Fusion [Fusion 93], or in Modern Structured Analysis [Yourdon 89].

Requirements concerning the technical constraints (also called non-functional requirements) also belong to the system analysis, but are not discussed here.

Models of the System Design:

While in the system analysis the business itself is modeled, the system design (also called system specification) defines the architectural requirements of one specific solution. The importance of the system design model depends on the complexity of the solution. For more complex systems the system design includes:

- Specification of the user interface: the purpose is to define exactly the “look and feel” of the interface from the users point of view. This is again an external view on the system as it does not describe how the user interface is designed internally.
- Specification of interfaces to other computer systems.
- Model of the system architecture and of the subsystems: the goal is the exact definition of the interfaces and responsibilities of the subsystems, which will then be designed independently.

The system design model thus shows how the requirements of the user are met by a certain software architecture. It is a technical model, using the language of the developer. Though some OO-methods stress the importance of the system design model (e.g. [Rumbaugh 91] and [Shlaer 93]), they do not provide modelling techniques for all parts of it.

Models of the Detailed Design:

Within the detailed design there are two kinds of models: the design model that is basically independent of a certain programming language and concentrates on the interface definition of the classes, and the implementation model that reflects the implementation and may be represented directly in the source code. The purpose is to define an object model that achieves as best as possible quality criteria like reusability, extendability, correctness and so on. The detailed design reflects the inside view of the system and incorporates all the details that are necessary to get a complete object model.

Not in every project models are needed at each level. There are many projects that start right away with the system analysis, the system design or even the detailed design. In these cases either the problem to be solved is very simple (the problem is implicitly defined in the minds of those, who solve it), or certain models are already defined in some other specifications, e.g. a hardware specification, a comprehensive and consistent problem statement or a company wide domain analysis.

Because the object oriented paradigm was first used in programming languages, there are many OO-methods that primarily address the detailed design. This, however, must not mislead us to forget the importance and purpose of the higher levels of abstraction.

3 Different kinds of seamlessness

For the further discussion it is useful to distinguish between different kinds of seamlessness, namely:

Seamlessness in the syntax

We speak of seamlessness in the syntax in different models or languages if they have the same set of symbols and if the rules how the symbols are put together to form a model or a program are equal. Of course, this does not mean that in every model every symbol must be used.

Seamlessness in the semantics

We speak about seamlessness in the semantics if the meaning and interpretation of the symbols is the same in the different models or languages. In the following examples we have seamlessness in the syntax, but no seamlessness in the semantics.

Example 1: The association symbol used in the analysis model may be the same as the one used in the design model. But the meaning of the symbol is different: in the analysis model it denotes a relationship that may be in both ways and may be a many-to-many relationship. In the design the symbol denotes a pointer, which is a one-way relationship and only points to one object.

Example 2: Depending on the context the symbol “class” may denote a type or a collection.

Seamlessness regarding the purpose of models

Every model has a certain purpose that determines its content and reflects a certain view of the system and a certain way of thinking. There are differences between modelling the business and modelling the software structure and there are differences between the outside view of a system, subsystem or object and its internal view. Two models are seamless in regard to their purpose, if they have the same purpose and reflect the same view of the system.

Seamlessness regarding the relation of two models

Even if two models differ in syntax, semantics and purpose, there may still be a weaker form of seamlessness: the seamlessness in the relation of two models. We say that two models are seamlessly related to each other if notions which were introduced in one model are represented in the other model in a very simple and straightforward manner [Jacobson 92]. This means that the relation between the components of different models and the transition from one model to another is well defined. Almost every model architecture claims this kind of seamlessness to a certain degree. But there are big differences in how smooth, precise and complete the transitions and relationships are defined.

