Object oriented development with BSDM

Lambregts, Paul J.T.

Award date:
1990

OBJECT ORIENTED DEVELOPMENT WITH BSDM

August 21st, 1990

P.J.T. Lambregts (Student T.U.E.)

IBM, EBSAT
Uithoorn, The Netherlands

Abstract

In this report is described:

- The IBM system development method BSDM with an evaluation.
- The advantages and disadvantages of developing applications based on the object oriented paradigm.
- The initial development process BSDM - OOD/OOP. This development process comprises BSDM, object oriented design and object oriented programming.

The main conclusion is that the integration of BSDM and the object oriented paradigm is possible and most promising.

Table of Contents

Introduction

In 1984 I started studying 'Industrial engineering and management science' at the Eindhoven University of Technology (TUE). Within the study, my interest was growing towards how information technology could as best as possible support and improve the functioning of organisations.

On 11th September 1989 I started my subject for final project at IBM in Uithoorn, within the department EBSAT. When I started, my task was vaguely defined. There were only some ideas about the direction of the assignment. During the first weeks the objectives of my assignment description concentrated on the IBM development method BSDM, the object oriented paradigm and how these can be integrated.

This report contains the results from my assignment.

Overview of the report

- Chapter 1 deals with the IBM organisation. It contains a description of how IBM worldwide is geographically divided and the three important principles of IBM.
- Chapter 2 contains a short description of the department EBSAT where I worked with its objectives.
- Chapter 3 contains the assignment description and the reasons why EBSAT is interested in the results of this assignment.
- In chapter 4 an overview is given of the IBM system development method BSDM with an evaluation of this method.
- In chapter 5 an overview is given of the object oriented paradigm. This chapter also contains an overview of the advantages and disadvantages of developing applications based on the object oriented paradigm.
- Chapter 6 contains the draft development process BSDM - OOD/OOP. It describes the steps that should be taken when developing applications based on BSDM and the object oriented paradigm. BSDM - OOD/OOP is not a 'ready to use' development method, but should be refined further.
- Chapter 7 evaluates BSDM - OOD/OOP, describing the strengths, the weaknesses, attention areas and whether the advantages and disadvantages of object oriented system development in general also applies to BSDM - OOD/OOP.
- Chapter 8 describes the main conclusion and the recommendations.

The summary of this report can be found immediately after this introduction.

The following appendices belong to this report:

- Appendix A gives an overview of BSDM data modelling.
- Appendix B gives an overview of BSDM process modelling.
- Appendix C gives an overview of BSDM Architecture.
- Appendix D contains the demands for the case study and gives the description of the case study.
- Appendix E contains the BSDM data model of the case study.
- Appendix F contains the BSDM process model of the case study.
- Appendix G gives an overview of Smalltalk/VPM.
- Appendix H gives some examples of Smalltalk/VPM system classes.
- Appendix I demonstrates BSDM - OOD/OOP for the case study.

During my assignment I made also another report that contains the results of my literature research about object oriented development {Lit. 18. on page 49}.

Word of thank

During my assignment there were some people who helped me doing my work. I like to thank them for their help. Jan de Lint (IBM) who made it possible for me to have my assignment within IBM. I like to thank my first mentor Ir.J.Trienekens (TUE) who was very critical in everything I did and really helped me doing my work. My second mentor Dr.Ir.P.Bots from Delft University of Technology (TUD), who was willing to be the very critical second mentor. He helped me very much. My thanks are also for my third mentor Prof.Dr.T.M.A.Bemelmans (TUE). My special thanks are for Rolf Moerman, Paul Achterberg and Eugene van Heusden, from whom I learned BSDM. My thanks are also for Rick Kooke, the (ex)student with whom I had several discussions about Smalltalk.
I also want to thank other IBM'ers with who I had several discussions about different topics. I learned a lot from them and also they helped me to do my work: Henk Jurriaans, Andrew Norton, Ashok Bhanaut, Doede Droog, Ghica H.v. Emde Boas, Herman van Goolen, Adriaan van Maarleveld, Johan Winter and (contractor) Cees van de Langerijt.

I am very glad that I had my assignment for final project within IBM, within EBSAT, and about a subject which is really very interesting.

Paul Lambregts

Summary

This report is the result of an assignment within IBM, concerning the development of object oriented applications, based on the IBM system development method BSDM. I demonstrate that BSDM is most promising for building applications based on the object oriented paradigm.

Problem area

BSDM (Business Systems Development Method) is the IBM systems development method, that is mandatory for IBM I/S in Europe, since December 1987.
Currently, there is another approach for building information systems which is of growing interest worldwide in literature and practice: the object oriented paradigm. Until now, this paradigm might improve the application development process in the programming phase.
Although there are some experiments within IBM with object oriented programming, it is unclear how BSDM and the object oriented paradigm relate to each other. Questions in this area are:

- Do BSDM and the object oriented paradigm exclude each other ?
- Can BSDM and the object oriented paradigm be integrated, using the best of both worlds?

BSDM

This method consists of:

1. Business Modelling.
   The objective of business modelling is to model the needed data and information processes, on the level of WHAT the business does (not HOW).
2. Systems Architecture.
   Provides a high-level framework within which multiple systems can be coherently planned and developed.
3. Requirements.
   Detailed needs are identified and defined.
4. Design.
   Based on the business model and the requirements, one or more designs for implementation are made.

The programming phase is not part of BSDM.

Strengths

- A business model will certainly give you more insight into the business.
- Business models are intended to model the nature of the business, not the business implementation.
- Business models are relatively stable.
- Business models have the potential to be used as reference models, that have to be adjusted for specific organisations.
- Data and processes are integrated very well.

- Non-functional demands are identified and defined.

Weaknesses

- It is not clear for which types of applications or for which types of organisations BSDM has proved to be less or more successfully. IBM should document this.
- Although making a business model will give you more insight into the business that you are modelling, it is not used for business improvement.
- Because business models are not models of the current business situation, users may find it difficult to understand the model. Especially when they were not involved in the modelling process (e.g. illness, new employee).
- There is no BSDM workbench available. A BSDM workbench is badly needed because of the enormous amount of documentation.

Recommendations

- BSDM should be extended with business improvement, based on the business model. For example, recording the current business implementation, the improved business implementation, and the changes that should be made.
- IBM should document for which situations BSDM has proven to be less or more successfully.

The object oriented paradigm

The object oriented paradigm is a way of structuring, that arises from a set of programming concepts and techniques. Within the object oriented paradigm, objects play a central role. An object corresponds with, or represents things in the real world. An object consists of data (information) and methods (procedures, code, routines) which surrounds the data. The methods are the only routines authorized to access the data of an object. Objects communicate with each other, using messages. A message is a request from an object for another object to carry out one of its methods.

Advantages

Advantages of ′object oriented systems development′ can be found in the areas of maintenance, extendibility, modifiability, productivity, quality, user friendliness, seamlessness and recording dynamic aspects.

Disadvantages

Disadvantages are the learning curve, the lack of a well defined and widely accepted methodology for developing applications, the lack of standards, non modular hierarchies, problems with combining object hierarchies, the lack of object-management systems and the often slowly running applications.

Conclusions

Although there are some important disadvantages for which solutions are needed, the object oriented paradigm is most promising in improving the development process. It certainly will influence the way applications are built. But it will take some years.
The extent of improvements largely depends on the way we handle it. The paradigm alone will not do it! But to improve the development process, we need a well defined methodology for analysis, design and programming, based on one clear set of (design) principles. The experience which will be gained must be used to improve the existing methodologies.

Not everything changes. The object paradigm does not make all the experiences and existing ideas worthless. A lot of things remains the same: Supporting the business, user involvement, education, etc.
Large organisations will not introduce object oriented development within a short time. Important reasons for this are: Lack of experience and maturity, the needed education effort and the problem of attitude of the developers who are process oriented.
Although learning the concepts is not difficult, the application of the concepts is. To enlighten this education problem for organisations, and to improve (technical) skills (as a country), universities should give education about the object oriented paradigm, and (interested) students should learn to build (simple) object oriented programs.

BSDM - Object oriented paradigm

During my assignment, I integrated BSDM with the object oriented paradigm, resulting in the BSDM - OOD/OOP development process. BSDM - OOD/OOP must be seen as a first attempt to combine BSDM with the object oriented paradigm. BSDM and BSDM - OOD/OOP are shown in Figure 1.

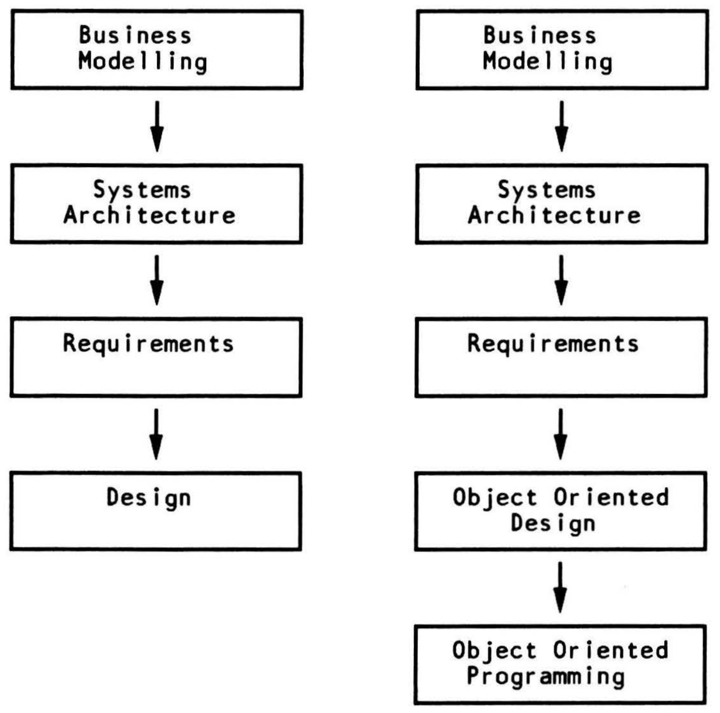


Figure 1. BSDM and BSDM - OOD/OOP: BSDM is depicted left in the figure and BSDM - OOD/OOP is depicted right in the figure. In this first attempt, Systems Architecture, Business Modelling and Requirements are for both processes the same.

Within BSDM and BSDM - OOD/OOP, the parts systems architecture, business modelling and requirements are the same.
Based on the business model and the requirements, one or more object oriented designs will be made which are independent of any specific language. Object oriented design consists of seven steps:

Step 1: Identify dependency structures
Step 2: Define inheritance structures
Step 3: Transform attributes into instance variables
Step 4: Identify the methods

Step 5: Establish relationships between classes
Step 6: Build class hierarchy
Step 7: Reuse predefined classes

During these steps, most of the information that is recorded during the previous BSDM phases is used.
The phase following object oriented design is object oriented programming, that consists of two steps:

Step 1: Implementation
Step 2: System test

Strengths

- Most of the information that is recorded during the BSDM phases architecture, business modelling and requirements can be used for developing object oriented applications.
- Current modelling effort can be used for developing applications based on the object oriented approach.

Weaknesses

- Draft development process, it is a first attempt.
- It is only used for the case study.

Conclusions

Theintegration of BSDM and the object oriented paradigm is possible and most promising.

Recommendation

Improve BSDM - OOD/OOP !

Chapter 1: IBM

IBM (International Business Machines) is in the business of applying (advanced) information technology to help solve the problems of business, government, science, space exploration, defense, education and other areas of human activity. IBM offers customers solutions that incorporate information processing systems, software, communication systems and other products and services to address specific needs. These solutions are provided by IBM's worldwide marketing organisations, as well as through the company's business partners, including authorized dealers and remarketers {Lit. 16. on page 49}.

Worldwide, the IBM Corporation is divided geographically into:

- U.S.A.
- EMEA (Europe, Middle East and Africa)
- Canada, middle and south America
- ASIA

Principles

Thomas J.Watson, SR, founded IBM in 1914. Like any ambitious entrepreneur, he wanted his company to be financially successful, but he also wanted it to reflect some of his personal values. These values became the foundation for his new company. Watson's tenets, which was reaffirmed by his son, Thomas Watson, Jr., in 1956, when he became IBM's second Chief Executive Officer (CEO) are {Lit. 22. on page 49}:

1. The individual must be respected.
2. The customer must be given the best possible service.
3. Excellence and superior performance must be pursued.

Since IBM opened its doors, the company has had a full-employment tradition, which sometimes took careful planning and commitment.
IBM's practice is to promote people from within. But almost everyone, who ever held a managerial or an executive position at IBM, started as a marketing trainee and moved up to the higher ranks. In fifty years, no person employed on a regular basis has lost as much as one hour of working time because of a layoff {Lit. 25. on page 49}.

Announcements

IBM makes essential investments in research, development and engineering. One of the areas of interest is the improvement of methods and tools for system development. Some important results of IBM's efforts in this area were recently announced {Lit. 16. on page 49}:

- OfficeVision, a family of easy-to-use applications for document preparation, filing, electronic mail and calendars. This first major Systems Application Architecture (SAA) offering, provides consistent, integrated office functions across IBM's product line.

- AD/Cycle, a series of products, tools and services and programming languages developed by IBM and their business partners. AD/Cycle uses computer automation to improve the productivity and quality of customer's application development efforts.
- CIM Advantage, a package of hardware and software products that provides a comprehensive approach to computer-integrated manufacturing. CIM advantage includes several solutions IBM developed for use in its own manufacturing plants.

The market

For almost a quarter of a century, growth in the information industry averaged nearly 15 percent each year, driven largely by advances in technology that dramatically reduced the cost of computing. Over the past three years, however, industry growth has slowed to between 8 and 10 percent worldwide, with the fastest growth coming outside the United States.
At the same time, the industry is undergoing fundamental change. Opportunities are shifting in favor of software and services, (services include maintenance, cabling, education and installation) and toward small computers, open systems and standards. And customers continue to raise the level of their expectations. Rapid changes in technology and advances in manufacturing productivity have created overcapacity in both physical and human resources. Intense competition is increasing pressure on prices, margins and profitability {Lit. 16. on page 49}.

Prospects for the future

IBM is planning for growth in all major geographies in 1990, led by the Asia Pacific area, followed by Europe and the United States. IBM's confidence is based on two fundamentals {Lit. 16. on page 49}:

1. The first is the opportunity in the potential for new customers and new applications. The great majority of companies worldwide do not use modern information products. And even a larger majority of employees who could use some type of workstation do not have them.
2. The second fundamental is the confidence that IBM is well positioned to participate fully in the opportunity they see ahead.

EMEA

EMEA (Europe, Middle East and Africa) is divided into the major countries:

- France
- England
- Germany
- Italy

and the multi country units:

- North (NU): Scandinavie
- Central (CU): The Netherlands, Belgium, Luxembourg, Austria
- South (SU): Greece, Egypt

The major countries are not part of the multi country units. The EMEA headquarters (EHQ) are in Paris.

Within 'IBM The Netherlands', the largest division is Marketing & Support. Other divisions are Intops Amsterdam, Intops Uithoorn, and the International Network Center in Zoetermeer.

Within IBM, there are national and international development centres. The national IS centres develop applications for their own country. Each country has its own IS centre.

International development centres develop (common) applications for EMEA. Some international development centres are:

- MISC, which builds applications for marketing.
- FINANCE, which builds financial applications.

Chapter 2: EBSAT

EBSAT means EMEA (Europe, Middle East and Africa) Business Systems Architecture and Technology. EBSAT advices EHQ about the strategic directions for systems development for the international development centres. Based on these advices, EHQ decides about the directions, which have to be followed by the international development centres.

EBSAT is funded by the national IS centres within EMEA. Although the advices are meant for the international centres, the national centres want to have more and more direction from EBSAT.

EBSAT Mission

EBSAT is an aid and support to EHQ I/S Management and EMEA Unit I/S Management, under direct control of the I/S Group Director in EHQ.
Its major functions are to:

- Establish, maintain and promote consistent architecture in the area of:
  - Data and business systems
  - Technical environment (Support Systems, technical support)
  - Data transfer
- Synthesize Information Processing product requirements, formalize justification and address agreed requirements to Product development. Promote usage of Standard Products
- Perform studies on Products/Methods. Based upon results and management decision, recommend utilization in Centers/Units.
- Assist I/S Management to achieve I/S functional excellence and demonstrate leadership in Information Processing.

Chapter 3: The assignment

Problem area

BSDM is the IBM development method for building information systems. This development method is mandatory for I/S within IBM - EMEA (Europe, Middle East and Africa). Currently, there is another approach for building applications which is of growing interest worldwide: the object oriented paradigm. The object oriented paradigm is getting more and more attention in literature and practice. EBSAT is interested in the improvements that the object oriented paradigm might offer to systems development.

For IBM, questions in this area are:

- What is the object oriented approach ?
- What are the advantages and disadvantages of the object oriented approach for system development ?
- Should we put effort in this approach ?
- Do BSDM and the object oriented paradigm exclude each other ?

EBSAT does not have the answers to these and other questions. Because they advice the European headquarters (EHQ) about the strategic directions of system development within EMEA, EBSAT is searching for the answers to these kind of questions.

The assignment description

1. Model a case study with BSDM and evaluate BSDM.
2. Describe the object oriented paradigm and the advantages and disadvantages of using the object oriented paradigm for system development.
3. Describe to what extent BSDM specifications can be used for object oriented system development.
4. Build a demonstration model of the development cycle with Smalltalk/VPM, using the modelled case.
5. Evaluate the mission, the case, the demonstration model and the development cycle.

The reasons to use Smalltalk/VPM is that it is the most extreme object oriented language and it supports the presentation manager possibilities of OS/2. OS/2 is the operating system for the PS/2. One of the main features is the possibility for multitasking which means that different applications can run simultaneously. The user interface of OS/2 consists of icons, windows, menubars, etc., using a mouse.

Working plan

The tasks for finding the solution for the problem area are:

1. Studying BSDM
   This task comprises mainly reading the BSDM manuals and asking questions about BSDM.
2. BSDM case study
   Working out a case study that demonstrates important aspects of BSDM.
3. Studying the object oriented paradigm
   This task consists of reading important literature about the object oriented paradigm and write a report about this literature. Part of it will be an overview of the advantages and disadvantages of using the object oriented paradigm for system development.
4. Learning Smalltalk/VPM
   Learn object oriented programming with Smalltalk/VPM.
5. Integrate BSDM and the object oriented paradigm.
   Define a draft development process that integrates BSDM and the object oriented paradigm. There is no connection (at the start of the assignment) between BSDM and the object oriented approach.
6. Transition for the case study
   Demonstrating BSDM - OOD/OOP for the case study.
7. Smalltalk implementation
   To demonstrate BSDM - OOD/OOP, the case study will be partly implemented into a Smalltalk/VPM application. This implementation is meant to demonstrate important aspects.
8. Conclusions and recommendations
   Finally, an evaluation will be given of BSDM, the case study, BSDM - OOD/OOP and the assignment as a whole.

Task 7 has not been carried out.

Chapter 4: BSDM

Overview

BSDM means Business Systems Development Method. BSDM is the IBM development method which was a result of an IBM U.K. task force, which originated in 1979. This task force was called to life because of negative experiences with Business Systems Planning. This task force had the objective to research methodologies and methods and eventually develop an own method.
The original name of this method SDM, which means systems development method, has been changed to IBS/DM (Business Systems/ Development Method).

Recently (June 1990), the name has been changed again into BSDM.
Within IBM - EMEA (Europe, Middle East and Africa), BSDM is mandatory for I/S since December 1987. IBM U.K. uses the method for (external) customers.

According to the manuals, the basic aims of BSDM are to improve:

- Productivity: The speed of systems development.
- Quality: The quality of the systems produced.
- Flexibility: The possibility of extending and changing these systems easily.

The principal features of BSDM that contribute to meeting the aims just given are:

- Business definitions and technical definitions are separated.
  Business definitions are relatively long-lasting and likely to affect more than one generation of a system. Technical definitions, are liable to change as technology changes, and should therefore be postponed till the basic business requirements have been settled.
- Data analysis and process analysis techniques are combined.
- A step-by-step approach helps developers to concentrate on one thing at a time, and then move on to the next step in a logical sequence. The manual states that although the method is presented sequentially, BSDM should be used as an iterative method.
- Documentation standards are recommended. BSDM makes use of a number of forms and diagrams. Narrative, which is difficult to verify, reference, and maintain is reduced to a minimum.