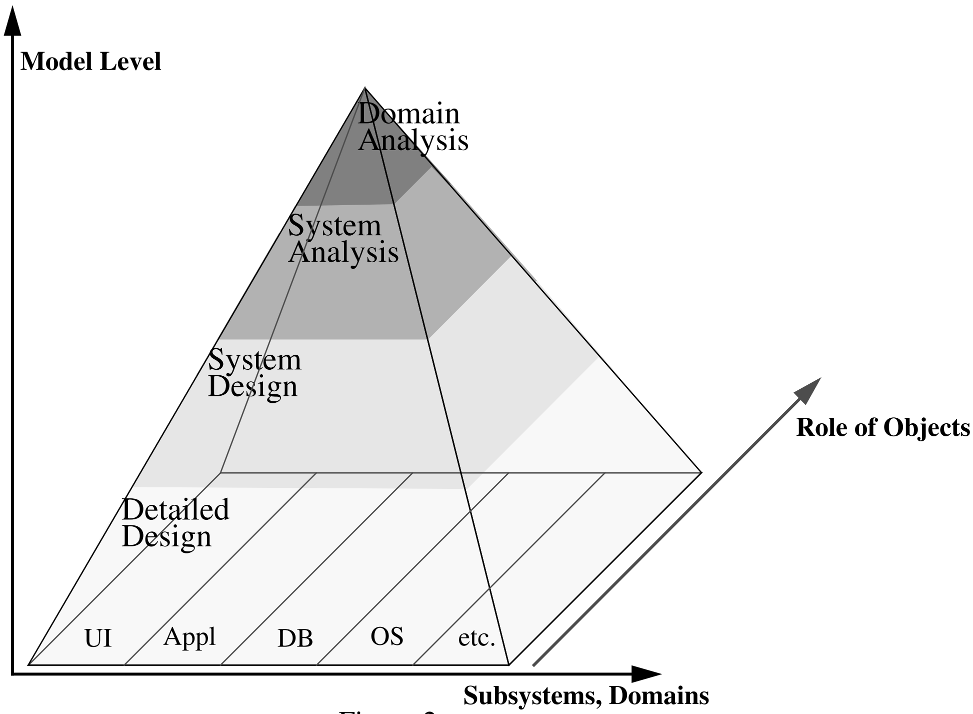

Figure 2

4 The three dimensions of seamlessness in the object oriented development

The above discussion of model architecture and seamlessness is independent of the paradigm used. It can be applied both for structured and object oriented methods and languages. In this paragraph we will now see what we can conclude for the object oriented software development. For this purpose, the different models are presented in a three-dimensional pyramid in figure 2. The first dimension is given by the different levels of the model architecture as described in chapter 2. The second dimension divides the object model into different subsystems or domains. Common subsystems are the user interface domain (UI), one or several application domains (Appl), a data base domain (DB) and the operating system domain (OS). The third dimension distinguishes the objects according to the role they have in the object model.

4.1 The different model levels

Because the different model levels have a different purpose by definition, there is clearly no seamlessness in purpose. What about the seamlessness in the semantics and syntax of these models as far as object oriented software development is concerned? This question is difficult to answer, because there is still no common agreement about the characteristics of object oriented analysis and design models, and because the characteristics used to determine if a programming language is object oriented or not do not apply directly on higher level models.

If we look at the system analysis models as proposed in [Fusion 93] and [Jacobson 92] it becomes clear, that they differ as well in semantics as in syntax from the models of the detailed design. They incorporate some, but not all of the semantics found in lower level models, and they add some more semantics to achieve purposes that are specific to their abstraction levels.

Even if object orientation cannot bring seamlessness regarding the purpose of the models, and has not yet brought the desired seamlessness in semantics and syntax into the first dimension of the model architecture, it has brought seamlessness into the relation between the models:

- As all the models are based on the object oriented paradigm, some characteristics, e.g. identifiable objects, classification, messaging and encapsulation are found in most models at the different levels. This is a great advantage compared to those methods that use completely different modeling paradigms for analysis and design (like Structured Analysis and Structured Design), and where neither the relation between the different syntax and semantics is obvious nor any methods that really support a clean transition are available.

- The way in which the elements of one model are related to the elements of another model can be well defined (see e.g. [Fusion 93], [Jacobson 93] and [OMT 91]), though no automatic transformation can be provided.

In the first dimension the seamlessness concerning the relationship is the most important one and here object orientation has brought a great breakthrough. Developers need different models to express different views. But the relations between these views must be clear and together these models should give a complete description of the system through different levels of abstraction. The goal for the future should be to improve the methods in supporting this kind of seamlessness, and not to enforce seamlessness in syntax and semantics to a degree where the different purposes of the model levels get lost.

4.2 The different domains

Though the different object oriented programming languages (OPLs) all incorporate the characteristics of object oriented languages, they differ in the details of the underlying object model (e.g. multiple inheritance in C++, class variables and methods in Smalltalk, pre- and postconditions in Eiffel) and in their syntax. Of course there are interfaces to bridge the seam between subsystems designed and implemented in different languages, but the seam remains.

Another source of seam in syntax and semantics are different libraries and different frameworks for the different subsystems. If we use different libraries for the user interface, for the application domains and for the database subsystem, these libraries do not necessarily match in the common parts. Certain user interface management systems (UIMS) and object oriented database management systems (ODBMS) even force the developer to use their underlying object model and their library for all communication. Therefore a mismatch cannot be avoided. The only help here is a very careful system design and a wise choice of the tools and libraries to be used.

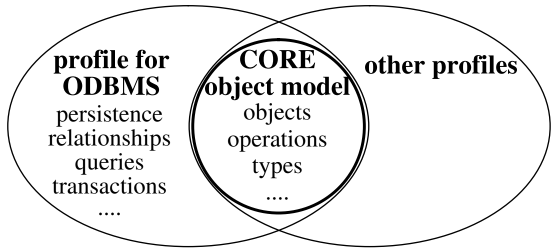

Figure 3

The big breakthrough object orientation has brought into the second dimension concerns the seamlessness between applications and database management systems (DBMS). One big concern in classical software development is the seam between the procedural languages and the relational calculus of the DBMS. By using object oriented languages together with object oriented database systems it is hoped to avoid this seam. As far as the underlying paradigm is concerned, the seam has indeed been closed and there is no more change of paradigm between relational and procedural models. But the seamlessness between the applications and the ODBMS is not yet perfect. Up to now the meta models underlying the different ODBMS differed from each other and they differed also from the meta model underlying the OPLs. Some of the latter differences are due to the fact that ODBMS fulfil requirements that are not found in most OPLs, e.g.:

- Each object in the database needs some kind of handle that is persistent over time and can be retrieved when the application is started again.
- The objects in the database may be used concurrently by several applications.
- Most OPLs do not differentiate between the type definition of an object and the collections of objects, a distinction necessary in data bases.

To overcome these differences in the object models and to allow the programmer to experience as much seamlessness as possible across different ODBMS and with certain programming languages, the Object Database Management Group (ODMG) has defined the Object DBMS Standard, which contains a common object model. The members of the ODMG will sell ODBMS that are based on this model [ODMG 93]. The Object DBMS Standard includes the core components of the OMG Object Model and will be proposed to become an accepted profile in the OMG architecture as illustrated in figure 3.

4.3 The role of objects

Within each model we find objects with different roles. Some objects provide low level services, e.g. mathematical functions or collection classes, others reflect the essence of a subsystem. Some objects act as servers, just being there to help other objects fulfil their responsibilities, while others have the role of interfaces between some groups of objects or are controllers that incorporate control mechanisms. In the design of a model it is often helpful to distinguish between the different roles of objects. Some methods even use different syntax for objects with different roles (e.g. entity objects, control objects and interface objects in the analysis and design models in [Jacobson 92]).

There are several schemes that may be used to distinguish between the roles of an object, e.g.:

- Classification according to the behavior and the responsibility of an object within a group of objects (see also [Wirfs 93]):
  - Server objects: objects that do not use other objects but only provide services.
  - Controller objects: objects whose main responsibility is controlling some higher level aspects.
  - Coordinator objects: objects that help to reduce the coupling.
  - Interface objects: objects that are used as an entrance point for accessing a group of objects from the outside.

- Classification according to the essence of an object for the system:
  - Essential objects: these objects reflect the essence of a certain subsystem.
  - Architectural objects: these objects complete the object model of a subsystem to get a higher level design.
  - Low-level objects: objects typically found in libraries, e.g. building blocks for lists, mathematical functions and collection classes.

Though the role of the objects may differ, they have the same syntax and the same semantics in the design. The quality metrics are the same for all different objects as far as their role is concerned. Therefore, if object oriented techniques are used, there is a high level of seamlessness in syntax and semantics in the third dimension of our system pyramid.

5 Misconceptions about seamlessness

5.1 Mismatch between abstraction levels and roles of objects

For the classifications of the roles of objects according to their essence for the system, there are also other terms in use, namely: analysis objects, design objects and implementation objects. Analysis objects in the design and implementation models are those objects that are derived from some objects already present in the analysis model. They represent the key concepts of the problem domain. Essential objects from other subsystems (e.g. objects for the user interface) and architectural objects are called design objects. The implementation objects are those objects that do not have any predecessors in higher level models (see also [Meyer 93]). Figure 4 shows this classification for the implementation model.