BSDM comprises the following:

1. Business Modelling (Appendix A and B).
   In this part the business model is defined. Business modelling is concerned with defining the WHAT (nature of the business), not the HOW (business implementation). This phase consists of creating a data model and a process model. First the relevant business entities (data model) with their attributes are defined and documented. Then a process model, which expresses business rules, is derived from, and related to, the data model. Figure 2 on page 17 provides an overview of business modelling.

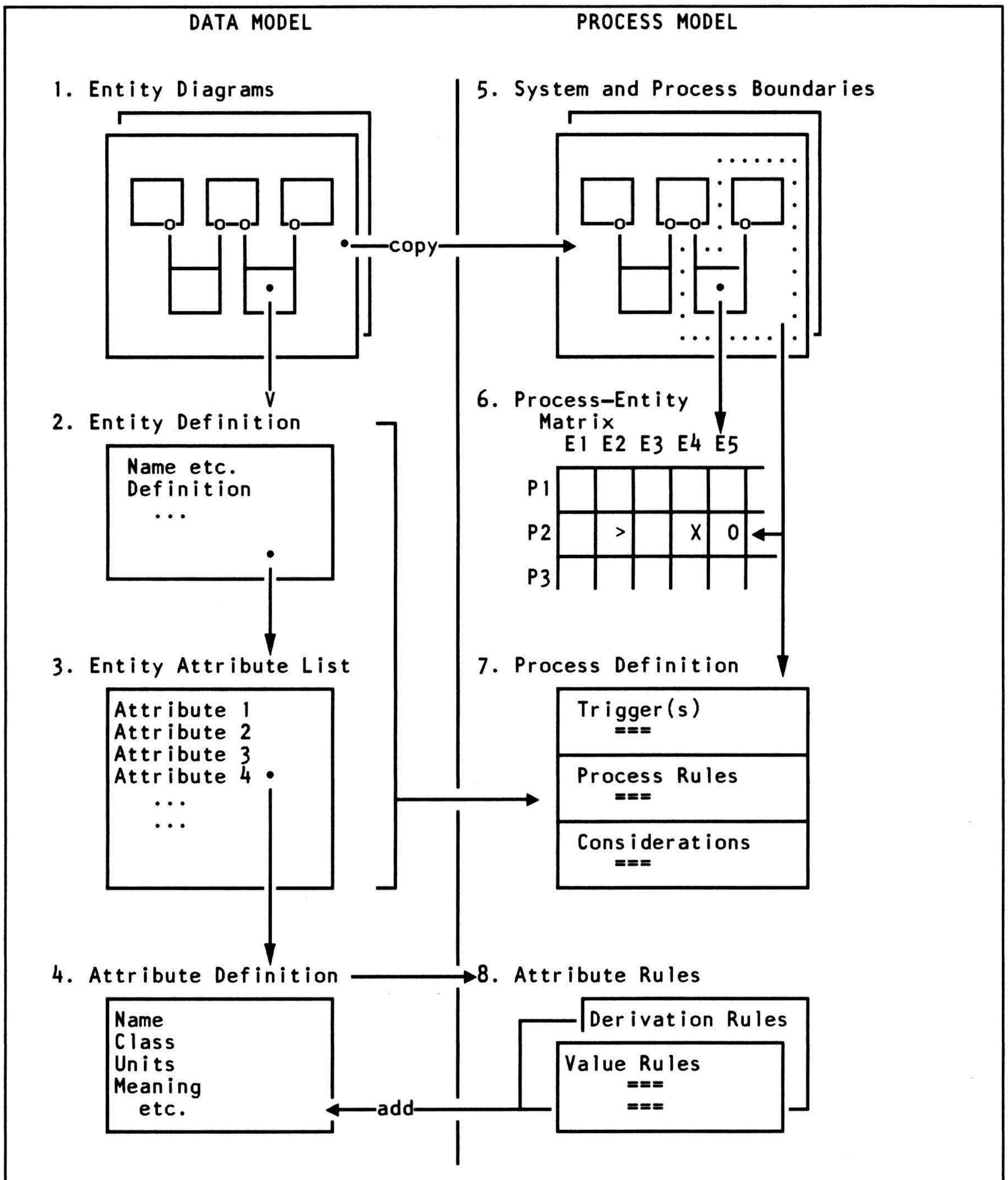


Figure 2. Overview of Business Modelling: First the data model is made: Identifying entities, making the entity diagram, defining the entities, identifying attributes and defining the attributes. Second the process model is made: Establishing system and process boundaries, making the process entity diagram, defining the processes and specifying the derivation and value rules.

The business model contains entities and processes on a far more detailed level than the master business model.

2. Systems Architecture (see Appendix C).
The purpose of the systems architecture part is to provide a high-level framework within which multiple systems can be coherently planned and developed. This implies taking a wider view than is possible when designing a single system. In this part a master business model is defined. BSDM recommends that the level of detail is limited to defining entities, business processes,

and detail on location (distribution) and function, as the need is to cover a wide scope of the business.
Building an BSDM architecture is a means of seeing, far more clearly than it otherwise could, the shapes of the systems and how those systems should fit together with as little conflict and interference as possible. This is instead of allowing each system to set its own boundaries, which may well cause serious interface problems in the future.
There is an overlap in the modelling phase between architectural and development work. Within BSDM, this is reflected by the use of common techniques for both. The modelling done in an architecture study is less detailed than when developing a system, but the principles are the same. Where an architecture is produced, this overlap of techniques should ensure a high degree of continuity between architecture and development.

3. Requirements
   In this part the detailed needs are identified and defined. Dependent on the defined requirements, design work will be planned.

4. Design
   This part involves transforming the business model and requirements into ***one or more*** designs for implementation. This part consists of an outline design and a detailed design. Outline design concentrates on the system as it will be experienced by the user. During detailed design, those parts that are to be handled by the computer are translated into appropriate files and programs. Each new version of a design should be based on, and cross referenced to, the common set of definitions and rules in the business model.

BSDM does not comprise the programming phase.

In the appendices D, E and F the case study 'Tiger Video' is described. This case study comprises BSDM Business modelling.

Evaluation of BSDM

The BSDM business model is a model of WHAT the business does and WHAT information it needs to record. The business model does not specify HOW the business does it or HOW it records its information. An example of what it records is the business process 'take CUSTOMER ORDER'. It is not important how this is done. It only means: take it. When we want to know how this is currently done, we have to analyse the current organisation. The result might be: 'take CUSTOMER ORDER by telephone' or 'take CUSTOMER ORDER by order form'. Business models are not intended to and do not record how the business is currently run, the current procedures within the business, the current departments, people, products, etc. BSDM business models are not models of the business realization.

BSDM does not comprise the programming phase, and it also does not comprise a project management method. Both are supported by other IBM products. I have not evaluated how these fit with BSDM. This is outside the scope of the assignment.

Part of BSDM are a set of standard forms which are recommended for documenting. These forms are filled in during the development process, so the documentation will not be made after, but during the development process.

The BSDM manuals alone are not sufficient to practice BSDM. Although this is partly due to the manuals themselves and to the experience someone already has with system development methods, practicing BSDM asks for education and BSDM experts. My own experience in making the business model of Tiger Video is that the manuals alone were not sufficient.

Because the essential activities and data are very similar for similar organisations, the business models of similar organisations will look very familiar! For example the business model of the case study will have much in common with the business models of other videotheques.
In other words, the business model of Tiger Video will be applicable (with some changes) to most of the (other) videotheques.

Therefore, business models have the potential to be used as reference models which have to be adjusted for specific organisations.

The users are needed to model their own business. They can understand the models. But to get the right information out of the users, you need well educated and experienced moderators.

Confronting end-users with entities, attributes and business processes can give certain problems like misunderstanding and unwillingness.
Practical use in England, Italy and Germany has shown that using BSDM business modelling with users seems to be successful. The end-users are active in the creation of the business model and they say that they are content with the current way of working.

Strengths

- Making a business model will certainly give you more insight into the business that you are modelling or is modelled.
- Business models are intended to model the nature of the business, not the business implementation.
- Business models are relatively stable. For example: When the business changes from taking an order by telephone to take an order only by order form, the business model remains stable. A model that represents the business realization must be adjusted in that case.
- Data and processes are integrated very well.
- Non-functional demands (constraints) are identified and defined (mainly during the requirements phase).
- Business models have the potential to be used as reference models, that have to be adjusted for specific organisations.

Weaknesses

- BSDM does not have a formal description to analyse improvements of the business. Starting from the current (and future) problems in the business, system development can be the right solution. But other solutions should be considered too. Business models can be used and should be used to analyse and improve the business[1].
  In the current situation, people are not forced to do this by the method.
  One might argue that this should not be part of a system development method. But in that case, there should be a relation with such a kind of process for improving the business. And that is not the case.
- There is no (precise) definition of the situations for which BSDM is suitable. There is also no information in which situations BSDM has proved successful and in which situations BSDM has proved to be less successful. This should be documented!
- There is still no workbench available for BSDM. It is possible to use BSDM without a workbench. But certainly for larger projects, a workbench is badly needed.
- BSDM recognizes the importance of some aspects, but does not elaborate on them. Examples are:
  - Corporate objectives and problems.
  - System objectives.
  - Organisation.
  - Relevant business functions, other than information processing functions.
  - Global and detailed development plan.

---

[1] A system development method which provides a possibility for analysing the organization is the system development method ISAC. ISAC contains the part 'change analysis'.
'By change analysis we mean analysis of what kind of changes (or rather improvements) of the activities in an organization should be strived for in concrete situations.
It should also be kept in mind that work with change analysis is really no extra work, even in cases in which we are relatively sure of what type of changes we are aiming at. The results from change analysis can be used directly in order to speed up the subsequent development' (Lundeberg, Goldkuhl and Nilsson).

  - System behavior. It is not clear whether BSDM can be used for technical information systems.
  - User manuals.
  - Detailed specification of the transition from the current situation to the new situation.
- Although there have been some changes in using BSDM, the manuals do not reflect these changes. The manuals have not been changed in the past 5 years!

General conclusion

BSDM is a good development method, but should be improved.

Recommendations

- BSDM should be used in all countries instead of each country using a different method.
- IBM should document in which situation BSDM is suitable and in which situations BSDM has proved to be less and more successful.
- IBM must have a good workbench for BSDM.
- IBM should investigate what the possibilities are for reuse of (parts of) business models. There is potential for 'reuse' of business models that should be investigated !
- There should be a study to what extent BSDM supports improvements of the organisations that are modelled with BSDM business modelling.

Chapter 5: The object oriented paradigm

The object oriented paradigm, which can be used for system development is liable to growing interest. This paradigm is a way of structuring {Lit. 9. on page 48}, which arises from the following set of programming concepts/techniques:

1. Data abstraction.
   The concept of defining data elements, based on abstract attributes, which are shared by the data elements.
2. Information Hiding / Encapsulation.
   The basic principle behind a modularization on the basis of the data. The principle states that all procedures manipulating a datastructure should be grouped around that datastructure. The data is then encapsulated by the operations that act upon them. Access to the data is only via the routines, so that in effect, the exact form and definition of the data becomes hidden for the rest of the program.
3. Inheritance.
   A reuse concept, based on specializations and generalizations. Inheritance relates to both elements of encapsulation: data and the operations that operate upon the data. Levels of specialization and generalization result in a hierarchical inheritance structure.
4. Polymorphism.
   Polymorphism allows programmers to use exactly the same name for similar operations on different types of objects.
5. Dynamic binding.
   This is also called late binding. Dynamic binding makes it possible to send messages to objects whose class/type will be known at run-time only, and not at compile-time.

The OO-literature and practice is mainly concerned with object oriented programming (OOP).

Objects

Within the object oriented paradigm, objects play a central role. An object corresponds with, or represents things in the real world. For example:

- Numbers: 6, 123, 99.
- Words: 'hello', 'IBM','telephone'.
- Persons: Mr.Johnson, M.Thatcher, the Queen.
- Cars: Your car.
- Files: Test.dat, Letter.scr.
- Windows and Icons.
- Products: PS/2, Manual.

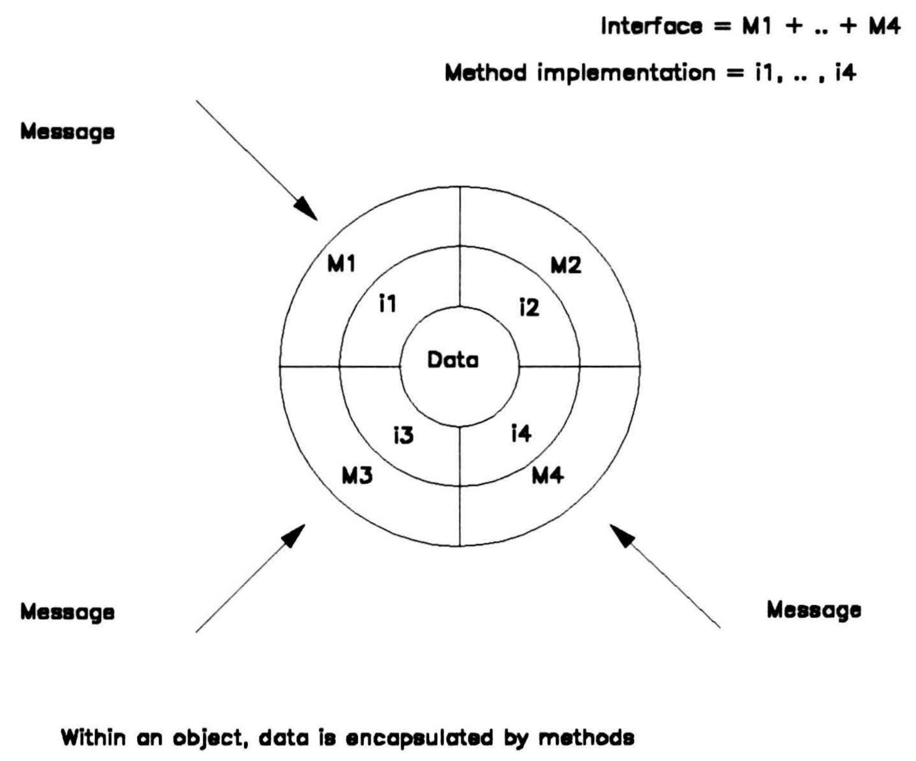

Interface = M1 + .. + M4
Method implementation = i1, .. , i4
Message
M1
M2
i1
i2
Data
i3
i4
M3
M4
Message
Message
Within an object, data is encapsulated by methods

Figure 3. Encapsulation / Information Hiding: The data can only be accessed by the methods. A method has its interface (M.) and its implementation (i.). The method implementation can be changed without changing the method interface. Methods will only execute, when a corrresponding message is sent to them.

An object consists of data (information) and methods (procedures, code, routines) which surrounds the data. The methods are the only routines authorised to access the data of an object. The data is encapsulated by the methods (encapsulation). From the outside of an object, the separation between the data and methods cannot be seen.

Messages

Objects communicate with each other using messages. Although this is not the only approach, this one is mostly used. A message is a request from an object to another object to carry out one of its methods. The object which sends a message, is the sender. The object which receives this message, is the receiver. This is shown in Figure 4.

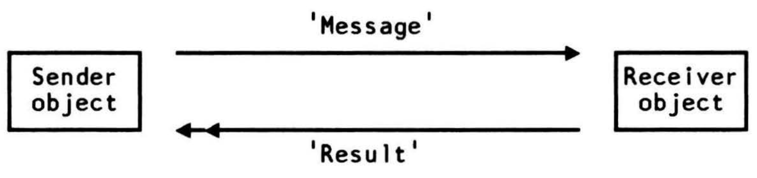


Figure 4. Communicating objects: The sender object sends a message to the receiver object. The receiver object executes the corresponding method and returns the result of this execution to the sender object.

The only thing a sender object should know is what should happen, not how that procedure should do it. How something should be done is recorded in the method implementation of the receiver object itself. See Figure 3. The receiver object executes the corresponding method, and then sends the result of executing this method to the sender object. The set of messages to which an object can respond is the interface of an object. See Figure 3 on page 22. Without changing the interface, the actual implementation of the methods can be changed.

The following example demonstrates the use of messages. There are two objects, object car and object person. This means, there is a car in the real world which is represented by an object, and there is a person in the real world which is represented by an object. See Figure 5.

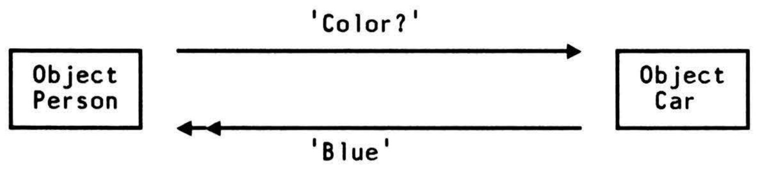


Figure 5. The message 'Color?' with the corresponding result: Object Person is the sender object. Object Car is the receiver object.

The object person asks the object car: 'What is your color?'. The object car answers: 'My color is blue!'. This sounds perhaps strange, in fact it is, but we are talking about an object oriented representation of the real world. Not the real world itself.

An object cannot respond to all messages. The next example shows this.
There are three objects: person, car and elephant. The object person sends the message 'What is your color' to the object car. The answer of the receiver is: 'Blue'.
The object person sends the same message to the object elephant. The answer of the elephant is: 'Grey'.
Until now, there is no problem. Let's consider the next situation:
The object person sends the object elephant the following message: 'What is the size of your ears? The answer is: 'Very large'. But when the object person sends this message to the object car, the car does not know what ears are. This message is unknown to the car, the object car does not know what to do.

Classes

The idea of information hiding has been further developed into abstract datatypes, which may be seen as the application of information hiding. The basic idea is that all data of a certain type is under the control of a single type manager module. The implications are that the consumers have absolutely no idea of the implementation of the type. Instead, they only know some 'abstract type' together with the names of all routines (See M1,..,M4 in Figure 3 on page 22). that operate on that type. More than one instance of a certain datatype can be used at the same time.

In OO, modularization is completely based on abstract datatypes in the form of classes. Each class contains similar objects. Each object belongs to one and only one class. All instances of a class have an identical set of methods and, therefore, uniform behavior. Each class has its own set of instance variables (attributes).

For example, every employee of IBM can be represented as an object, and all the employees of IBM can be represented as 400.000 objects! For this set of similar objects, the class IBM-Employee can be defined. See Figure 6 on page 24. Instance variables are name, birth date, phone number, etc.

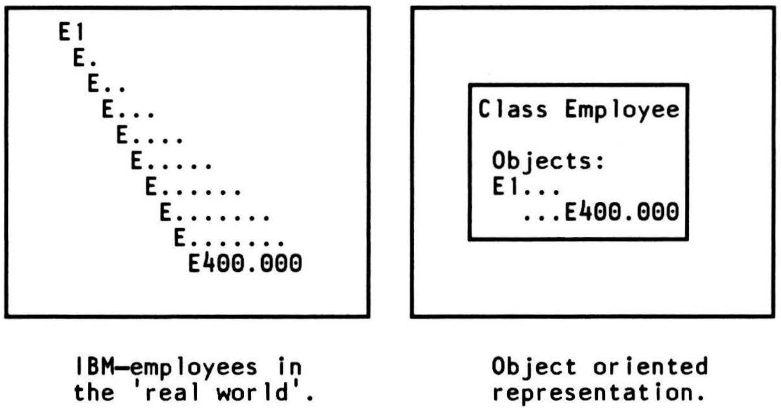


Figure 6. Objects represent 'things' in the real world: The objects of class Employee are the individual employees.

Figure 7 shows the classes Person and Car, with their communicating objects, where 'msg1' and 'msg2' represent two messages.

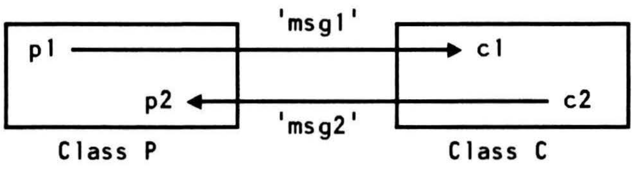


Figure 7. Two classes with their objects: Class P has two instances: p1 and p2. Class C has also two instances: c1 and c2. The object p1 sends the message 'msg1' to the object c1, and object c2 sends the message 'msg2' to the object p2.