The misconception comes in when this classification is used as an argument for the seamlessness in syntax, semantics and purpose of the analysis, design and implementation model. If we call the part of the implementation model that is made up of analysis objects the analysis model, that analysis model corresponds no longer to the analysis model defined in our model architecture. It is just a first cut of the implementation model. In this case there is only one level of abstraction, and only one model being developed during the whole development process, namely the implementation model[3]. As a consequence the models of analysis, system design and detailed design do not address different purposes, different views (external and internal views) and different experts (domain, user and programming experts), but denote only what kind of objects are added to the model.

3.This view is promoted by those CASE-Tools that either advocate code generation out of analysis models or allow only one abstraction level of the model architecture.

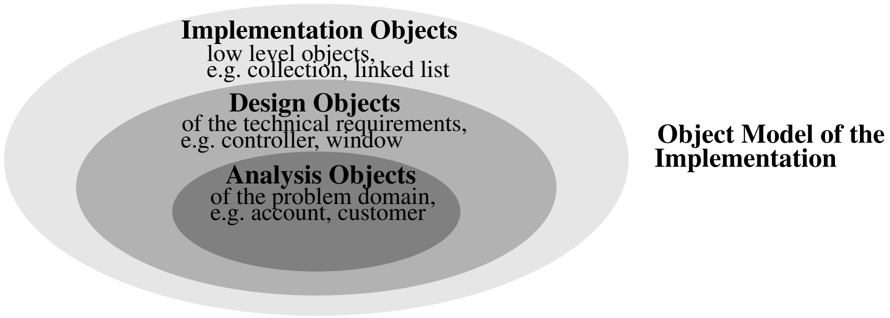

Figure 4

Thus if we do not distinguish clearly between the abstraction levels of models and the role of objects, we are likely to run into some of the following problems:

- We degrade analysis to be the first draft of design. This means that we just start with design and use the quality criteria and semantics of the detailed design already in the first steps of analysis.
- The focus in the analysis model is on describing the real world according to the purposes and criteria of the analysis model. When the resulting objects are just carried over into the design, this can result in a bad design, because the different purpose and semantics of the analysis and design models also imply different quality criteria.
- The essential objects of the domains other than the problem domain are often handled as less important add-ons, thus neglecting a careful system design.

As far as seamlessness is concerned, considering the analysis model as being just a part of the implementation model results in a very high degree of seamlessness, but it is a seamlessness in the third dimension (the role of objects) of our system pyramid and not in the first one (the abstraction level of models).

5.2 Object models based on an extended entity relationship model

One popular way of developing an object model is the data driven approach. In a first step, some kind of extended entity relationship model (ERM) is developed. In a second step, operations and messages are added. Though the model gets refined and completed by further classes, the entities and relationships discovered and modelled in the first step remain more or less the same during analysis, design and implementation. The entities modelled by the criteria of data modelling are then the objects that should satisfy encapsulation, high cohesion, low coupling and a good subsystem structure. There seems to be a high degree of seamlessness, but this seamlessness is primarily one in the syntax. The syntax of an ERM is a subset of the syntax of the object model of the analysis and the design. But what about the semantics of the models?

- ERMs have a flat structure. The unit of encapsulation is a very small one: entities. These entities can be put into aggregations, but the aggregations are always viewed as a loose collection of entities and their inside is known to the rest of the model.
- Normalization is a very important notion in ER-modelling. But normalization makes it necessary to handle structured or polyvalent attributes as separate entities (e.g. in [Coad 91]). In an object model, however, objects may have a complex structure.
- ERMs do not support information hiding. Every entity and everything inside an entity is visible to any other entity in the model. The interface to the entity is defined by its internal structure. Visibility criteria do not play any role in the modelling of an ERM. In an object model, however, there is a clear distinction between the interface and the internals of a class and it should also be clearly stated, to which other classes a class is visible.
- Attributes and operations are distinguished from the beginning of the modelling process.

But analysis should only answer the question which features and responsibilities a certain class or group of classes has, and not if they are provided as data or as operations. This design and implementation decisions should remain the responsibility of the detailed design, and not be decided upon in the first steps of analysis.