Within the constraint that each object belongs to a class, there are some possibilities to classify objects:

1. An object is an instance of exactly one class. See Figure 8.

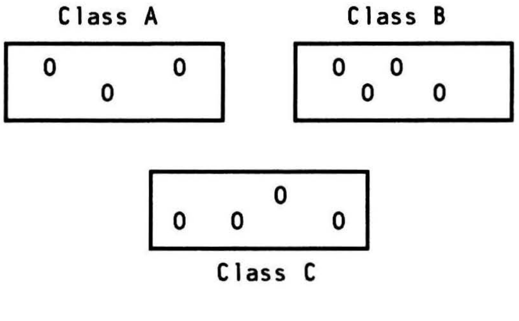


Figure 8. Each object is an instance of exactly one class.

2. An object can be an instance of more than one class.

   a. Subclassing. See Figure 9 on page 25.
   Some classes include all instances of another class.

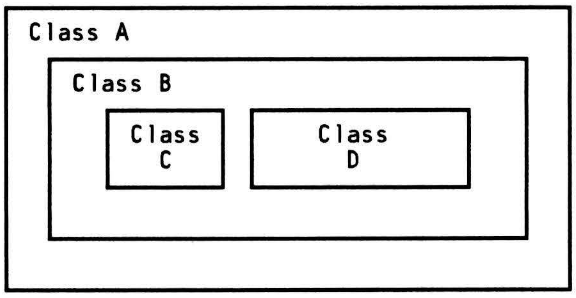


Figure 9. Subclassing

Subclassing is strictly hierarchical. If any instance of class C is also an instance of another class, class B, then all instances of class C are also instances of class B.

b. Intersection of class boundaries. See Figure 10.
Some objects are instances of two or more classes, while other objects are instances of only one class. The 'O''s in the figure indicate objects. The area, which is marked with '-', does not contain objects. The areas which are overlapped, are separate classes.

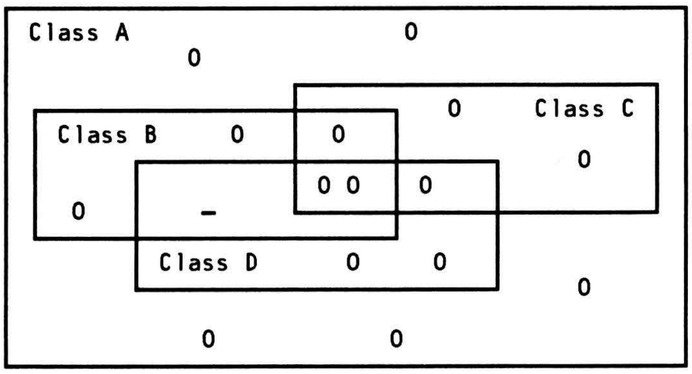


Figure 10. Intersection of class boundaries

The idea of subclassing and intersection of class boundaries is closely related to one of the most essential object oriented concepts: Inheritance.
Inheritance is a reuse concept, based on specialization and generalization. Creating a specialization of an existing class is called subclassing. The new class is a subclass (derived class) of the existing class, and the existing class is the superclass (base class) of the new class. Each subclass inherits the properties (methods, variables) of its superclass, including the method implementations. The subclass may define additional methods and variables and redefine old methods. Changes to the superclass will lead to changes in the subclasses, because the changes will also be inherited by the subclasses.

In the case that a class may only have one superclass, we speak of single inheritance. See Figure 11 on page 26.

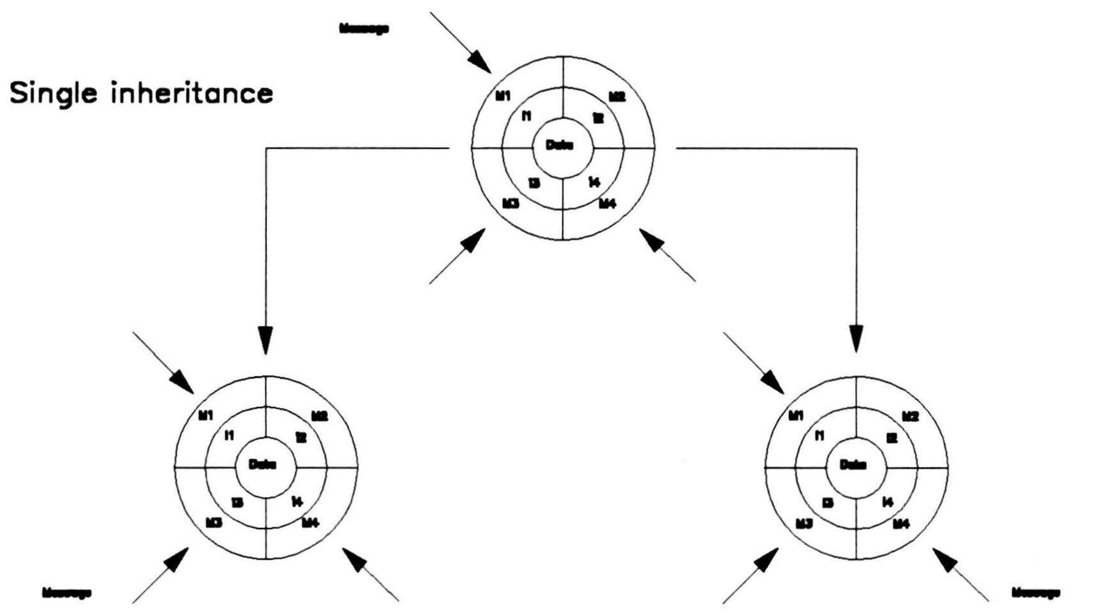


Figure 11. Single inheritance: Each object is an instance of a different class. Two subclasses inherit from one superclass.

With multiple inheritance, a class may inherit from more than one superclass. See Figure 12 on page 27.

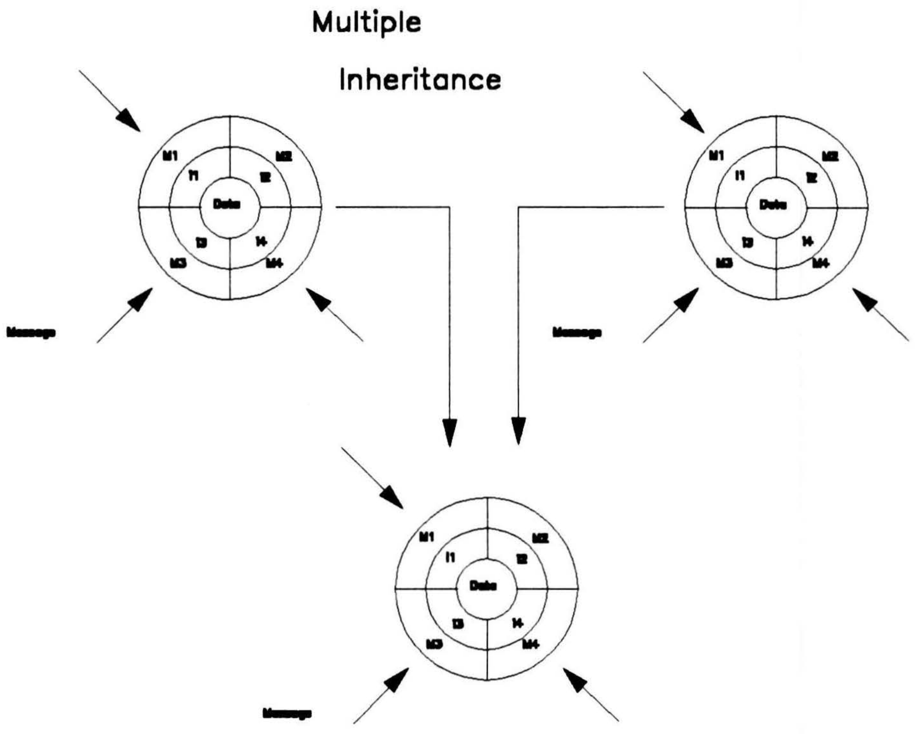


Figure 12. Multiple inheritance: Each object is an instance of a different class. One subclass inherits from two superclasses.

When each object is an instance of exactly one class, subclassing corresponds with single inheritance and intersection of class boundaries corresponds with multiple inheritance.
Multiple seems more natural, but has some important disadvantages. The programs will become more complex and more difficult to change. The potential for code sharing is greater, but the possibility of conflicts between multiple superclasses increases the complexity of such systems.

In order to support the building of object oriented programs, some object oriented languages (OOL) are developed, like Smalltalk. These languages are completely different than conventional languages like C, Cobol, and PL/1.
The various programming languages and systems vary in their support for OOP from allowing OOP to really supporting OOP.

Developing applications comprises more than only programming. So, we need more than only OO programming. Conventional designs do not fit properly with OO programming. We need something like OO design and OO analysis, based on the same underlying representation! This might improve the seamlessness of the application development process.

(Dis)Advantages of object oriented systems development

The advantages

1. Maintenance.
   This will become less, but also easier. In conventional software, the code for the functions are separated from the data. The relation between these functions and data, which exists in the technical design, will not be recorded in the software. When, for example, a data definition

has to be changed, it is not clear which subprograms must be changed. In OO-software, changes to data remain local.

2. Inheritance/Reuse.
   Inheritance enables programmers to create new classes of objects by specifying the differences between a new class and an existing class instead of starting from scratch each time. A large amount of code can be reused this way.
   Re-use is possible, using the conventional approach with subroutine libraries and/or a data dictionary. But this is in conflict with the principle of information hiding. With classes it is possible to re-use bigger parts of a program.
3. Extendibility.
   Enhancements can be made simply by creating new classes as variations of existing ones.
4. Modifiability.
   Close correspondence between the computer model and our view of the problem domain in reality.
5. Productivity.
   Starting with a good design, programming by experienced programmers will costs less time. Also maintenance will take less time.
6. Quality.
   Probably, the quality can be raised. But this depends strongly on the analysts and designers.
7. User friendly.
   Users may find objects easier to understand than entities and processes.
8. Seamless transition.
   Two models are seamlessly related to one another if concepts introduced in one of the models can be found in the other model through a simple mapping. Seamlessness is an important property of modelling, since it is a prerequisite for the development of powerful support tools. The gap between analysis and design will then become smaller.
9. Modeling dynamic aspects.
   Objects can probably be used as building blocks for modelling the dynamic aspects of the problem domain. Executing such a model may simulate the problem domain.

The disadvantages

1. Learning curve.
   Not the concepts, but using the concepts and the languages is difficult. A programmer must learn an extensive class library before becoming proficient in object oriented programming.
2. Methodology.
   There is not a well defined and widely accepted OO-methodology for developing applications. (BSDM - OOD/OOP is an attempt for an object oriented development process).
3. Experience.
   There is not enough experience which shows that object oriented development is the right choice.
   Guthery {Lit. 12. on page 48} states this in another way: If you want to write, say, an accounting system or a reservation system using OOP you're going where no man or woman has gone before. In fact, you'll be performing on yourself the very experiments that the OOP peddlers should have performed to substantiate their claims. Would you accept this situation if OOP were, for example, a new surgical procedure?
4. Standardization.
   There are no standards. We need heavily standards concerning terminology, languages and methodologies.
   One attempt to settle standardization issues comes in the form of the Object Management Group, which includes American Airlines, Canon, Data General, Hewlett-Packard, Philips International, Prime Computer and Sun Microsystems. The group is actively looking for new members. However, no guarantee exists that this group will settle standardization issues {Lit. 29. on page 49}.
5. Hierarchies are nonmodular.
   You can't just clip the objects you want to reuse out of the hierarchy because you don't know how the objects are entangled in the hierarchy. So, the cost of OOP-style code reuse seems to be that you must (re)use a lot more code than you want or need. Your system will be bigger, run slower, and cost more to maintain than with subroutine-style reuse {Lit. 12. on page 48}
6. Combining object hierarchies.

Object oriented programming makes building code fragments easier, but it makes integration much more difficult. There are neither in theory nor in practice any OOP hierarchy combiners.

7. Lack of object-management systems.
   When objects become numerous, and when relationships between objects become more complex, object management will be important. For instance, an object-based system should be consistent; changes to an object-based system should not produce harmful side effects to other classes or objects.
8. Efficiency.
   Applications which are built, using certain object oriented languages, are too slow.

Conclusions

The object oriented paradigm certainly will influence the way information systems are built. But it will take some years. It enables us to improve the development process. But the extent of improvement depends on the way we handle it. The paradigm alone will not do it!

But to improve the development process, we need a well defined methodology for analysis, design and implementation, based on one underlying representation. The experience which will be gained, must be used to improve the methodologies.

The paradigm, supported by a well defined methodology, will not guarantee an improved development process. Quality and productivity are largely determined by the people who use new technologies. Because objects should be derived from the problem domain, understanding the problem domain by analysts is essential. Analysts should be experts in the problem area.

Not everyone seems to realise that not everything changes. The object oriented paradigm does not make all the experiences and existing ideas worthless. A lot of things remain the same: Supporting the business, user involvement, education etc. We can and must use the experience and knowledge we have gained.

Large organisations will not introduce object oriented development within a short time. Lack of experience and maturity, but also the needed education effort, are important reasons for this. But if large organisations decide to go object oriented, hybrid languages, like C++, are probably the languages which they will use. Connections to existing applications, efficiency and using programming experience are important reasons which justify a hybrid transition. Within one year, the first commercial 'OO Cobol' is expected. This hybrid language may fasten the process of introducing OO within (Cobol oriented) organisations.

Although learning the concepts is not difficult, the application of the concepts is. For example, it will costs some months to learn Smalltalk and use it for serious applications. To enlighten this education problem for organisations, and to improve (technical) skills (as a country), universities should give education about the object oriented paradigm, and (interested) students should learn to build (simple) object oriented programs.

Chapter 6: Integrating BSDM and the OO-paradigm.

In this chapter the BSDM - OOD/OOP development process is described. This development process must be seen as a first attempt to integrate BSDM and the object oriented paradigm.

The approach for integrating BSDM and the object oriented paradigm lies between:

- Not OO, fully BSDM.
  This is the current situation. First business modelling and systems architecture, then requirements and design. Applications are implemented using third and fourth generation languages.
- Not BSDM, fully OO.
  This approach is fully OO, without BSDM. Starting with identifying the objects, methods, etc. from the real world, applications will be designed and implemented in an object oriented language.

One of the most important strengths of BSDM is business modelling. In the area of object oriented system development, the programming phase is (currently) the most powerful one.
Combining the strong parts might be better than only BSDM or only an object oriented approach.
Two possibilities for combining the strengths are:

- BSDM business modelling and design with object oriented programming.
  The problem here is that the results of BSDM design cannot directly be used for object oriented programming.
- BSDM business modelling with object oriented design and programming.
  The problem here is that the object oriented design phase have to be defined.

The last approach is worked out. The reasons for this approach are that IBM has invested and will invest much effort in using BSDM, and this (evolutionary) approach builds upon existing ideas, techniques and experience.

Demands

During object oriented programming and design as much as possible information from the business model should be used.

A development method should deliver a language independent design.

Classes should be identified independent of existing languages.

The transitions between the different phases should be seamless.

BSDM - OOD/OOP

BSDM - OOD/OOP starts with BSDM business modelling, systems architecture and the requirements as described in the corresponding BSDM manuals. Based on the business model, systems architecture and the requirements, one or more object oriented designs will be made which is independent of any specific language.

This BSDM - OOD/OOP development process must be seen as a first attempt to combine BSDM with the object oriented paradigm. BSDM - OOD/OOP is not a ready to use development method, but contains useful ideas which should be refined.

The business model contains a lot of information which can be used. Figure 13 gives an overview of what can be (re)used.

	Use in OOD, step:
Data Modelling	
Entities	1, 2.
Dependency structures	1, 2.
Attributes	3, 4.
Derived attributes	3, 4.
Process Modelling	
System Scope	1
Business processes	4.
Process-Entity interaction	4.
Triggers	4
Business Rules	4.
Business considerations	4.
Value rules	4.
Derivation rules	4.
Design	
Tasks	4.
Dialogues	7.

Figure 13. Usage of BSDM deliverables: Behind each deliverable, the step of phase 4 is mentioned where this deliverable is used.

Behind each deliverable, the step of BSDM - OOD/OOP is mentioned where it is used.

- The BSDM entities will be used for identifying classes.
- The dependency structures will be used for identifying pieces of class hierarchy.
- The attributes of the entities correspond with the instance variables of objects.
- The system scope will be used for determining the scope of the application.
- The business processes (in combination with the attributes) will be used for deriving methods. A method can be used by more than one business process.
- Value rules and derivation rules are used to identify and define methods.
- Dialogues are used for designing the user interface.

BSDM - OOD/OOP is partly demonstrated for the case study 'Tiger Video' in appendix I.

Phase 1: Business Modelling

This phase is described in the BSDM 'Business Modelling' manuals {Lit. 6. on page 48}.

Phase 2: Systems Architecture

This phase is described in the BSDM 'Systems Architecture' manual {Lit. 6. on page 48}.

Phase 3: Requirements

This phase is described in the BSDM 'Requirements' manual {Lit. 6. on page 48}.

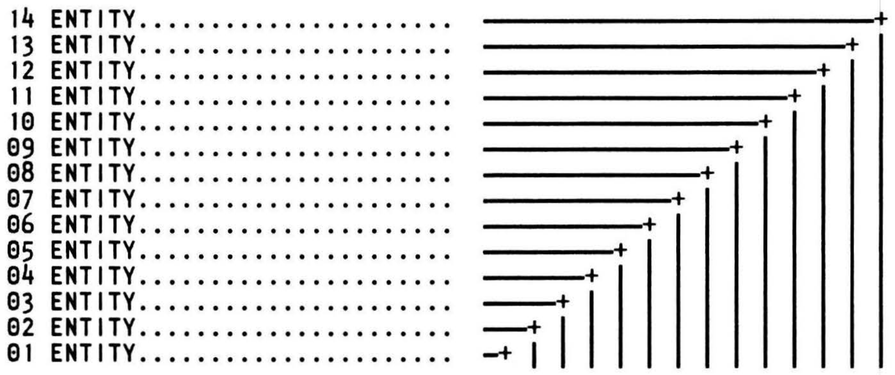
Phase 4: Object Oriented Design

Step 1: Identify dependency structures

In this step, the dependency structures (see Appendix A) in the entity diagram of the BSDM business model are explicitly identified. These dependency structures are:

1. Simple structure
2. Double dependency
3. Involution
4. Generalisation
5. Membership
6. Reconciliation
7. Alternative parents

The defined dependency structures cover all the entities. Some entities are part of more than one dependency structure. These are called the multi-covered entities. An overview of the dependency structures and the (multi-covered) entities, is given in the 'entity-dependency structure diagram'. See Figure 14 on page 33. This diagram shows all the entities, dependency structures and the relations between them. It shows clearly which entity is part of which dependency structure. It also shows clearly in how many dependency structures an entity is involved.


14 ENTITY......................
13 ENTITY......................
12 ENTITY......................
11 ENTITY......................
10 ENTITY......................
09 ENTITY......................
08 ENTITY......................
07 ENTITY......................
06 ENTITY......................
05 ENTITY......................
04 ENTITY......................
03 ENTITY......................
02 ENTITY......................
01 ENTITY......................

id.	Description	01	02	03	04	05	06	07	08	09	10	11	12	13	14
S01	Simple structure 1														
02	Simple structure 2														
D01	Double dependency 1				i		d								
02	Double dependency 2														
I01	Involution 1														
02	Involution 2														
G01	Generalisation 1														
02	Generalisation 2														
R01	Reconciliation 1														
02	Reconciliation 2														
M01	Membership 1														
02	Membership 2														
A01	Alternative Parent 1														
02	Alternative Parent 2														

Figure 14. Entity - dependency structure matrix: The relation between entities within each dependency structure is indicated by an 'i' or a 'd'. The 'i' means independent entity within the dependency structure, and the 'd' means dependent entity within the dependency structure. For example, ENTITY 6 is double dependent on ENTITY 4.

When there is a large amount of entities and dependency structures, it is more surveyable to make a diagram for each type of dependency structure.

Step 2: Define inheritance structures

During this step, each dependency structure will be transformed into one corresponding and convenient inheritance structure. An inheritance structure is the result of transforming one separate dependency structure into a little class hierarchy. During this step, classes are identified based on the BSDM business model.

Show generalization structure

In an entity diagram, only the 'highest level entity' of a generalization structure is shown. This entity is represented in the class hierarchy as a class. Its subclasses have to be identified here.

Transform involution structure

The dependent entity within the involution structure corresponds with a subclass of a class that is similar to the Smalltalk dictionary class. The independent entity is a subclass of class object.

Transform Membership structure

The membership structure consists of three entities and will be transformed into a corresponding class hierarchy with two classes.
The 'type' entity (for example: PRODUCT TYPE) will be the superclass of the entity type (PRODUCT). The 'member' entity will not be implemented as a separate class (for example: MEMBER OF PRODUCT TYPE). The reason is that Product as subclass of Product Type implies that a product is of a certain product type.

Step 3: Transform attributes into instance variables

An attribute of an entity is comparable with an instance variable of a class. Therefore, the attributes of entities help identify and define the instance variables of objects.
The attributes of the entities have to be put as high as possible in the class hierarchy (when useful).

Step 4: Identify methods

For each class, the needed methods have to be identified and defined. These methods will be derived from the BSDM business processes, attribute derivation rules, and attribute value rules.

For the business processes, we have to identify methods that:

1. Create objects.
   These methods become class methods of the class whose instances are created.
2. Assign values to the attributes.
   These methods assign the values to the instance variables.
3. Change values of attributes.
   Existing values of attributes can be changed by this method.
4. Perform calculations.
   These methods calculate due dates, magnitudes, time differences, dates, etc. Such a method may be periodic in nature. A calculate method can be used for calculating the values of BSDM derived attributes. Calculate methods become class methods.
5. Monitors for real-time processing.
   These methods perform on-going monitoring of an external system, device, or user.

Task 1: Determine create methods

For each process-entity interaction, determine whether create methods are needed. See Figure 15 on page 35. These can easily be identified, because during business modelling is recorded (e.g. process-attribute matrix) which processes originate occurrences of which entities. These methods become class methods.

Business Process (O)	ENTITY (O)	originate method
	ENTITY (O)	originate method
	ENTITY (C)	-
	ENTITY (R)	-

Figure 15. **Process Entities Table with originate methods:** The entities are easily derived from the process entity matrix. If the business process originate occurrences of an entity, this business process will use the originate method for the class that corresponds with that entity. class.

Task 2: Identify process attribute interactions

For each process entity interaction, determine which attributes are involved. See Figure 16.

Business Process (O)	ENTITY (O)	or.	Attribute	O
			Attribute	O
			Attribute	O
			Attribute	O
			Attribute	O
			Attribute	O
			Attribute	O
	ENTITY (O)	or.	Attribute	O
			Attribute	O
			Attribute	O
			Attribute	O
	ENTITY (C)		Attribute	C
			Attribute	C
			Attribute	R
	ENTITY (R)		Attribute	R
			Attribute	R

Figure 16. **Process Attributes Table:** This table contains all the attributes whose values are needed by the business process for which this table is made. The 'or.' means that this process needs a create method for the class that corresponds with that entity.

A process attributes group (PAG) is a group of attributes that is needed by one business process. A process attributes table represents one PAG.

Task 3: Identify attribute methods

For each involved attribute, determine whether assign, change, calculate, reference or monitor methods are needed. A process methods group (PMG) is a group of methods that is needed for a specific business process.

Business Process A (O)	ENTITY 1 (O)	Attribute (O)	assign change reference
		Attribute (O)	assign change reference
	ENTITY 2 (C)	Attribute (C)	change reference
		Attribute (R)	reference
	ENTITY 3 (R)	Attribute (R)	reference
		Attribute (R)	reference

Figure 17. Process methods table.

Task 4: Specify the assign methods

Using the BSDM value rules, the assign methods can be specified. Value rules state what values an attribute may have.

Task 5: Specify the calculate methods

Using the BSDM attribute derivation rules, the calculate methods can be specified. Attribute derivation rules state how the values are derived for an attribute.

Task 6: Collect the methods for each attribute.

For each attribute, the methods that belong to that attribute are collected. A group of methods that belong to an attribute is called an attribute methods group (AMG).

Task 7: Collect the methods for each class.

Class Name:		
Class methods	Instance variables	Instance methods

class method 1
class method 2
class method 3
	Instance variable 1	method 1
		method 2

	Instance variable 2	method 4 method 5 method 6 method 7
	Instance variable 3	method 8 method 9

Figure 18. Class specification table: In this table the class methods, instance variables, class methods, instance methods are summarised for each class.

For each class, the methods are collected.

Task 8: Make business methods

Business methods are methods that start executing a useful combination of methods. For example, a useful combination can be: assign the name, assign the birth-date and the phonenumber of a new customer. Business methods have to be derived from business processes.

Step 5: Establish relationships between classes

In this step the relationship of each object in relation to other objects will be established. This is needed when methods of an object needs the methods or attribute values of other objects.

Step 6: Build Class Hierarchy

In step 2 the classes are specified, in step 3 the instance variables are specified, in step 4 the attribute methods are specified and in step 5 the messages are specified. During this step, all the information that is gathered in the previous phases will be used for building the class hierarchy.

Step 7: Reuse classes

In this step reuse of predefined classes will be identified. The identification of reusable classes will be done independent on a specific programming language. Reuse of classes depends on several aspects like the kind of business, the application, the user interface etc.

Reuse Business classes

A business class is a class that is specific for a business area. For example, a baker will certainly uses some other classes than a computer firm. During this task, classes are reused that are dependent on the business area of the business for which the application is made.

Reuse application classes

An application class is a class that is specific for a certain application area. For example customer registration, product information system, manufacturing applications. During this task, classes are reused that are dependent on the kind of application that is under development. For example. when different organisations want to build an application that registers employees, they need some similar classes.

Identify and reuse user interface classes.

One of the important aspects of building an application is the user-interface. The future applications within IBM will have a user interface that consists of windows, icons, and so on. Within a language like Smalltalk, the windows, icons, and so on will be represented as instances of 'user-interface classes'. But the choice for the right classes depends heavily on the CUA guidelines. Before building object oriented applications, there have to be classes that are based upon these guidelines. So the choice for the windows, icons, menus, etc, can be made when these 'CUA classes' are made.

Identify and reuse Error handling classes

During this task, error handling is defined. Reuse of 'error handling classes' should be possible. Smalltalk defines the error handling in class object, which is inherited by all the subclasses.

Identify and reuse I/O classes

During this task the needed I/O classes are identified and reused.

Phase 4: Object Oriented Programming

Step 1: Implementation

In this step, the application is build, using a language like Smalltalk or C + + . During the implementation phase, the implementation of the reusable predefined classes will be used.

Step 2: Testing

In this step, the application will be tested.

Chapter 7: Evaluation of BSDM - OOD/OOP

In the previous chapter BSDM - OOD/OOP is described.
In this chapter BSDM - OOD/OOP will be discussed: the strengths, the weaknesses and the areas that need further attention.

Strengths

- The class hierarchy that is designed within BSDM - OOD/OOP is language independent. This means that BSDM - OOD/OOP is not limited to one specific programming language.
- BSDM - OOD/OOP assumes the availability of a set of reusable classes that is independent of the languages which will be used to implement the application.
- BSDM - OOD/OOP shows that almost all the information that is recorded during BSDM business modelling can be used for developing object oriented applications.
- Because BSDM - OOD/OOP heavily depends on business modelling, business objects are easily identified.
- BSDM - OOD/OOP is rather seamless. The transformations are not very complex.
- The entities are more clearly reflected in the software than in conventional programs.

Weaknesses

- BSDM - OOD/OOP assumes the availability of a set of predefined classes. But there is no set of predefined classes available.
- BSDM - OOD/OOP is demonstrated only one case study. It should be used for a real application, based on a real business model.
- This is not a ready to use development method. It is a preliminary development method that should be refined further.

Attention areas

Predefined classes

The development process that I have defined is intended to make use of a set of predefined classes. In practice, a development department of a company should have a set of classes that will be reused as much as possible. Before we are able to reuse predefined classes, such a set has to be defined. We need business classes, user interface classes, I/O classes, etc. The set of system classes within Smalltalk contains some useful classes, but there should be more.
There are several questions that must be answered:

- What are the criteria for making a class predefined ?
  Before we identify and define reusable classes it should be clear what the criteria are for making a class predefined or not.
  Criteria might be: Can be reused many times, relatively stable, of interest for a long time, etc.
- How are predefined derived ?

Definition of entities must be consistent with the definitions of the same entities in other departments. So must be the definitions of predefined classes. Therefore, some predefined classes might be derived from business models.

- When should a predefined class be reused ?
  A predefined class will certainly not only be used when this class fits fully to certain requirements. Many times, some parts of a predefined class are not necessary and sometimes it is missing something that is needed. There should be rules for deciding under which circumstances classes may be or must be reused. A (simple) rule might be: At least 80 percent of the methods is reusable.
- How are changes to existing predefined classes managed ?
  Changing the definition of predefined classes will be unavoidable from time to time. But when are changes allowed ? And what are the consequences for the applications that have used that predefined class ?
- Should we only have predefined classes or also predefined methods and variables ?
  Instead of reusing classes it might be possible and useful to reuse bigger parts, like a piece of class hierarchy or smaller parts, like separate methods and variables.
- Can different organisations use a same set of predefined application classes?
  Some kind of applications are made by many firms, active in many business areas. They need the same kind of predefined application classes.
- Can different organisations in the same business area use the same set of classes ?
  There might be a possibility for reusing business classes by different firms in the same business area.

Business simulation

Simulation of the business is only possible, when we take the current business into consideration, included the forms, products, etc. BSDM business modelling does not want to record the way the business is run. Certain objects are not identified by business modelling when we want to simulate the business, like order form. This is missing in business modelling when we want to simulate the business with BSDM business modelling.

Generalisation and inheritance

BSDM tends to generalise entities. But the generalization - specialization relationships are needed to use inheritance. BSDM will more generalise than the object oriented paradigm does. Because BSDM business modelling tends to generalize and the object oriented paradigm tends to identify the specializations of the generalizations, inheritance and generalization conflict.
In sequence, they do not conflict. First generalization and then specifying the inheritance.

Other development methods

BSDM - OOD/OOP contains ideas that possibly can be used for other development methods too. For example, identifying structures in an entity relation model, transforming attributes into instance variables, deriving methods from functions, dividing classes into different groups (business, user interface, etc.) and dividing methods in different groups (create, change, originate, etc.).

Roles

BSDM sees people as having different roles. But the object oriented paradigm does not represent this in its paradigm.

Reuse of predefined Smalltalk classes

The set of predefined classes from Smalltalk are not sufficient to build the applications that IBM and other companies are aiming at. For example, we need a set of classes that support CUA and error handling.

Structure within Smalltalk

In Smalltalk everything is an object. Not only numbers, characters, or instances of class Person, but also the classes, instance variables and methods are objects! For system development, there must be a clear distinction between objects, classes, instance variables, methods, and so on. An instance variable must be seen as an instance variable and a class must be seen as a class, etc. There also should be a clear distinction between business classes, application classes I/O classes, etc. For system development, there should not be an approach that is so unstructured and fuzzy as Smalltalk does.

Method components

Smalltalk has a 'magnificent' possibility to copy (parts of) methods. But copying code this way has great disadvantages, especially when maintaining the system. Each change, which you want to make in code that is copied all over the place, has to be made to each copy. This way of reuse will give you a lot of problems, when building and maintaining complex applications, where a lot of people are involved.
Certain pieces of code are part of more than one method implementation. It might be very useful to identify these parts as 'method components' and define them explicitly for reuse. See Figure 19.

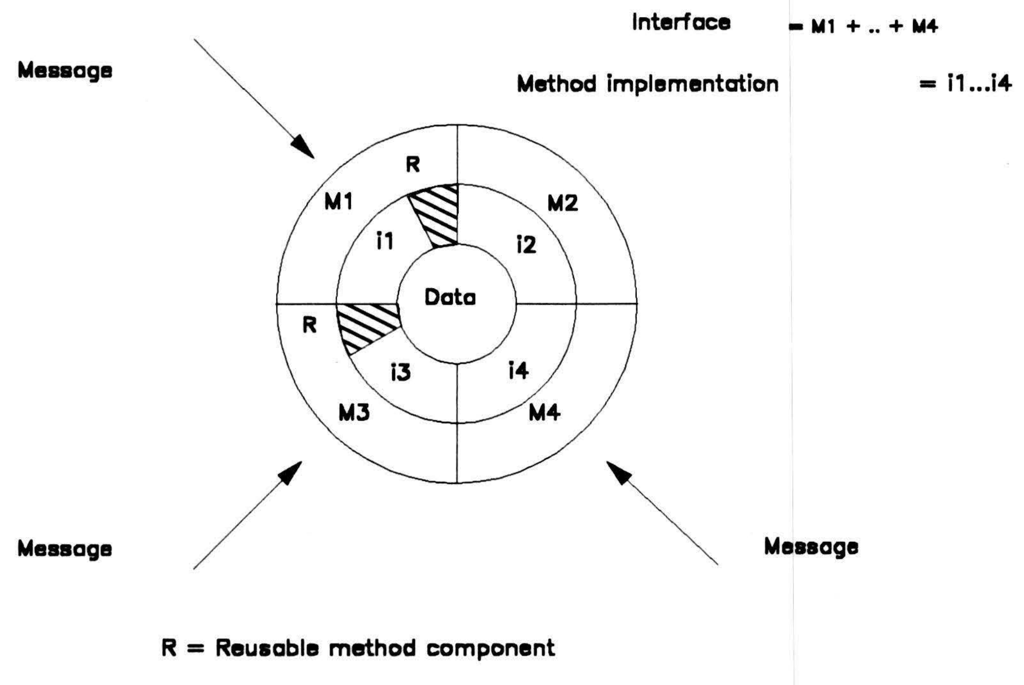


Figure 19. An object with reusable method components.

Specifying the method components in a superclass makes it possible to reuse the method components by inheritance.

Inheritance indicator

Inheritance is seen as one of the benefits of the OO paradigm. 'The more inheritance, the nicer the program!' Within Smalltalk, class variables, instance variables and methods are inherited. But can we measure the extent of inheritance? Can we define an indicator which shows the extent of inheritance? Is it useful to have an inheritance indicator?
These kind of questions should be answered.

Triggers and business processes

During BSDM - OOD/OOP the triggers that are identified during process modelling are not used, while they are needed. A possibility is to have a class business process, whose instances represent business processes. When these processes are triggered, they send messages to invoke the processing that is wanted. The trigger is for example input from the user.
This is an interesting idea, but we are missing the information from the business implementation. One of the things we need are the task definitions.

Multiple and single inheritance

BSDM - OOD/OOP assumes only the possibility for single inheritance, not for multiple inheritance. BSDM - OOD/OOP is based upon single inheritance. Multiple inheritance provides the possibility to inherit from more than one superclass. The class Customer can in that case be a subclass of class Person and class Business. Some dependency structures might be be transformed different when multiple inheritance is supported. The class customer inherits from class party and class customer contract.

Membership structure

The membership structure consists of three entities. The dependent entity in this structure will not be represented as a class. The membership structure means that a product ***may be*** a member of a product type, but the corresponding inheritance structure means that product ***is a*** member of a product type.

OO change for BSDM

During BSDM business modelling it is determined whether entities are originated, changed or referenced by certain business processes. This is done on entity level.
During the transition, for every business process - attribute combination is determined if originate, change, reference, assign, calculate, or value rules methods are needed. This is done on attribute level.
For each business process, all the attributes within each process-entity combination should be provided with the identification of the different methods.

(Dis)Advantages of BSDM - OOD/OOP

The advantages and disadvantages of object oriented system development are described in Chapter 5. In this paragraph is described whether these advantages and disadvantages apply for BSDM - OOD/OOP.

Advantages

1. Maintenance
   If the stage of reusing predefined classes is reached, and the developers and programmers know these classes in depth, they'll know where to maintain. This is also true for developers and programmers who maintain, but did not develop and build the system.

2. Inheritance/Reuse
   Inheritance enables programmers to create new classes of objects by specifying the differences between a new class and an existing class instead of starting from scratch each time. This might improve the productivity.
3. Extendibility
   Enhancements can be made simply by creating new classes as variations of existing ones. This is certainly true. But we have to be careful that there should not be an uncontrolled creation of new classes, based on predefined classes.
4. Modifiability
   It is often claimed that there is (or should be) a close correspondence between our view of the problem domain and the computer model. Regarding to BSDM - OOD/OOP the real world is not reflected in the class hierarchy. It is the BSDM business model that is reflected in the resulting software.
5. Productivity
   I stated that starting with a good design, programming by experienced programmers will cost less time. Looking at BSDM - OOD/OOP the design contains specifications that directly can be implemented. This should improve the productivity.
6. Quality
   The business model is reflected in the resulting applications. The quality of the applications strongly depends on the quality of the business model.
7. User friendly
   The users are involved in making the business models. They are not involved during design. Therefore, user friendliness will not be improved.
8. Seamless transition
   BSDM - OOD/OOP is rather seamless. There is a simple mapping from business modelling to object oriented design and programming.
9. Modelling dynamic aspects
   Outside scope.

Disadvantages

1. Learning curve
   Because BSDM - OOD/OOP is based upon existing knowledge of BSDM, I expect the learning curve to be reasonably.
2. Methodology
   BSDM -OOD/OOP is a step in the direction of establishing a well defined and widely accepted methodology.
3. Experience
   Improving and using BSDM - OOD/OOP will certainly raise the amount of experience in the area of object oriented system development.
4. Standardization
   Improving and using BSDM -OOD/OOP may be helpful for defining standards.
5. Hierarchies are nonmodular
   Outside scope.
6. Combining object hierarchies
   Outside scope.
7. Lack of class-management systems
   This is a very important subject. Managing reuse of classes within BSDM - OOD/OOP is not easily organized. Lack of a good class-management system is disastrous.
8. Efficiency
   Outside scope.

Chapter 8: Conclusion and Recommendations

The assignment

1. Model a case study with BSDM and evaluate BSDM.
   The case study is made for BSDM business modelling and is described in Appendix D, E and F. The case study demonstrates very clearly most of the techniques and ideas of business modelling.
   The evaluation of BSDM is described in chapter 4 and is mainly oriented towards business modelling.
2. Describe the object oriented paradigm and the advantages and disadvantages of using the object oriented paradigm for system development.
   This is done very elaborate in the literature report {Lit. 18. on page 49} and in chapter 5.
3. Describe to what extent BSDM specifications can be used for object oriented system development.
   This can be done to a high extent. The development process should be as seamless as possible. BSDM - OOD/OOP, that is described in chapter 6, demonstrates that a rather seamless development process is possible.
4. Build a demonstration model of the development cycle with Smalltalk/VPM, using the modelled case.
   I have not implemented a demonstration model that demonstrates BSDM - OOD/OOP. To do this, the development process should be refined some further and (the design of) predefined classes have to be available. Just implement an application is not very useful for demonstrating BSDM - OOD/OOP.

Conclusion

The main conclusion is that the integration of BSDM and the object oriented paradigm is possible and most promising.

Recommendations

Improve BSDM - OOD/OOP !

The following projects are recommended for improving BSDM - OOD/OOP:

- Study BSDM - OOD/OOP and improve it.
  Study BSDM - OOD/OOP, add new ideas and improve existing ideas.
- Design and implement predefined classes.
  Design and implement user interface classes, business classes and application classes.
- Transform (part of) real business into a class hierarchy.
- Use BSDM - OOD/OOP for a (little) real application.
  When BSDM - OOD/OOP is improved and some initial predefined classes are designed and implemented, use BSDM - OOD/OOP for a little application, based on a part of a real business model.

Abbreviations

AMG	Attribute Methods Group
BSDM	Business Systems Development Method
CUA	Common User Access
EBSAT	EMEA Business Systems Architecture & Technology
EMEA	Europe, Middle East and Africa
IBM	International Business Machines
OOA	Object Oriented Analysis
OOD	Object Oriented Design
OOP	Object Oriented Programming
PAG	Process Attributes Group
PM	Presentation Manager
PMG	Process Methods Group
SAA	Systems Application Architecture
Smalltalk/VPM	Smalltalk/V for Presentation Manager

Glossary

Abstract class	A class that has no instances, but only specifies protocol (for its subclasses).
Attribute	Property of an entity which the business may be interested in knowing.
Base class	See superclass.
Class	An object that describes the implementation of a set of similar objects.
Common User Access	Defines the rules for the dialog between the human and the computer.
Dependence	The general concept of dependencies between entities.
Dependency	A particular example or occurrence of dependence.
Dependent entity	The existence of an occurrence is dependent on the existence of an occurrence of another (parent) entity.
Derivable attribute	Attribute of which the value can be obtained from one or more other (derivable) attributes.
Derived class	See subclass.
Entity	Class of thing about the business wants to keep information.
Global variable	A variable shared by all the instances of all classes.
Independent entity	The existence of an occurrence is independent on the existence of an occurrence of another (parent) entity.
Instance of a class	An object, described by a class. It has memory (variables) and responds to messages by executing methods.
Instance variable	Part of an object's private memory.
Interface	The set of messages to which an object can respond.
Message	A request for an object to carry out one of its operations.
Method	A procedure describing how to perform one of an object's operations.
Method overriding	Implementing a method in a subclass, which was already implemented in a superclass. The implementation in the subclass differs from the implementation in the superclass.
Object	Represents something in the real world and consists of data and methods.
Occurrence of an attribute	Individual instance of an attribute.