- The quality of an object model can be measured by the degree of cohesion inside the classes and the coupling between the classes. In order to achieve good quality the interface of a class, i.e. its operations, must be determined according to these criteria. If classes are determined only based on the static data that resides in them and all other features are added later on, either the model has to be thoroughly redone (i.e. transformed into a totally new model), thus sacrificing the claimed seamlessness, or the object model will be of a bad quality.[4]
- Large systems are divided in subsystems according to the criteria of high cohesion and low coupling. An ERM itself does not support the notion of subsystems. If they are introduced later on, either the subsystems cannot satisfy the mentioned criteria, or the model must be thoroughly revised, thus sacrificing the seamless extension of an ERM into an object model.

So there is seamlessness in the syntax, but not in the semantics between an ERM and an object model, and there is certainly no seamlessness in purpose between an ERM of the analysis and an object model of the design.

If we nevertheless want to use the ERM in the analysis, there are two possibilities: First, we may enforce a seamless extension of the ERM into the object model by just adding to the data model of the analysis operations and further classes as necessary for the design and the implementation. Then the resulting object model will have a high degree of coupling between the classes, and size, complexity and visibility of the classes are not minimized.

The second possibility is to sacrifice seamlessness in order to get a really good design. The data driven approach is used in analysis (with or without defining operations for the classes). The object model of the design uses the analysis model only as a source of well defined information but is made according to the quality criteria of the design, thus introducing a clear seam between the two models (though they may have the same syntax):

- ERMs are a good modelling technique to investigate the data especially in information engineering and to model part of the system requirements. We can use ERMs as part of the problem analysis in order to describe the data-side of the requirements and with the goal of modelling the problem from a user's perspective.
- After the analysis and after having decided upon the overall system architecture, we can use the ERM as an input to the object models of the application domains. However the object model must be built according to the criteria of good object design and does not necessarily resemble the original ERM from analysis. The developer then makes the design with the goal to get easily maintainable, extendable and reusable classes.

5.3 "The use of OPLs has made all higher level models superfluous"

In the times of assembler programming the gap between the description of an algorithm in structured English and its code was quite large. When the first third generation languages appeared, this gap nearly vanished, because these languages were just as expressive as structured English. Still the overall program structure, especially the interfaces to the different modules, had to be described separately. With more advanced programming languages like Modula2, Ada or Eiffel, even this could be done just as nicely in the programming language. As a matter of fact, there are many well documented systems around that are structured and documented directly in the source code, using the features of the programming languages (and comments, of course). Having now object orientation, and thus languages

4. A comparison of the quality of an object model developed according to the data driven approach and the quality of one developed according to other approaches may be found in [Sharble 93]

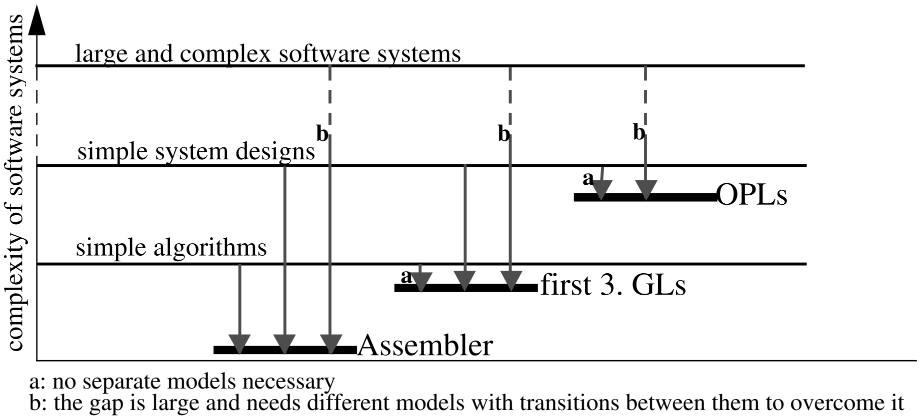

Figure 5

that allow to implement directly all features of an object oriented system, are there still any higher level models necessary? If we look at certain systems that were implemented some years ago and that needed several transitions from the problem statement down to the assembler coding, we are tempted to say no, because their problem definitions are simple enough to be described directly in modern OO-languages, obtaining total seamlessness by having only one model. We get the same picture if we look into books on object oriented programming where many examples with a textual problem statement can be directly implemented in a high level OO-language.