Occurrence of an entity	Acknowledged instance of an entity.
Receiver	The object to which a message is sent.
Sender	An object that sends a message to another object.
Subclass	A class that inherits variables and methods from an existing (super)class.
Superclass	The class from which a subclass inherits variables and methods.
System classes	The set of classes that come with an object oriented language (OOL).
Class variable	A variable shared by all the instances of a single class.

Literature

1. *Anderson, Bruce..*
   Object oriented programming.
   Microprocessors and Microsystems, Vol 12, No.8, October 1988.
2. *Berghout, E.W.; Maarssen, L.A.*
   OOT, Modegril en must.
   Conference documentation: 'De object georienteerde aanpak: Modegril of must?'; 6 april 1990.
3. *Blaha, M.R; Premerlani, J; Rumbaugh, J.E.*
   Relational database design using an object oriented language.
   Communications of the ACM, April 1988.
4. *Bailin, Sidney C.*
   An object oriented requirements specification method.
   Communications of the ACM, Volume 32, Number 5 May 1989.
5. *Booch, Grady.*
   Object Oriented Development.
   IEEE Transactions on Software Engineering, No.2, February 1986.
6. *BSDM manuals.*
   - SDM Introduction
   - SDM Systems Architecture
   - SDM Business Model - Data
   - SDM Business Model - Process
   - SDM Requirements
   - SDM System Design
7. *BSO*
   Introducing BSO / Aerospace & Systems.
   January 1990.
8. *Coad, P; Yourdon, E.*
   Object Oriented Analysis.
   Yourdon Press, Prentice Hall, New Yersey, 1989.
9. *Constantine, Larry.*
   De object-georienteerde aanpak en gestructureerde methoden.
   Computable, 2 maart 1990 en 9 maart 1990.
10. *Goldberg, A; Robson, D.*
    Smalltalk 80, the language.
    Addison-Wesley Publishing Company, 1989.
11. *Grant, T.J.*
    An Object Oriented Simulator for ESTEC.
    Conference documentation: 'De object georienteerde aanpak: Modegril of must?'; 6 april 1990.
12. *Guthery, Scott.*
    Are the emperor's new clothes object oriented?
    Dr.Dobb's Journal, December 1989.
13. *Hermans, M.M.J.M.*
    Object georienteerde systeemontwikkelingsmethoden.
    Seminar informatica '89/'90, Katholieke Universiteit Tilburg.
14. *Holmes, Keith; Mac Aonghusa.*
    Object Oriented Application Development
    OPUS, 25 april, 1990.
15. *Hoven, Christian van.*

Object georienteerd programmeren: Erfelijkheidsleer voor programmeurs.
PCM, Oktober 1989.

16. ***IBM.***
Annual Report 1989

17. ***Jacobson, Ivar.***
Object Oriented Development in an Industrial Environment.
OOPSLA '87 Proceedings, October 4-8, 1987.

18. ***Lambregts, P.J.T..***
Object oriented systems development.
August 1990.

19. ***Meersman, R.A.***
Object Oriented Programming, Systems and Databases: An overview.
Conference documentation: ' De object georienteerde aanpak: Modegril of must?'; 6 april 1990.

• 20. ***Meyer, Bertrand***
Object Oriented Software Construction.
Prentice Hall, 1988.

21. ***Pascoe, Geoffrey A.***
Elements of Object Oriented Programming.
Byte, August 1986.

22. ***Peters/Waterman***
In search of excellence

23. ***Ramackers, G.J.; Goedvolk, J.G.***
OO, A solution for the software crisis?.
Journal of Software Research (Vleermuis Software Research b.v.), nr.1, april 1989.

· 24. ***Ramackers, G.J.; Kuil, W.J.J.van der Kuil.***
Wat is object-oriented programmeren?.
Journal of Software Research (Vleermuis Software Research b.v.), nr.1, april 1989.

25. ***Rodgers, Buck.***
The IBM way.
Fontana Paperback,1989.

26. ***Schultz, Ron***
Object Oriented Organization Analysis.

27. ***Smalltalk/VPM***
Tutorial and programming Handbook
Digitalk Inc., 1989.

28. ***Sobel, Robert.***
IBM, marktleider, ook in de toekomst
1986, A.W.Bruna & Zoon.

29. ***SRI International.***
Object-Oriented Systems
TechMonitoring, December 1989.

30. ***Stroustrup, B.***
What is object oriented programming?
IEEE Software, May 1988.

31. ***Systems Application Architecture, an overview.***
GC26-4341-1

32. ***Tesler, Larry***
Programming Experiences.
Byte, August 1986.

33. ***Tolido, R.J.H.***
Waarom niemand op object-georienteerd programmeren zit te wachten.
Conference documentation: 'De object georienteerde aanpak: Modegril of must?'; 6 april 1990.

34. ***Tsichritzis, D.***
Object-Oriented Development for Open Systems
IFIP, 1989.

35. ***Ward, Paul T.***
How to integrate object orientation with structured analysis and design.
IEEE Software, March 1989.

36. ***Wegner, Peter.***
Learning the language.

Byte, March 1989.

Aankondiging Afstudeervoordracht

Technische Universiteit Eindhoven
Faculteit Bedrijfskunde
Vakgroep BISA

Datum	8 augustus 1990
Afstudeerder	P.J.T. Lambregts
Datum afstudeervoordracht	***16 augustus 1990***
Tijd	13.30 uur
Plaats	Eindhoven
Zaal	Kopzaal BISA
Bedrijf	IBM
Adres	Watsonweg 2
Plaats	Uithoorn
Begeleiders Bedrijf	Ir.J.de Lint
Begeleiders/Beoordelaars T.U.E.	Ir.J.Trienekens
	Dr.Ir.P.W.G.Bots (T.U.D.)
	Prof.Dr.T.M.A.Bemelmans
Start Afstudeerproject	11 September 1989
Datum afstuderen	29 augustus 1990

Samenvatting:

De afstudeeropdracht wordt uitgevoerd binnen de afdeling EBSAT (EMEA Business Systems Architecture and Technology) van IBM. Deze afdeling onderzoekt en adviseert over de interne IBM strategie voor systeemontwikkeling binnen EMEA (Europe, Middle East and Africa).
Het afstudeerwerk richt zich op twee belangrijke ontwikkelingen voor IBM. De ene ontwikkeling is het toenemende gebruik van de systeem ontwikkelingsmethode BSDM (Business Systems Development Method) binnen IBM. Sinds enkele maanden is BSDM de standaard ontwikkelingsmethode voor de interne automatisering binnen Europa. De andere ontwikkeling betreft de toegenomen aandacht voor object georienteerde systeemontwikkeling.
Gedurende mijn afstudeerperiode heb ik een initieel ontwikkelingsprocess gedefinieerd, dat beide benaderingen integreert.

APPENDICES BSDM - OOD/OOP

August 21st, 1990

P.J.T. Lambregts (Student T.U.E.)

IBM, EBSAT
Uithoorn, The Netherlands

Table of Contents

Appendix A: BSDM Data Modelling

Stage 1: Identify and define entities

This stage consists of the following steps:

1. Identify candidate entities
2. Create working diagrams
3. Verify the entities
4. Create formal diagrams
5. Define the entities

Step 1: Identify candidate entities

An *entity* is a class of thing about which the business wants to keep information (in a database). Entities are written in CAPITALS.
Examples are CUSTOMER, PRODUCT and PLACE.

The *occurrence of an entity* is an acknowledged, individual instance.
Examples are:

1. The CUSTOMER 'Mr.Johnson'
2. The PRODUCT 'Video Recorder'
3. The PLACE 'Amsterdam'

The *population* is all acknowledged instances of an entity.
Example: The population of the entity PERSON is all the people with whom our business deals.

BSDM describes two approaches for identifying the entities. The first approach follows a list of entity families, the second approach uses verb-object pairs.

1. List of likely entity families (with examples)
   - Physical objects (PRODUCT, VEHICLE)
   - People (PERSON)
   - Places (PLACE, COUNTRY)
   - Organizations (BUSINESS)
   - Groupings (CATEGORY)
   - Agreements (EMPLOYMENT CONTRACT, TRADE AGREEMENT, ACCOUNT)
   - Requests (REQUEST FOR DELIVERY)
   - Movements (DELIVERY, INVOICE, ORDER)
   - Assignments (TASKS TO PEOPLE)
   - Memberships (MEMBER OF PRODUCT TYPE)
   - Equivalences or overlaps between other entities
2. Verb - Candidate Entity pairs
   This approach may be used in addition to the first approach, or even in place of it. You work from a rough list of the various business activities carried out in the area being considered. These business activities should be expressed in simple verb-object pairs.

Examples:

- Send INVOICE
- Schedule DELIVERY
- Change ADDRESS

Step 2: Create working diagrams

The most logical links between entities are called *dependencies*. For example, making the entity ADDRESS dependent on PERSON and PLACE expresses the fact that an occurrence of ADDRESS cannot logically exist independently of an occurrence of PERSON and PLACE. ADDRESS is then called a *dependent entity*.

Independent entity	The existence of an occurrence is **independent** on the existence of an occurrence of another (parent) entity.
Dependent entity	The existence of an occurrence is **dependent** on the existence of an occurrence of another (parent) entity.

In the previous step, the first five entity families relate to the independent entities. The other entity families relate to the dependent entities.

In an entity diagram, each entity is represented as a rectangle. The dependencies between the entities are represented by a line. Figure 1 shows a simple entity diagram which shows the entities PERSON, PLACE and ADDRESS with their dependencies.

Figure 1. The entities PERSON, PLACE and ADDRESS

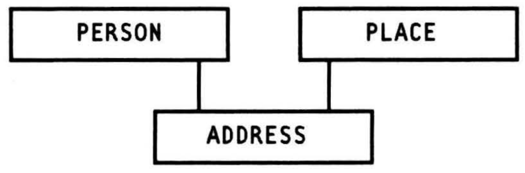


Figure 1 shows some interesting things:

1. Each occurrence of REGISTERED ADDRESS (child occurrence) is dependent on a specific occurrence of PERSON (parent occurrence) and dependent on a specific occurrence of PLACE (parent occurrence).
2. An occurrence of the entity REGISTERED ADDRESS:
   - Is dependent on one occurrence of PERSON
   - Is dependent on one occurrence of PLACE
3. There may be:
   - Any number of occurrences of PERSON
   - Any number of occurrences of PLACE
   - Any number of REGISTERED ADDRESS occurrences for a given occurrence of PERSON
   - Any number of REGISTERED ADDRESS occurrences for a given occurrence of PLACE

Because BSDM does not formally distinguish between entities and relationships, a dependent entity may indicate a relationship between entities !

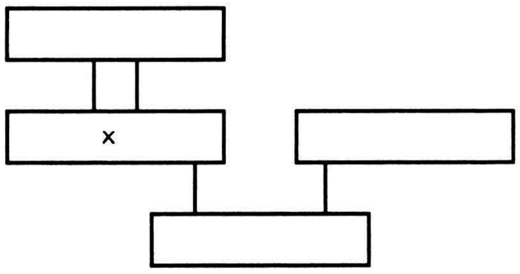
There are some different variations of dependency within the diagrams of BSDM. I will make here a distinction between dependency aspects and dependency structures.

1. Dependency aspects

   a. Levels of dependence

A dependent entity may itself be a parent and have further dependents.

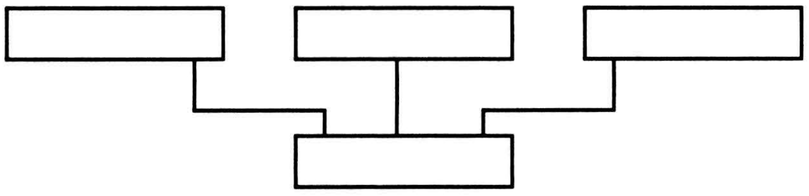
Figure 2. **The dependent entity 'x' may itself be a parent entity**



b. Multiple dependence
A given type of entity may have more than one dependent entity.

c. Number of parents
Although most of the dependent entities are dependent on two parents, a dependent entity is not required to have two and only two parents.

Figure 3. **A dependent entity has one or more parent entities**



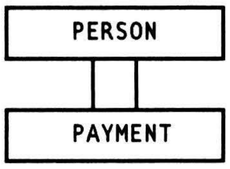
2. Dependency structures

a. Simple structure
This structure was shown in figure 1. The dependent entity in this structure relates two parent entities which each other.

b. Single dependency
An entity is dependent on one parent entity.

c. Double dependency
Double dependency means that each occurrence of the dependent entity is dependent on two occurrences of the same parent entity. The dependent can also have other parents.

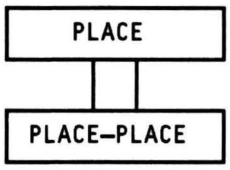
Figure 4. **PAYMENT is double dependent on PERSON:**



d. Involution
1) Each occurrence of the dependent entity is dependent on two occurrences of the same parent entity.
2) Occurrences of the independent entity form new (de)composed groups.

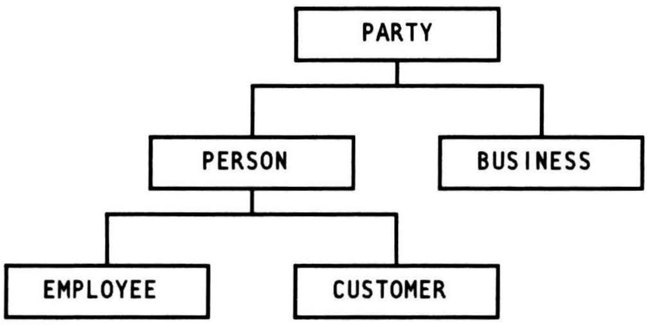
3) The dependent does not have other parents
4) Involution is double dependence, double dependence is not necessarily involution.

Figure 5. Involution of PLACE:



e. Generalisation An entity can be represented at different levels of generalisation.

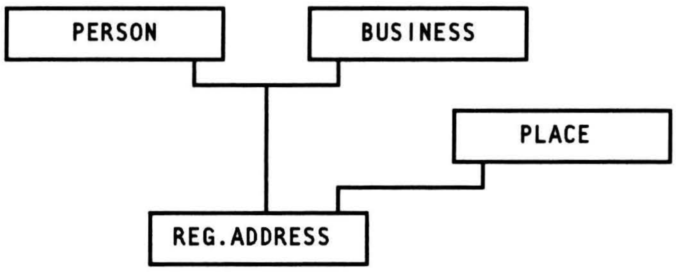
Figure 6. The generalisation structure of EMPLOYEE and CUSTOMER



The independent entity PARTY in this structure is the top level entity of this generalisation structure. The other entities in this structure are generalized entities. The BSDM entity diagram shows only the top level entity of a generalisation structure.

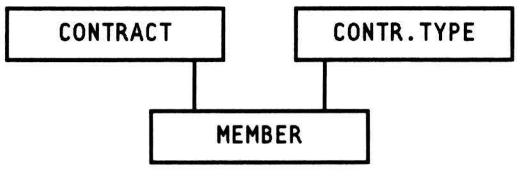
f. Alternative parents Some dependent entities may have alternative parents. This is shown in the BSDM diagram by joining the potential parents with a branched dependency line.

Figure 7. REGISTERED ADDRESS has alternative parents:



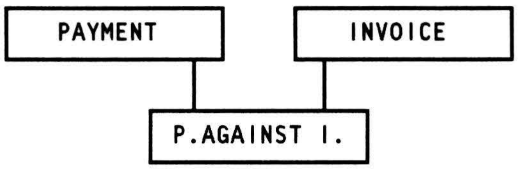
g. Membership (Entity type/Type of entity)

Figure 8. MEMBER OF CONTRACT TYPE:



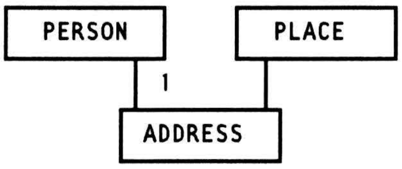
h. Reconciliation
A dependent entity which reconciliates two parent entities. This structure is used for situations like ORDER against RECEIPT, PAYMENT against INVOICE and RESERVATION of a book against LOAN of that book.

Figure 9. Reconciliation of PAYMENT AGAINST INVOICE:



i. Single occurrence per parent
While there may be any number of occurrences of a dependent entity for a given occurrence of any of its parents, there is sometimes a clear business rule that there may be no more than one. In this cases it is helpful to mark the dependency line with a '1'.

Figure 10. A person may have no more than one address



Step 3: Verify the entities This step consist of the following activities:

1. Each entity will be checked against some criteria:

   - Does it have a singular, definite name?
   - Is it possible to give concrete examples of the occurrences of the entity?
   - Will every occurrence of the entity be related to one, and only one, occurrence of each of its parent entities?

2. The entity diagram is verified for supporting the business's 'real world'.

   - Re-examine the list of rejected entities to ensure that earlier ideas on what should and should not be entities still hold good.
   - If necessary, arrange interviews with users to cover areas of particular difficulty or doubt.
   - Use any existing information on current data problems to see whether those which are traceable to existing data structures would be removed if the model's structure were adopted.

3. Challenge each area with a series of "what if?" questions, for example:

- What if an employee leaves? Does the data model imply that his or her responsibilities disappear when he or she does?
- What if a supplier, which is also a customer, changes his address? Will the address of the customer also change?

4. Check that things which have unique keys in existing files and databases are adequately represented in the model.
5. Review the model with users.

For more details see the manual 'BSDM Business Model - Data'

Step 4: Create formal diagrams

1. Convert the working diagrams to formal versions
2. Check that:
   - All entities are correctly and consistently named
   - Dependencies are named where helpful
   - Cross-page references are completed
3. Assign entity numbers

Step 5: Define entities

In this step an entity form of standard layout will be completed for each entity. For more details see the manual 'BSDM Business Model - Data'.

Stage 2: Identify and define attributes

Step 1: Identify candidate attributes

An *attribute* is a property of an entity which the business may be interested in knowing. Attributes are attached only to entities.
Example: The attributes of the entity PERSON are Name, Address and Phonenumber.

An *occurrence: of an attribute* is the individual instance of an attribute.
Example:

1. The Name 'Mr.Johnson'
2. The date 'December 19th'
3. The product number 'T236885'

A *derivable attribute* is an attribute of which the value can be obtained from one ore more other (derivable) attributes:

- Of the same entity
- Of another entity

In this first step of stage 2, an entity attribute list is completed for each entity, showing both normal and derivable attributes.

Step 2: Review attributes

In this step the (candidate) entities are reviewed. Multiple values have to be eliminated and missing entities have to be find. The approach is similar to the process of normalisation. The approach of reviewing each attribute involves asking two questions:

1. Do we want multiple values for the same entity occurrence?
2. Do we expect the same value for multiple occurrences?

If the answer to either of these questions is 'yes' there may be one or more missing entities.
In practice, according to the manual, the process of applying these questions becomes automatic.

Step 3: Define attributes

In this step an attribute form of standard layout is completed for each attribute. For more details see the manual.

Appendix B: BSDM Process Modelling

The process model is developed from the data model. In this part the individual business processes will be identified. A business process is a specific activity which is determined by the nature and objectives of the business, is of defined scope, and is expressed in terms that do not assume any specific implementation or information system support. The relations between the entities and processes are summarised in a process-entity matrix. This matrix shows which processes use and affect a given entity and which entities are used or affected by a given process. The following stages will need to be followed:

1. Establish system boundaries
2. Establish process boundaries
3. Define processes
4. Define attribute processing

Stage 1: Establish system boundaries

Step 1: Document system scope

The first step is to outline the scope of the system in terms of the entity types which are used or affected by it, distinguishing between those which the system is intended to:

1. Reference (R)
2. Change (C)
3. Originate (O)

To produce such a list the attributes should first be marked with a R, C or O.

- If at least one attribute is marked with 'O', mark the entire entity with 'O'.
- If no attribute is marked with 'O', but at least one with 'C', mark the entire entity with 'C'.
- If no attribute is marked with 'O' or 'C', but at least one with 'R' mark the entire entity with 'R'.

The *active scope* of the system defines which parts of the data model, entity types and attributes, can be affected by the system which is build.

Step 2: Identify overlaps with other systems

In this step the areas are identified where the use of entities by other systems overlaps that of the one being considered.

Stage 2: Establish process boundaries

The main activity in this stage is to identify, name, and bind individual 'significant processes'. This stage consists of two steps:

Step 1: Identify processes

The basic procedure of this step:

1. Start with an independent entity originated by the system
2. What process has as its primary purpose the origination of this entity?
3. Give the process a verb-object name.
4. Consider what new facts of interest to the business become available in the real world when this process is performed.
5. Mark the active scope of the process on the entity diagram.
6. Consider each of the entity's attributes and ask what process within the system has as its primary purpose the making of a 'significant' change to its value.
7. Repeat steps 3-5 for each such 'change process'.
8. Repeat all steps for every entity the system originates or changes.

Step 2: Summarise Process-Entity Interactions

In this step, a process-entity matrix is made. Such a matrix shows in a compact form:

1. Which entities are affected or used by a given process, and how.
2. Which processes affect or use a given entity, and how.

The matrix makes it easier to see things like:

- Whether every entity is originated by some process
- Which processes originate more than one entity
- Which entities are originated by more than one process

The manual gives some instructions about completing and adjusting the matrix.

Stage 3: Define processes

Processes are documented in terms of:

1. The triggers by which the process may be activated (step 1).

2. The process rules for the origination or changing of entities (step 3).

3. Significant business considerations to be taken into account (step 3).

In BSDM, a process form should be completed for each process. Instructions are given in the manual.

Step 1: Identify triggers

A trigger is an action or event which cause a process to start. A trigger may originate either inside or outside the business. Some typical examples are:

- Request for enrollment from a person
- Return of a video tape from a customer
- A certain date

A trigger is considered to be external to the processes they activate. Once initiated by a trigger a process may have several legitimate outcomes because its execution is subject to other conditions. Like processes themselves, a trigger should be expressed in general terms, but may be implemented in a variety of ways. Sometimes, more than one trigger has to be present at the same time, for the process to be executed.

Step 2: Document business rules

A form of 'structured English' is recommended for documenting process rules, supplemented with decision tables where necessary. For a more detailed explanation, see the manual.

Step 3: Document business considerations

In this step the business considerations have to be documented. Business considerations have to do with things like criticality, security, problems with current systems, objectives of new systems, etc.

Stage 4: Define attribute processing

This step is to add to the attribute definition forms (created during the data modelling phase) two kinds of rules:

1. Value rules, which state what values an attribute may have
2. Derivation rules, which state how the values are derived for an attribute.

Step 1: Document value rules

The possible values for an attribute can often be considered in isolation from other attributes. Such values may need periodic review to check their continuing validity in the light of changes in factors such as inflation. Sometimes the components parts of an attribute's value must be considered in relation to one another, and when this is so it is best to use a decision table.

Step 2: Document derivation rules

Some attributes may take their values directly from input while others may be derived from one or more other attributes. Attribute derivation rules apply only to the second case. Some attributes are so closely linked that if a value for one of them is altered then so must the value(s) of one or more occurrences of another at the same time.
Example: STOCK Reserved and STOCK Unreserved.

Appendix C: BSDM Architecture

This appendix is a summary of the BSDM manual 'BSDM Architecture'

Introduction

Purpose of Architecture

The purpose of an architecture is to provide a high-level framework within which applications and data can be developed.

Background

There are three main approaches for developing computer supported information systems:

1. Non-integrated approach.
   Designing individual systems and files to meet the local needs of any department which wants to have (new) Information Systems support. This 'unplanned' approach brought a number of problems which grew as more and more of these local systems were added.
2. Totally integrated approach.
   This approach is aimed at creating a single 'corporate management information system', which would cover all business's activities and all the data it needs about these activities. Problems of this approach:
   - Unmanageability of (too) large project
   - Number of existing systems that would require changing
3. Partial integration.
   This is the approach of BSDM

The architecture provides a high-level framework within which The architecture study makes it possible to establish a realistic degree of integration, and to identify the appropriate boundaries and interfaces in support of that integration.
The benefits to business of having an architecture are:

- It helps to ensure that IS is producing applications that fit the needs of the business as a whole.
- It provides IS with a means of ensuring coherence of IS applications and data bases.
- It enables future IS planning to operate within a clearly-defined defined framework.

Once created, an IS architecture should be used on a continuous basis to ensure the coherence of application and data base designs.

Nature

An IS architecture focuses on those applications and data bases that support the control and operation of the day-to-day processing of the business. An IS architecture is developed in three main phases:

1. Define entities and processes.

The needs of the business as a whole are established in terms of entities and processes. This phase comprise stage 1 and stage 2.

2. Group processes and entities.
Processes and entities are clustered into groups. This phase comprise stage 3 and stage 4. Includes: Criteria to determine the grouping.

3. Define data and applications architecture
The architecture consists of two separate, but highly inter-related architectures:

   - Data Architecture
   The data architecture corresponds closely to the entities defined previously.
   - Applications Architecture
   The applications architecture corresponds closely to the processes defined earlier. This phase comprises stage 5.

Basic dilemma's

There are two basic dilemma's of building an architecture:

1. Scope of architecture.
How do we set the scope of an architecture study so that it (a) covers a scope which is sufficiently wide (maybe the whole business) to ensure the main benefits of integration of data and processes while (b) still representing an achievable target for implementation.
2. Degree of physical integration
Can we assure a proper balance between: (a) the need to integrate data so that it can be more easily shared and (b) the need to divide data and processes into small enough units to allow efficient development and operation.

Concepts

The basis of an information systems architecture is the nature and structure of the business it is intended to serve. This structure must be defined as part of the architecture.
Examples of business models:

- Organization chart
- Financial statement
- Entity model
- Process model

An entity model describes the things about which the business needs to keep data. Processes, describe the things the business does. Within BSDM, the processes are defined against the entity model.
Although the entities are defined first, the formal work on processes may well cause a revision of the work on entities.

Data integration: Types of data integration

There are two types of data integration:

1. Logical integration, this involves ensuring that all data within the given scope conforms with one consistent set of definitions.
Although it may be difficult to achieve logical integration fully, it is always desirable.
2. Physical integration, this involves ensuring that there is only a single copy of any given piece of data within the scope.
Too little physical integration means there may be an unnecessarily large number of copies of data and a great deal of effort expended on establishing and controlling interfaces.
Too much physical integration carries the risk that the resulting mass of data may be inflexible, vulnerable to breakdown, and unnecessarily difficult to access.

Data integration: Levels of data integration

1. Unintegrated data
   The elements of data may exist in multiple versions which are in use at the same time, which are not necessarily kept in step with one another, and which may not even be consistently defined.

2. Integrated data
   Every element of data within the set of records has not only a single agreed definition, but the data is also held in one place and entered from only one place. There is only one active master copy of a given piece of data at any one time.
   In many cases, and in particular for large businesses, full integration of a large number of processes and entities, is virtually impossible to control and manage successfully.

3. Partially integrated data
   This is the compromise between the previous two levels. The data is logically integrated but are divided, physically, into groups. These groups:

   - May be stored and operated on different computers at different locations.
   - Are, individually, of manageable size.
   - Are, individually, fully integrated.
   - May be stored and operated on different computers at different locations.

Data access and data sharing

There are four basic ways in which a process may act upon the data stored about a given entity occurrence:

1. Create a record
2. Update a record
3. Read the master version of a record
4. Read (but not necessarily the master version)

There is at one point in time, a master version and one or more copies. These copies are created and/or maintained from the master.
Any process that requires to create or update data about an entity occurrence must have access to the master version of that data (data integrity!).
In some cases where read access only is needed a process may still need access to the master version, because the most up-to-date information is required. In other cases a master version may not be needed, such as in the situation of 'business delay' and 'acceptable delay' (see also the manual: page 11).

Concurrent master access

One of the main criteria for deciding how processes should be grouped is to consider those that need master access and concurrent access to the same data.

Master access	A process needs to create, update or read the master version.
Concurrent access	Several processes need access to stored data about the same entity occurrences at the same time.
Concurrent master access	Processes that need master access and concurrent access to the same data.

Types of data sharing

There are four ways in which data relating to the occurrences of a given entity may reasonably be available to multiple users at a time:

1. Concurrent master access. One integrated copy of the data, shared by all processes needing access.
2. Multiple copies. Each sharing process can have access to a copy. This is useful where the same data is required by more than one group of applications and where the data is created in one of these groups only.
3. Partitioning the entity's population. Occurrences of an entity are separated into different groups. This is particularly useful where the same process is carried out at several places on a section of the entity population.
4. Splitted attributes groups. Attributes are splitted into different groups. Splitting may be used when different processes each need access to non overlapping sets of attributes.

Stages of Systems Architecture

Stage 1: Develop a Business Model

Entities can be defined independently of processes, whereas processes should be defined in terms of the entities upon which they operate. This stage consists of three steps. First the entities are identified and defined. Second the processes are identified and defined in terms of the entities upon which they operate. Third, the entities and processes are summarised in a process entity matrix.

Step 1: Define Entities Define entities as described in the manual 'IBS/DM Business Model - Data'. In the development of an architecture it is recommended that, as the need is to cover a wide scope of the business (perhaps all of it) that the level of detail is limited to defining entities. If attributes of these entities can easily be identified then this should be done. However, no significant effort should be expended in identifying and defining describing individual attributes.
The outputs of this phase should therefore include entity diagrams and entity definitions. In addition, lists should be kept of how the entity model differs from the way the business is currently run and from the way current computer systems operate.

Step 2: Define Business Processes Build a process model as defined in 'IBS/DM Business Model - Processes'. However, it is not necessary in an architecture study to define the process rules in the rigorous fashion the manual describes. It should be sufficient to define processes by means of a narrative description, as opposed to structured English or decision tables.

Step 3: Create Process-Entity Matrix Build a process-entity matrix according to the method described in the manual 'IBS/DM Business Model - Processes'.

Stage 2: Tabulate Business Needs

This step involves tabulating the business's needs from the model in preparation for developing the architecture itself. This stage consists of four steps.

Step 1: Create Location Table In this step the physical locations are identified, at which processing may take place and/or data may be stored. If it is known that data distribution is not required in the foreseeable future this step may be skipped or postponed.

1. A list is made of the major functional units in the business (see manual).
2. Functions which are not split geographically and which share the same place as other functions are grouped together. Thus a location is considered here as a group of business functions which are, and are likely to continue to be, performed at the same type of place.
3. The results are recorded in a Location Table, which is shown in the manual.

Step 2: Create Entity Table A list of entities is made. For each entity the number of attributes and the number of occurrences is estimated.

Step 3: Create Process Table A list of processes, in sequence of process identifiers, is created. This table provides a base of data about each process needed for reference during the subsequent architectural work.

Step 4: Review Process-Entity Matrix In this step the entities and processes are reviewed. The process-entity matrix is updated accordingly.

Stage 3: Define Minimum Process Groups

Step 1: Identify Candidate Minimum Process Groups In this step processes are separated into groups so that any which require master access to the same entity are in the same group.

Step 2: Verify Minimum Process Groups The purpose of this stage is to examine each process group and adjust to avoid, as far as possible:

- Unnecessary grouping and
- Groups serving several types of location

First the original Process Table is rearranged and divided to correspond with the groups in the matrix. Then each group is reviewed for:

1. Concurrent master access
2. Location type

Step 3: Establish Partitioning for Multiple Locations This stage is relevant only when distributing data is being considered. This stage establishes the necessity for, and possibility of, partitioning data across multiple location occurrences (individual locations as distinct from location types).

Stage 4: Establish Maximum Process and Data Groups

The purpose of this stage is to establish the process groups which represent the final decision regarding the data architecture, that is the committed level of data integration. This is defined as the maximum process groups. In some cases, this may mean just one process group (full physical integration). In most cases, the number of maximum process groups will exceed one.

Step 1: Identify Candidate Maximum Groups

Step 2: Establish Control Mechanisms In this step the architecture is reviewed for mechanisms to ensure consistency of duplicated data.

Stage 5: Translate to Outline Data Bases and Applications

Step 1: Translate Date to Outline Data Bases

Step 2: Translate Processes to Outline Applications

Step 3: Review against Physical Constraints

Appendix D: The case study 'Tiger Video'

Introduction

Part of the assignment is an BSDM case study. I made some requirements, concerning a situation to be modelled with BSDM:

1. Description of a business situation.
   Because BSDM will be used for business situations, the description must be of a business situation.
2. Known (business) situation.
   The case study has to be a business situation which is well known to most of the readers. This allows the reader to concentrate quickly on the modelling aspects and not on the description of a business situation which is hardly known.
3. Shows aspects of BSDM.
   Showing the aspects of BSDM is more important than showing the described business situation in full detail with all exceptions that might occur. Therefore, when there was a trade-off, the modelling aspects where more important.

The case study is about an imaginary videotheque, called Tiger Video.

Description of Tiger Video

The videotheque, Tiger Video, sells and hires out video tapes to customers. These tapes can be used on the video systems VHS, BETAMAX and V2000. Only members of Tiger Video can hire tapes. Enrollment is free. For the loan of video tapes, only cash payment is accepted. A member can be a person, but a member can also be an enterprise. The owner, Mr.Black, wants an optimal service for his customers. One of the services of Tiger Video is the reservation of video tapes. Customers can reserve video tapes for free. Customers reserve tapes for a certain date and period.
Tiger Video has a catalogue which contains the titles and descriptions of the tapes which customers can hire. Because a lot of the reservations relate to the newest tapes, Mr.Black likes to have an automated system for the catalogue, where he can store the most recent tapes.
Tiger Video has one establishment. Mr.Black has contracted for this establishment two full-time employees and one part-time employee.
Some competitors have also videoboxes for hire, but Mr.Black does not want to do this. His motivation is that the possession of video recorders is growing and there are already enough competitors which hire out video boxes. Mr.Black has another idea to take competitive advantage. He has a plan to hire out compact discs.

The new automated system, which will be developed with the method BSDM, must register:

1. Enrollment of customers.
2. Hiring out of video tapes.
3. Reservation of video tapes.
4. Payments from customers.

The following areas fall outside the scope of this case:

- Payment to the employees.
- Physical location of tapes in the videotheque.
- VCRs which are used in the videotheque.
- Purchase of video tapes.

Aspects like these can be included when the case will be extended.

Appendix E: Data Model of Tiger Video

Stage 1: Identify candidate entities

The first thing which should be done is to produce a list of candidate entities. This list contains suggested entities. Therefore, this list is not definite. If needed, later on, entities can be removed and other entities can be added.

The BSDM manual gives a list of likely entity families. First the likely independent entities are described, then the likely dependent entities.

Step 1: Identify candidate entities

Physical objects.

The physical objects of the Video Store are videotapes, VCRs, television, chairs, the carpet, etc.. Looking to the description of Tiger Video, the entity VIDEO TAPE seems to be a likely candidate. This entity compromises:

1. Blank video tapes and
2. Movie-tapes.

The VCRs, television and so on are not of our interest.
Mr.Black has some ideas of hire out compact discs in the future. Therefore, the entity COMPACT DISC is identified.

Identified entities: VIDEO TAPE and COMPACT DISC.

When this is done for all products, we have to identify more entities. Imagine identifying the entities for a big corporation like IBM. All the products together will lead to a large amount of entities. Every time a new product is announced, a new entity is needed! In other words, every time a new product is announced, the data model must be adjusted! This is not wanted. Therefore, BSDM generalises entities. For products, BSDM defines the entity PRODUCT. Every time a new product is announced, the data model remains stable.

--> Identified entity: PRODUCT.

People.

The people which are of interest to Tiger Video are the customers, the employees and the suppliers. The entities which can be defined are CUSTOMER, SUPPLIER and EMPLOYEE.

Identified entities: CUSTOMER, SUPPLIER and EMPLOYEE.

In this situation, BSDM generalises these entities to PERSON.

--> Identified entity: PERSON.

Places.

The Video Store wants to track the addresses of the customers. Therefore, the entities PLACE and REGISTERED ADDRESS are identified. REGISTERED ADDRESS is dependent on PLACE.

--> Identified entities: PLACE and REGISTERED ADDRESS.

These two entities provide much more flexibility than having one entity REGISTERED ADDRESS. For example, when the name of a street changes, this has to be done once. With one entity, this has to be done for all the members that live in that street.

Organizations.

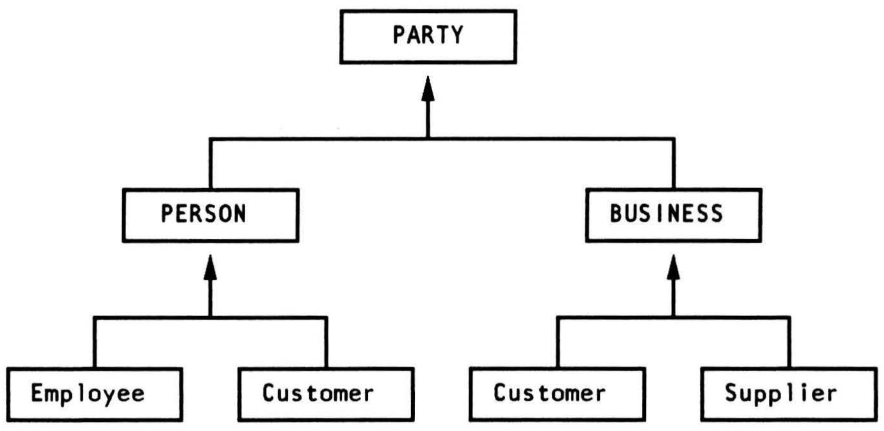
For organisations like:

- Tiger Video,
- Suppliers and
- Enterprises which hire tapes,

the entities VIDEO STORE, SUPPLIER and CUSTOMER can be identified. BSDM generalises organisations to the entity BUSINESS.

But the entity CUSTOMER was already identified as people. So, CUSTOMER belongs to the PERSON and to BUSINESS! To eliminate this construction, PERSON and BUSINESS will be generalised to PARTY. The following figure shows this generalisation structure, where the entity CUSTOMER is generalised to PERSON and to BUSINESS.

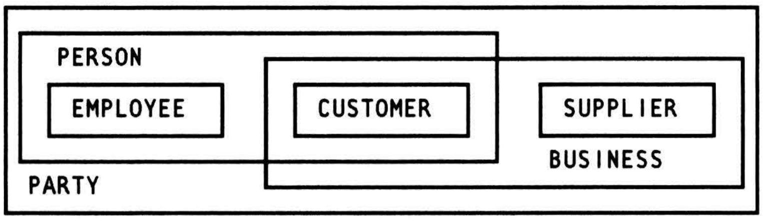
Figure 11. Generalisation structure



--> Identified entities: PARTY.

Another way to depict the generalisation structure is shown in the following figure. Only the outer boundary (PARTY) has a corresponding entity in the BSDM entity diagram.

Figure 12. Representation of the generalisation structure



Groupings.

When you are walking into a videotheque, you see a lot of tapes which are divided into categories, like Action, Horror, Humor, Science Fiction and so on. Because it is something of the real world where the Video Store wants to keep information about, you define an entity for it, the entity CATEGORY.

Another grouping is the grouping of products into groups which shows the type of the products, the entity PRODUCT TYPE.

--> Identified entities: CATEGORY, PRODUCT TYPE.

Identified likely independent entities:

1. PARTY (Includes: CUSTOMER, SUPPLIER, TIGER VIDEO, EMPLOYEE).
2. PRODUCT (Includes: VIDEO TAPE, COMPACT DISC).
3. PLACE.
4. PRODUCT TYPE
5. CATEGORY

Agreements.

An agreement can be for example:

- a contract between Tiger Video and a supplier.
- a contract between Tiger Video and a customer.
- a contract between Tiger Video and an employee.