Nevertheless there are two reasons, why still several model levels are necessary and why the problem definition and the system design in general cannot be directly included into the source code:

- Complexity of the problems to be solved: As the programming languages evolved the systems to be built also became more complex. The system architecture and the user interactions are much more complex today. The systems are larger, different software systems need to be connected and the use of off-the-shelf components like DMBS, UIMS and networking software became much more important. For complex systems it is not possible to model all the different views in one model. The danger to get lost in a complex project is too high.
- Abstraction levels, ways of thinking: Even if we have very mighty programming languages (even mightier ones than todays OPLs), there will still be the difference in the purpose between designing a system by system experts and analyzing the problem domain by domain experts. There must be a transition from the analysis model to the design model, because they represent different ways of thinking, are approved by different kinds of people and are created according to different quality criteria. As long as we want to reflect the business world as directly as possible, we need object models that are able to reflect the way of thinking of the business world. Thus a transition form the object model of the business to the object model of the implementation is unavoidable (see also [Palmer 93]).

6 Conclusions

In this paper we have discussed object oriented software development with respect to seamlessness. In order to lay a common foundation, we have first defined a model architecture consisting of different abstraction levels, and a classification of seamlessness according to the seam in the syntax, the semantics, the purpose, and the seam in the relationship. In a second step we have examined seamlessness with respect to three dimensions, namely the different abstraction levels, the different subsystems or domains found in a system, and the different roles of objects in a model.

As far as the different roles of objects in a model are concerned, seamlessness in syntax and semantics is already available today.

For the different domains, object orientation can bring seamlessness in syntax and semantics at least to a certain degree, too. However, due to differences in syntax and semantics of different OPLs, ODBMS, UIMS and libraries, the maximum degree of seamlessness has not yet been reached. As soon as a standard object model will be generally acknowledged and incorporated, the situation will be much better. Corresponding efforts of the OMG concerning a core object model and of the ODMG concerning database systems are on the way.

Another advantage of the object oriented software development is the seamlessness in the relationship between the models of different abstraction levels (the first dimension in the system pyramid). The fact that all models are based on the same object oriented paradigm makes well defined transitions from one abstraction level to another possible. But because purpose and content of the different abstraction levels are different by definition, there will always remain a seam in purpose between them. The question to what extent seamlessness in semantics between different abstraction levels can be reached remains open because todays methods do not yet cover all areas of the model architecture, and because new methods are still emerging.

7 References

[Booch 91] Object Oriented Design, G. Booch, Benjamin/Cummings, 1991

[Coad 91] Object-Oriented Analysis, P. Coad and E. Yourdon, Prentice-Hall, 1991

[D'Souza 93] Comparing OO Analysis and Design Methods, D. D'Souza, Proceedings of the OOP93 in Muenchen

[Fusion 93] Object-Oriented Development: The Fusion Method, D. Coleman et al., HP Laboratories, 1993

[Jacobson 92] Object-Oriented Software Engineering, I. Jacobson, Addison Wesley, 1992

[Meyer 93] Design by Contract, B. Meyer and J. M. Nerson, Proceedings of the Summer School of EDBT in Leysin, 1993

[ODMG 93] The Object DBMS Standard, Th. Atwood, Object Magazine, Sept.-Oct. 1993; The ODMG Object Model, M. Loomis, JOOP June 1993

[OMG 92] The OMG Object Model, Object Management Architecture Guide V2, OMG, 1992

[Palmer 93] Seamlessness or shamelessness?, J. Palmer, Object Magazine, Sept.-Oct. 1993

[Rumbaugh 91] Object Oriented Modelling and Design, J. Rumbaugh et al., Prentice-Hall, 1991

[Sharble 93] The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods, R. Sharble and S. Cohen, ACM Software Engineering Notes, April 1993

[Shlaer 93] A Deeper Look at the Transition from Analysis to Design, S. Shlaer and S. Mellor, JOOP 5(9), 1993

[Wirfs 90] Designing Object-Oriented Software, R. Wirfs-Brock et al., Prentice-Hall, 1990

[Wirfs 93] R. Wirfs-Brock, talk given at the CHOOSE-Forum (Zuerich, Switzerland, 1993)

[Yourdon 89] Modern Structured Analysis, E. Yourdon, Prentice-Hall, 1989