Therefore, the entities SUPPLIER CONTRACT, CUSTOMER CONTRACT and EMPLOYEE CONTRACT can be identified. These entities will be generalised to CONTRACT.
Another kind of agreement is LOAN OF PRODUCT and RESERVATION OF PRODUCT TYPE.

--> Identified entities: CONTRACT, LOAN OF PRODUCT and RESERVATION OF PRODUCT TYPE.

Requests.

A request can be for example a PURCHASE ORDER of video tapes or a REQUEST FOR RESERVATION.

--> Identified entities: PURCHASE ORDER and REQUEST FOR RESERVATION.

Movements.

This can be a SHIPMENT from the supplier to the Video Store. But it can also be a PAYMENT IN for the Video Store and a PAYMENT OUT from the Video Store.

--> Defined entities: SHIPMENT, PAYMENT IN and PAYMENT OUT.

Memberships.

PRODUCT IN PRODUCT is the entity which indicate that the Video Store creates a new product by composing different products. The entity MEMBER OF PRODUCT TYPE indicates that a certain tape is a member of a certain PRODUCT TYPE.

--> Identified entities: MEMBER OF PRODUCT TYPE, PRODUCT (DE)COMPOSITION PRODUCT TYPE (DE)COMPOSITION.

Equivalences or overlap between entities.

These are not identified.

Summary of likely entities

The following list of likely entities is identified.

Independent entities:

1. PRODUCT
2. PRODUCT TYPE
3. PLACE
4. CATEGORY
5. PARTY

Dependent entities:

1. CONTRACT
2. LOAN AGAINST RESERVATION
3. MEMBER OF PRODUCT TYPE
4. PAYMENT AGAINST LOAN OF PRODUCT
5. PAYMENT IN
6. PAYMENT OUT
7. PLACE-PLACE
8. PRODUCT
9. PRODUCT (DE)COMPOSITION
10. PRODUCT TYPE
11. PRODUCT TYPE (DE)COMPOSITION
12. PURCHASE ORDER
13. REGISTERED ADDRESS (See Places.)
14. REQUEST FOR RESERVATION
15. RESERVATION OF PRODUCT TYPE
16. SHIPMENT

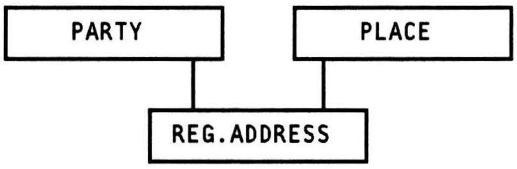
Step 2: Create working diagrams

In this step simple diagrams are made of related entities. These simple diagrams will be the starting-point for the whole data model of Tiger Video.

PARTY, PLACE and REGISTERED ADDRESS (Simple structure)

This diagram shows the three entities PARTY, PLACE and REGISTERED ADDRESS related to each other.

Figure 13. The entities PARTY, PLACE and REGISTERED ADDRESS:



1. PARTY

   - Name : Johnson
   - Person/Business : Person
   - Phone number : 02975-1234
   - Current amount of tapes : 4

2. REGISTERED ADDRESS

   - Start date : 19/19/89
   - ZIP code : 1020DS
   - Number : 16
   - End date : -

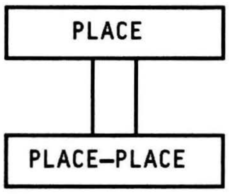
3. PLACE

   - Name : Dam Square
   - Type : Street

PLACE, PLACE-PLACE (Involution)

This simple diagram shows an example of involution. As described in the BSDM report this means double dependency where occurrences of the parent entity form new (de)composed groups. Occurrences of the entity PLACE include names like street names, names of districts and names of countries. Involution here means that a set of street forms in fact another occurrence of this entity, a place. A set of different places forms in fact a country, which is also an occurrence of the entity PLACE. The following figure shows this relationship in a simple entity diagram.

Figure 14. The entities PLACE and PLACE-PLACE



1. PLACE

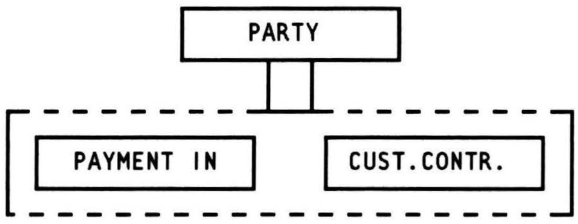
• Name	: (1) Amsterdam (2) Holland
• Type	: (1) City (2) Country

2. PLACE-PLACE

• Name1	: Amsterdam
• Name2	: Holland
• Type	: City-Country

PARTY, PAYMENT IN, CUSTOMER CONTRACT (Double dependence)

Figure 15. The entities PARTY, PAYMENT IN and CUSTOMER CONTRACT



1. PARTY

• Name	: Johnson
• Person/Business	: Person
• Phone number	: 02975-1234
• Current amount of tapes	: 4

2. PAYMENT IN

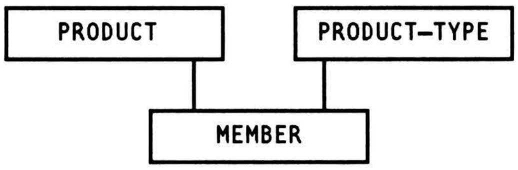
• Amount ($)	: 9,15
• Date	: 24/12/89
• Reason	: Hire of 3 tapes

3. CUSTOMER CONTRACT

• Start date	: 01/12/89
• Date cancelled	:01/12/90

The entities PRODUCT, PRODUCT-TYPE and MEMBER (Membership)

Figure 16. A membership structure



1. PRODUCT

• Name	: Police Academy1 on VHS
• Purchase price ($)	: 49,95
• Current state	: Excellent

2. PRODUCT TYPE

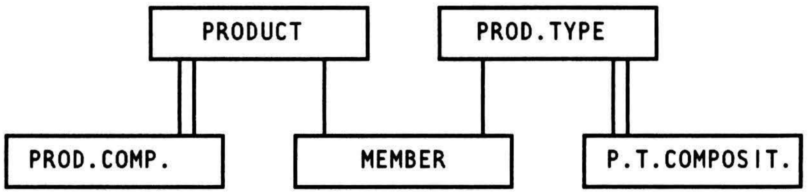
• Name	: Police Academy 1 on VHS
• Supplier	: Video Store 'Big Blue'
• Current price ($)	: 45,05

3. MEMBER (OF PRODUCT TYPE)

• Start date	: 19/06/89
• End Date	: -

PRODUCT (DE)COMPOSITION and PRODUCT TYPE (DE)COMPOSITION

Figure 17. Two involution structures



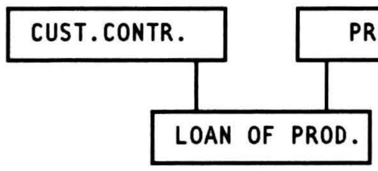
The following example will illustrate the use of the entities PRODUCT COMPOSITION and PRODUCT TYPE COMPOSITION.

Example:
Tiger Video hires out the tapes JAWS1, JAWS2 and JAWS3. But Mr.Black might decide to hire out these tapes together as one product. This 'new product' consists of three existing products. In fact, we composited a 'JAWS123' out of three existing products. To show this relation BSDM uses the involution structure. Do not forget that when Tiger Video has hired out the package 'JAWS123', the individual tapes 'JAWS1', 'JAWS2', and 'JAWS3' cannot be hired out.

This composition might also be done on PRODUCT TYPE level.
When the supplier of video tapes creates a new product which is composed of existing product types, the entity PRODUCT TYPE COMPOSITION is needed.

CUSTOMER CONTRACT, PRODUCT and LOAN OF PRODUCT (Reconciliation)

Figure 18. Reconciliation structure: LOAN OF PRODUCT



1. CUSTOMER CONTRACT

• Start date	: 19/06/89
• End Date	: -

2. PRODUCT

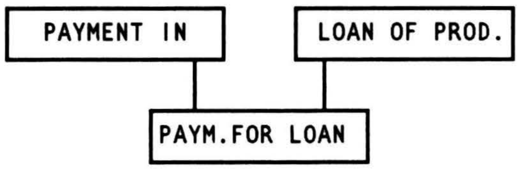
• Name	: Police Academy1 on VHS
• Purchase price ($)	: 49,95
• Current state	: Excellent

3. LOAN OF PRODUCT

• Date of issue	: 24/12/89
• Expected return date	: 27/12/89
• Actual return date	: 29/12/89
• Penalty period	: 2 days (Derivable attribute !)
• Penalty charge ($)	: 2,00

PAYMENT IN, LOAN OF PRODUCT and PAYMENT FOR LOAN

Figure 19. Reconciliation structure: PAYMENT FOR LOAN



1. PAYMENT FOR LOAN

• Date of allocation	: 24/12/89
• Date cancelled	:-

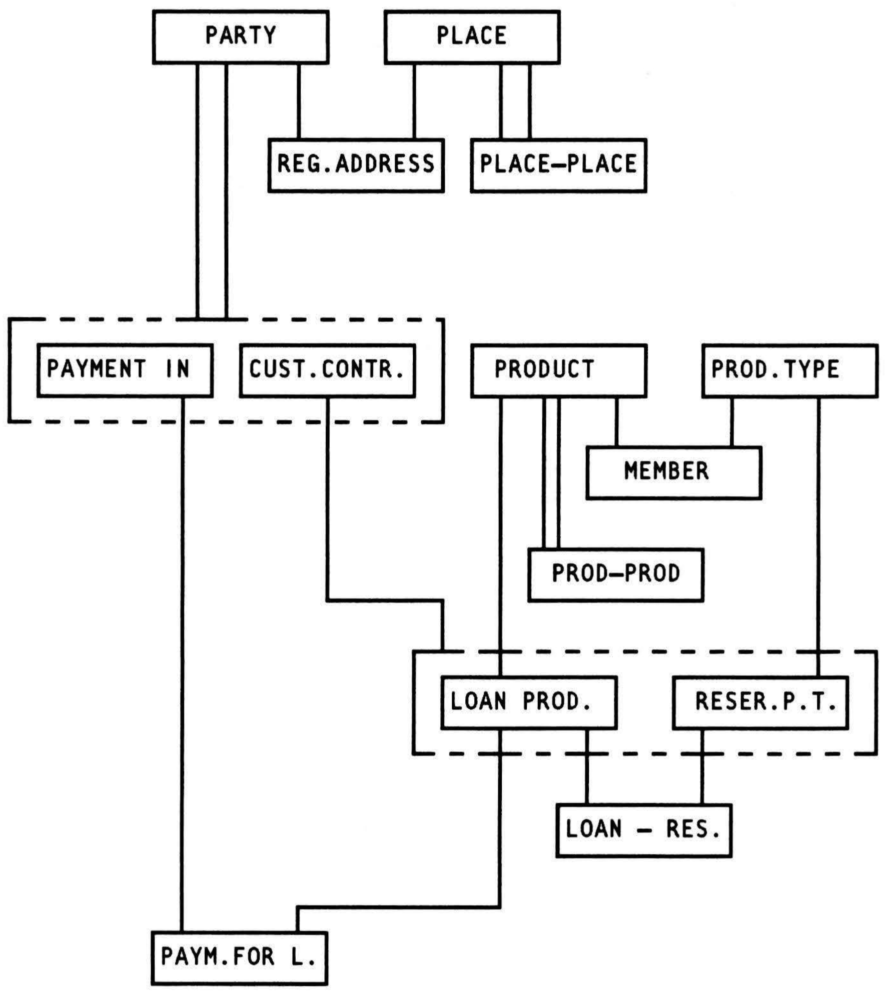
Step 3: Verify the entities

One of the ways to verify the entities is to give concrete examples of the occurrences of the entities. Occurrences of entities are described in the previous step.

Step 4: Create formal diagrams

This is my solution for the case study. Other solutions are possible.

Figure 20. Entity diagram of 'Tiger Video'



Step 5: Define entities

Define independent entities

1. PARTY
The parties which are of interest to Tiger Video.
Includes: Customers, Suppliers, Employees and Tiger Video.

2. PLACE
Geographical areas that are of interest to Tiger Video.
Includes: Streets, districts and cities.

3. PRODUCT
Items which Tiger Video hires out or sells.
Includes: Blank Video Tapes, Movie Tapes, Compact Discs.

4. PRODUCT-TYPE
The type of a product.

Define dependent entities

5. REGISTERED ADDRESS
The address of a customer which is registered by Tiger Video.

6. PLACE-PLACE
Place (de)composition.

7. PRODUCT-PRODUCT
Product (de)composition. Composes products into one product or decomposes a product into several products.

8. PAYMENT IN
The amount of money, which Tiger Video receives for:
    - Hiring out video tapes
    - Receiving tapes, which are hired out, too late (penalty charge)
    - Selling of products

9. CUSTOMER CONTRACT
A contract between Tiger Video and the customer. In this contract are stated the terms of hiring video tapes which are accepted by both Tiger Video and the customer.

10. MEMBER OF PRODUCT TYPE
Relates a product to its product type.

11. LOAN OF PRODUCT
Loan of a specific video tape by the customer.

12. RESERVATION OF PRODUCT TYPE
Record the request from parties to reserve a certain product type on a certain date.

13. LOAN AGAINST RESERVATION
Allocates the loan of certain products against the reservation of their product type.

14. PAYMENT FOR LOAN
Allocates payments from customers to their customer contract.

Stage 2: Identify and define attributes

Step 1: Identify candidate attributes

1. PARTY (I)

   a. Name
   b. Person/Business
   c. Initials
   d. Number of passport
   e. Number of driver's licence
   f. Contact Person
   g. Phone number
   h. Credit limit
   i. Credit status
   j. Current amount of tapes

2. PLACE (I)

   a. Name
   b. Suffix
   c. Type (country, town,etc.)

3. PRODUCT (I)

   a. Name
   b. Description
   c. Purchase price
   d. Purchase date
   e. Obsolete date
   f. Current state

4. PRODUCT TYPE (I)

   a. Name
   b. Description
   c. Supplier
   d. Current price
   e. Current delivery time

5. REGISTERED ADDRESS (D)

   a. Start date
   b. Premises
   c. ZIP code
   d. End date

6. PLACE-PLACE (D)

   a. Name1
   b. Type1
   c. Name2
   d. Type2
   e. Startdate
   f. Enddate

7. PRODUCT-PRODUCT (D)

   a. Original product
   b. Derived product

c. Start date
d. End date

8. PAYMENT IN (D)

a. Amount
b. Date
c. Reason

9. CUSTOMER CONTRACT (D)

a. Start date
b. Date cancelled

10. MEMBER OF PRODUCT TYPE (D)

a. Start date
b. End date

11. LOAN OF PRODUCT (D)

a. Date of issue
b. Issue of condition
c. Date of return
d. Return condition
e. Expected return date
f. Actual return date
g. Additional period
h. Additional rental charge
i. Repeat period
j. Repeat rental charge
k. Penalty period
l. Penalty charge

12. RESERVATION OF PRODUCT TYPE (D)

a. Date-time of reservation
b. Requested reservation date
c. Requested reservation period

13. LOAN AGAINST RESERVATION (D)

a. Date-time
b. End date

14. PAYMENT FOR LOAN (D)

a. Date of allocation
b. Date cancelled

Appendix F: BSDM Process Model of Tiger Video

Stage 2: Establish Process Boundaries

Stage 3: Define processes

Step 1: Identify processes

In this appendix, the business processes of Tiger Video are identified and defined and the process attribute matrix is made. In this appendix, stage 2 and stage 3 are combined.
In the description of Tiger Video, the areas of interest are:

1. Enrollment of customers
2. Hiring out of video tapes
3. Reservation of video tapes
4. Payments from customers

Seven business processes will be described and explained.

Process 1: Accept PARTY

This process is needed to enroll new customers, suppliers and employees. But because suppliers and employees fall outside the scope of the case study, this process is only defined for enrolling new customers. To become a customer, you need to have a customer contract. This is a business decision. There cannot be an occurrence of PARTY which does not have a customer contract i.e. which is a prospect.

Purpose: To register a new customer with his or her address and the corresponding customer contract.

In terms of the entities, we have to:

- create a new occurrence of the entity PARTY
- create a new occurrence of REGISTERED ADDRESS
- create a new occurrence of the entity CUSTOMER CONTRACT

Process description in Structured English

```
1.  IF PARTY not exists
       Create new occurrence of PARTY
       Create new occurrence of REGISTERED ADDRESS
       Create new occurrence of CUSTOMER CONTRACT
2.  IF PLACE unknown
       Recognize PLACE (Process 2)
```

Process 2: Register/Recognize PLACE

This process lets you acknowledge something in the real world which already existed: Streets, districts and cities. In fact, we register a place which already exists in the real world.

Purpose: To register a new PLACE (city, district or street).

In terms of the entities we have to create a new occurrence of PLACE.
An occurrence of the entity PLACE can be a city, a district or a street. Because a district or a street means nothing, unless we know the city:

- A district can only be registered if the district is related to a city.
- A street can only be registered if street is related to a district or city.

Process description in Structured English

```
1.  IF PLACE is street
    a.    IF PLACE (district or city) related to street not exist
    b.       Create district or city
    c.       Register street and relate street to district or city

2.  IF PLACE is district
    a.    IF PLACE (city) related to district not exist
    b.       Create city
    c.       Register district and relate district to district or city

3.  IF PLACE is city
    a.    Create city
```

(Change PLACE-PLACE is a trivial change process)

Process 3: Change REGISTERED ADDRESS

If a customer moves to another living place, but still wants to be a member of Tiger Video, Tiger Video has to change the registered address of the customer.

Purpose: To change the REGISTERED ADDRESS of a party.

This process refers to PARTY and creates a new occurrence of REGISTERED ADDRESS. The existing registered address gets an end date. When the PLACE is unknown, it will be created in-flight.

Process description in Structured English

```
1.  IF PARTY not exists STOP.
2.  IF PLACE unknown
      Recognize PLACE (Process 2)
3.  IF PARTY exists
      Change occurrence of REGISTERED ADDRESS
      Add occurrence of REGISTERED ADDRESS
```

Process 4: Reserve PRODUCT TYPE

A customer who wants to have a specific movie for a certain period has the possibility to make a reservation. Tiger Video has to register this reservation. A reservation will be on PRODUCT TYPE level. If Tiger Video has three copies of a certain movie, the customer is not interested in a specific copy. He wants to loan one of them, it does not matter which one.

Purpose: To register a reservation of a product type for a customer for a specific period.
Trigger: Customer requests reservation of tape

A customer reserves for a specific day and period. When all the copies of the reserved product type are reserved or hired out, then the request cannot be accepted.

1. IF product not exists: STOP.
2. IF CUSTOMER CONTRACT not exists: process 1.
3. IF all products of PRODUCT TYPE are registered for specified period as reserved or on loan: STOP.
4. In all other cases:
   Register reservation for product type and contract on specified day and loan period.

Process 5: Establish LOAN OF PRODUCT

This business process is needed to register the loans of tapes and CDs to customers.

Purpose: To register the loan of a product (tape or CD), the payment for this loan and eventual the reconciliation of this loan to its reservation.

Process description in Structured English

```
IF CUSTOMER CONTRACT and PRODUCT exists THEN
  IF all PRODUCTS of PRODUCT TYPE are reserved, AND/OR all products of PRODUCT
TYPE are on loan for requested period: STOP.

  IF product is reserved and available
    originate occurrence of LOAN AGAINST RESERVATION
    originate occurrence of LOAN OF PRODUCT
  IF product available and not reserved
    originate occurrence of LOAN OF PRODUCT
```

Process 6: Add PRODUCT

Purpose: To register products which Tiger Video has purchased and received.
Trigger: Receipt of product

- Add a new occurrence of the entity PRODUCT.
- Add a new occurrence of the entity MEMBER.
- Refer to PRODUCT TYPE
- In flight creation of occurrence of PRODUCT-PRODUCT
  1. To indicate that the new occurrence is part of another product
  2. To indicate that the new occurrence consists of more than one product.

Process description in Structured English

```
1.  IF new product is received
        THEN create new occurrence of PRODUCT and
             create new occurrence of MEMBER
2.  IF new product is part of another product
        create new occurrence of PRODUCT-PRODUCT
3.  IF new product is consists of more than one product
        create new occurrence of PRODUCT-PRODUCT
```

Process 7: Add PRODUCT TYPE

Purpose: To add product types of products (tapes) which are ordered, but of which the product types are not registered. This process adds a new occurrence of the entity PRODUCT TYPE.
Trigger: Ordering of a product.

Process description in Structured English

```
IF PRODUCT TYPE not exists
  Create new occurrence of PRODUCT TYPE.
```

Summarise process-entity interactions

In this step the process entity matrix for Tiger Video is made.

```
14 PAYMENT FOR LOAN............
13 LOAN AGAINST RESERVATION....
12 RESERVATION OF PRODUCT TYPE.
11 LOAN OF PRODUCT.............
10 MEMBER OF PRODUCT TYPE......
09 CUSTOMER CONTRACT...........
08 PAYMENT IN..................
07 PRODUCT-PRODUCT.............
06 PLACE-PLACE.................
05 REGISTERED ADDRESS..........
04 PRODUCT TYPE................
03 PRODUCT.....................
02 PLACE.......................
01 PARTY.......................
```

No.	PROCESS NAME	01	02	03	04	05	06	07	08	09	10	11	12	13	14
010	Accept PARTY..............	O	▶			O	▶			O					
020	Recognize PLACE...........		O				▶								
030	Change REGISTERED ADDRESS..	–	▶			O	▶								
040	Reserve PRODUCT TYPE.......			=	=			=		=	=	=	O		
050	Establish LOAN OF PRODUCT..			=					O	=		O		▶	O
060	Add PRODUCT................			O	▶			O			O				
070	Add PRODUCT TYPE...........				O										
080	....														
090	....														

Figure 21. Process-entity matrix of Tiger Video: The 'O' means originate, the '>' means originate in flight, and the '=' means reference

Appendix G: Smalltalk/VPM

Introduction

Smalltalk/VPM is an object oriented programming system under OS/2 Presentation Manager (PM). There are other versions of Smalltalk, like Smalltalk-80, Smalltalk/V, and Smalltalk/V286. But Smalltalk/VPM is the only version that runs under OS/2 PM.
Smalltalk/VPM requires an 80286, 80386, 80486 computer capable of running either the MicroSoft or IBM implementation of OS/2 with Presentation Manager (PM version 1.1 or beyond).
When you really want to have a good idea what OOP is, you have to do object oriented programming by yourself. Programming in Smalltalk/VPM is a good means to understand the object oriented approach better.

Within Smalltalk, everything is an object! And every object belongs to a class. Included within the Smalltalk environment are a set of system classes. A system class is a class which is part of the class hierarchy when you install Smalltalk. These system classes are the basic building blocks for building Smalltalk applications.
When you want to build applications in Smalltalk, you have to understand very well the system classes. But the system classes are generic, meaning that they can be used for a lot of applications. But it also means that you will need some classes which are not part of the hierarchy. You have to make them by yourself. Making new classes is an important characteristic of Smalltalk programming.
The system classes that come with the Smalltalk package helps the novice programmer to build (simple) applications. But the experienced programmer can and will build its own classes.
The class hierarchy of Smalltalk/VPM contains 137 system classes. The different versions of Smalltalk do have some different system classes. For example, only the Smalltalk/VPM version has classes to communicate with the PM libraries. The name of each class begins with a capital. The class Object is the highest in the hierarchy. So it is the superclass of all classes.

Starting up Smalltalk

Smalltalk is started with the command VPM. When you first run Smalltalk, the transcript window appears. This window is used by the system for various messages and can be used by the programmer as an editor and code evaluation window. This window is the only window that cannot be closed when running Smalltalk.

Exit Smalltalk

To exit Smalltalk, select Exit Smalltalk/V... from the pull-down menu. You will then be asked whether or not to save the image. If you answer yes, again starting up Smalltalk will give the same screen as you left it.

Windows

All the windows in Smalltalk behave conform to PM conventions, like:

- Activate a window
- Deactivate a window

- Open a window
- Close a window
- Resize a window
- Move a window
- Scroll a window

Also the menus conform to PM menu conventions:

- Browse menus
- View a hierarchical sub-menu
- Choose an item
- Keyboard select an item
- Dismiss a menu

Each window has one or more panes. Smalltalk makes extensive use of multiple-pane windows. An important example is the class hierarchy browser. See Figure 22 on page 41.
This multi-paned window shows you the interrelationship of classes and subclasses within Smalltalk. It enables you to edit subclasses (including adding methods and variables). The bulk of application development takes place in this window.

The pane upper left shows the class hierarchy. Class Object is the highest class in the hierarchy, Animated Object is a subclass of class Object, Class Bag is a subclass of class Collection, and the three dots behind a class means that the subclasses are not shown. If you want to see the subclasses, double click on this class.

Class Collection is highlighted.
The pane upper right shows the methods that belongs to the class which is highlighted in the upper left pane.

The method asArray is highlighted. In the lower half, the method implementation of the method asArray is shown.

The other two panes show the class and instance variables that are inherited by its subclasses. In the figure, no variables are inherited by (the instances of) class collection.

Execute code

To evaluate a Smalltalk expression, enter the code into a window or text pane and select it. Then pull down the Smalltalk menu and select Do it (= Ctrl D) or Show it (= Ctrl S) from the menu. When you choose Do it, Smalltalk executes the expression, but does not show the results on the screen. When you select Show it, Smalltalk both executes the expression and returns the result. The returned result is now the selected text, so this can easily be deleted, using Backspace or Cut.

Walkback window

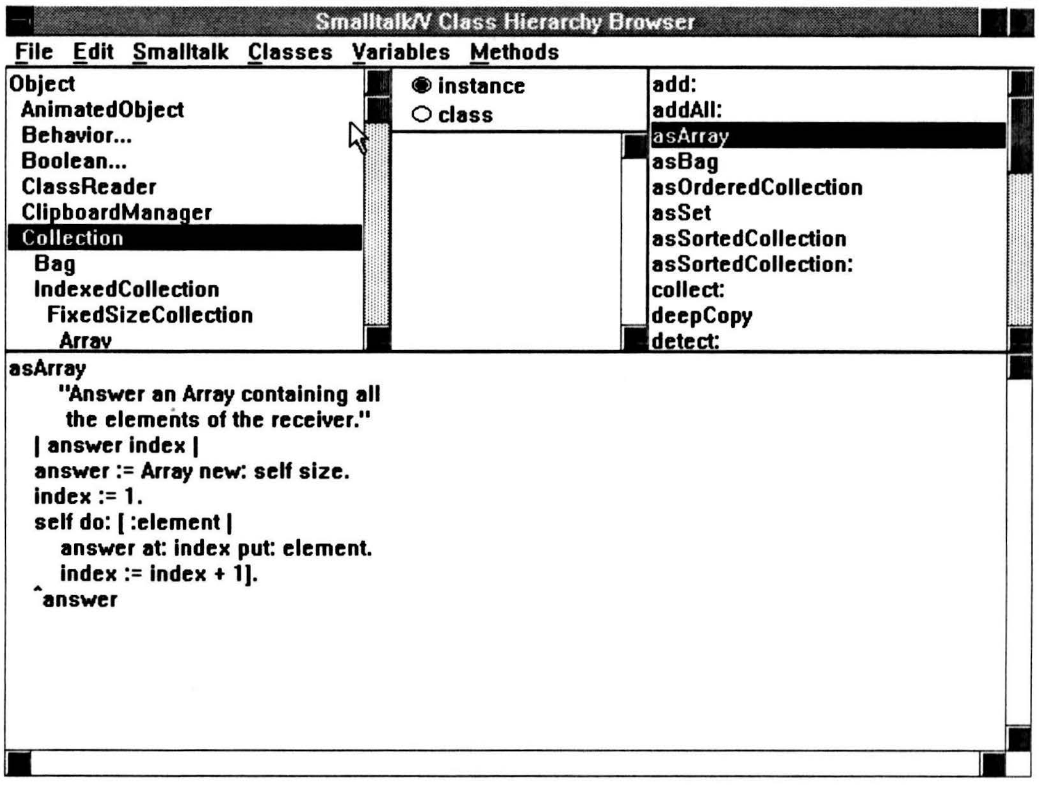
When Smalltalk detects an error, a walkback window pops up automatically. You can either close the window or ask for the debugger window. This window helps you to correct programming errors.

Copy text

The Edit menu provides the possibility to copy selected text. This means that reuse of code is not restricted to inheritance only.

Adding a new subclass

Within the Class Hierarchy Browser, click on the class which is the direct superclass of the new class you want to add. Then, pull down the classes menu and select Add Subclass. Fill in the name of



```
asArray
    "Answer an Array containing all
     the elements of the receiver."
  | answer index |
  answer := Array new: self size.
  index := 1.
  self do: [ :element |
     answer at: index put: element.
     index := index + 1].
  ^answer
```

Figure 22. Class Hierarchy Browser

the new class and press enter. You now have added the new subclass, which is shown in the window. After the class is added to the hierarchy, its methods and variables can be added.

New and new:

Objects are mostly created with the message new or new:. When an object is created, it also has its instance variables, but not the values of these instance variables.

Variables

There are different kind of variables that can be added:

- Global variables
  The value of a global variable is known in the whole system. Global variable names always begin with an upper case letter. When you introduces a new global variable, you get a dialog box allows you whether or not you want to create it. Assignment statements are used to assign values to global variables.
- Instance variables
  The value of an instance variable belongs to one instance of a class. All the instances of a class have the same instance variables. But every instance has its own values to the instance variables.
- Class variables
  Within the class, there is one value for a class variable. This value can be accessed by all the instances of that class. Class variables are used to share data within a class. They begin with a capital letter.
- Temporary variables
  Temporary variables exist only within the execution of a method. Smalltalk discards them as soon as you are done using them. Temporarily variables are declared by closing them in ver-

tical bars in the first line of a method implementation. The name of a temporarily variable must start with a lower case letter. Assignment statements are used to assign values to temporarily variables.
- Pool variables
  These variables are shared by the instances of a subset of the classes in the system.

Class and instance variables are inherited by the subclasses.

Methods

- Class methods
  Class methods respond to messages sent to class objects, rather than to instances of a class.
- Instance methods
  Instance methods respond to messages sent to the instances of a class.

Adding a new method

Within the Class Hierarchy Browser, select New Method from the Methods menu. A method 'template' appears in the bottom pane of the class hierarchy browser, that contains:.

- messagePattern
  This is a message pattern, where the instances of this class can respond to. All the message patterns of one object together, is the interface of that object.
- comment
  Comments are stated between double quotes and are ignored when compiling.
- temporaries
  Temporary variables are identified here.
- Statements
  This is the actual code that is executed when an instance of the class receives a message that corresponds to the message pattern.

Messages

There are:

- Unary messages
  Methods without arguments.
- Keyword messages
  Methods with one or more arguments.
- Binary messages
  Messages that consist of one or two numerical characters, and (mainly) will be used for arithmetic purposes.

Sending a message involves:

1. Identifying the object to which the message is sent (the receiver of the message).
2. Identifying the additional objects that are included in the message (the message arguments).
3. Specifying the desired operation to be performed (the message selector).
4. Accepting the single object that is returned as the message answer.

Smalltalk 'Application Development Cycle'

Within the Smalltalk manual, the iterative Smalltalk 'Application Development Cycle' is described.

1. State the problem

Application development begins by describing your problem and what the software should do. The purpose is to have a short, understandable statement of the problem which points toward the kind of solution that will address your need.

2. Draw the window
   Draw the window that will help you solve your problem. The drawing is a specification of the application user interface and is the starting point for specifying the rest of the application.
3. Identify the classes
   List both the new classes defined for the application and existing classes from the Smalltalk hierarchy to be used.
4. Describe the object states
   Identify the instance variables.
5. List the object interfaces
   Object interfaces are the messages that the object responds to.
6. Implement the methods

Appendix H: Smalltalk/VPM System classes

Class Object

Class Object is the highest class in the hierarchy. Class Object defines the protocol common to all objects. It defines the default behavior for displaying, comparing, copying, and inspecting objects, and error handling. Some examples of instance methods are:

- class
  Answer the class of the receiver.
- inspect
  Open an inspector window on the receiver.
- isKindOf: aClass
  Answer true if the receiver is an instance of aClass or one of its subclasses, else answer false.
- isMemberOf: aClass
  Answer true if the receiver is an instance of aClass, else answer false.
- sender
  Answer the sender object that invoked the current method.
- = anObject
  This is the default equality test. Answer true if two objects represent the same object, else answer false.
- = = anObject
  This is the default equivalence test. Answer true if two objects are the same object, else answer false.

Collection classes

A part of the hierarchy of system classes is depicted in Figure 23. Each class inherits the functionality of all its superclasses in the hierarchy.

Figure 23. Part of the hierarchy of system classes

```
Object
   Collection
      Bag
      IndexedCollection
         FixedSizeCollection
            Array
            String
      Set
```

- Class Collection is the superclass of all the collection classes. It is an abstract class defining the common protocol for all its subclasses. An abstract class is a class which cannot have any instances, but only specifies protocol for its subclasses. Collections are the basic data structures used to store objects in groups in either a linear or nonlinear fashion. Class Collection provides the protocol to directly access or store a particular element in a collection or to access all the elements of a collection in a particular order.

- Class Bag is a collection of unordered elements in which duplicates are allowed.
- Class Set is like class Bag, except that duplicates are not allowed.
- Class IndexedCollection is an abstract class providing the common protocol for all the indexable collection subclasses.
- Class FixedSizeCollection is an abstract class for all the indexable fixed size collections. A fixed size collection cannot grow or sink, hence elements cannot be added or removed from it. Only the element values can be changed.
- Class Array is a collection of any objects accessed through a fixed range of integer indices. Most of the protocol to handle arrays is inherited from its superclasses.
- A String is a fixed size indexable sequence of characters (ASCII codes from 0 to 255). This class provides the protocol to compare strings, replace characters within the string and to convert characters to upper or lower case.

Although class Collection is an abstract class, instances can be defined! This should not be possible. The following example, demonstrates this.

Figure 24. Execute with 'show it'

```
| cars |    "Temporarily variable"

verzameling := Collection new   'verzameling becomes instance'
verzameling class   'Asks the class of instance verzameling'
```

This little program starts with making cars a temporarily variable. Then cars is made an instance of the class Collection. To demonstrate that an instance of class Collection is created, the object car is sent the message class. Because class Collection has not implemented a method for this message, Smalltalk will look in the direct superclass of class Collection if a corresponding method is implemented. This direct superclass is class Object. This class has implemented this method, as we have seen. Smalltalk executes this method and the answer is: Collection. Although in theory abstract classes do not have instances, in Smalltalk they can !

Creating objects

Objects are mostly created with the message new or new:. The message new: must have an argument added. For example, creating the string 'This is a string' will be done as follows:

```
| sentence |    "Comment: make sentence a (temporarily) variable"
sentence := String new.   "Make sentence an instance of class String"
sentence := 'This is a string'   "Assignment statement"
```

The string is an instance of the class String. Creating a new array is similar to the previous example:

```
| abcd |
abcd := Array new.
abcd := #(A B C D)
```

The array #(A B C D) is an instance of the system class Array. This array consists of four characters, which are instances of another system class, the system class Character. These examples shows that an object (String or Array) can consist of other objects (Characters).

Messages and methods

To every class belongs a set of methods. There are class methods and instance methods.

Class methods implement the messages sent to the class. The receiver of a class message is always the class itself, not an instance of the class.
Instance methods implement messages sent to instances of the class. The receiver of an instance message is always an object that is an instance of a class.

For example, some instance methods of class String are:

- asLowerCase,
  Answers a string, containing the receiver with alphabetic characters in lower case.
- asUpperCase
  Answer a string, containing the receiver with alphabetic characters in upper case.
- at: anInteger
  Answers the character at position anInteger in the receiver string.
- at:anInteger put:aCharacter Answer a character.
  At index position anInteger in the receiver put the character aCharacter.
- size
  Answers the size of the receiver string.
- = aString
  Answer true if the receiver is equal to aString, else answer false.

These methods will be demonstrated by the following examples:

```
| sentence |   "Temporarily variable sentence"
sentence := String new.

sentence := 'This is a string'.   "Assignment statement"
Answer: aString   "Every method results in an answer"

sentence size.   "Sends the message size to the object sentence"
Answer: 16   "The result is the size of sentence"

sentence asUpperCase
Answer: 'THIS IS A STRING'

sentence asLowerCase.
Answer: 'this a string'

sentence at:2.
Answer: $h

sentence = 'This is a string'.
Answer: TRUE
```

Messages in a protocol description are described in the form of message patterns. A message pattern contains a message selector and a set of argument names.
For example, the message pattern at:anInteger put:aCharacter matches the messages described by (for example):

- at:9 put :I.
- at:10 put :B.
- at:11 put :M.

The arguments are 9, 10, 11, I, B and M. Messages with one or more arguments are called keyword messages. A message with no argument is called an unary message, like size, asLowerCase and asUpperCase.

Magnitude classes

Class Magnitude is an abstract class used for comparing, counting, and measuring instances of its subclasses. The figure shows the subclasses of class Magnitude.

Figure 25. Class DialogBox and some of its subclasses

```
Magnitude
  Association
  Character
  Date
  Number
    Float
    Fraction
    Integer
      LargeNegativeInteger
      LargePositiveInteger
      SmallInteger
    Time
```

Instances of class Date represents a particular day since the start of the Julian calendar. Class Date defines the protocol for creating, comparing, and computing dates. The protocol of the superclass Magnitude can be used to compare different dates.

Dialog Box classes

Figure 26. Class DialogBox and its subclasses

```
DialogBox
  FindReplaceDialog
  FontDialog
  MessageBox
  NewSubclassDialog
  Prompter
```

Class MessageBox

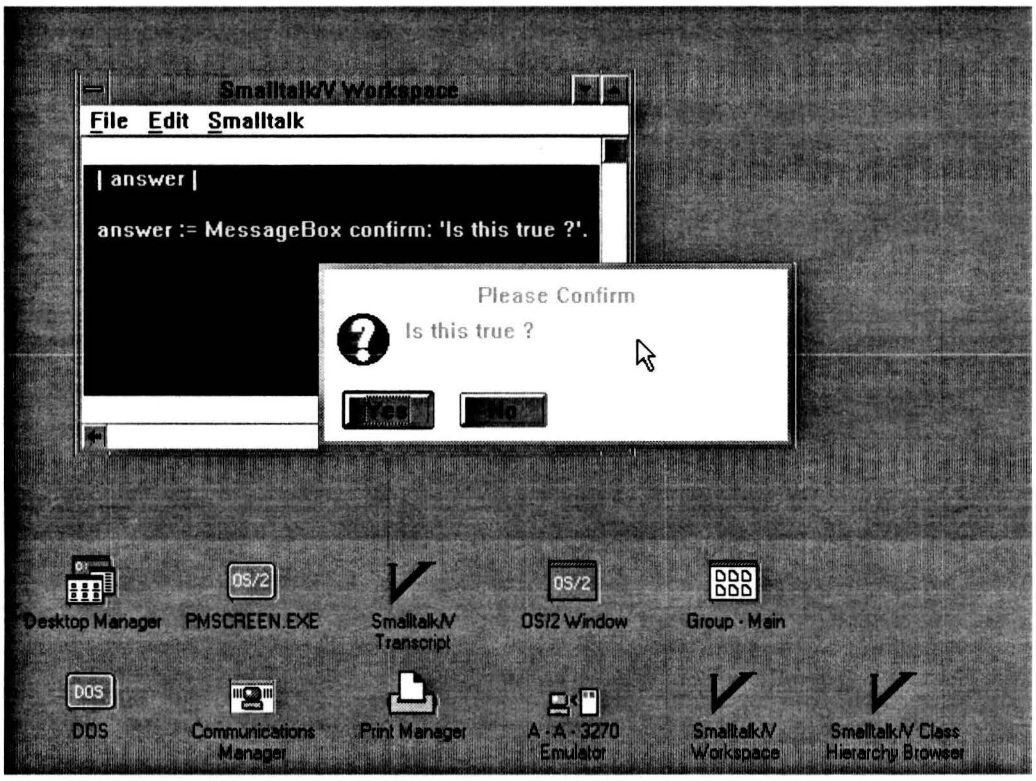
Class MessageBox is used to create a PM Message Box. A message box is a dialog box with a limited set of options and requires a simple acknowledgement or choice by the user.

The following example creates a messagebox that asks for a confirmatiom for something. The variable 'answer' becomes true or false.

```
| answer |

confirmation := MessageBox confirm: 'Is this true?'
```

Figure 27. Message Box example


```
| answer |

answer := MessageBox confirm: 'Is this true ?'.
```

Please Confirm

Is this true ?

Figure 28. Message Box example

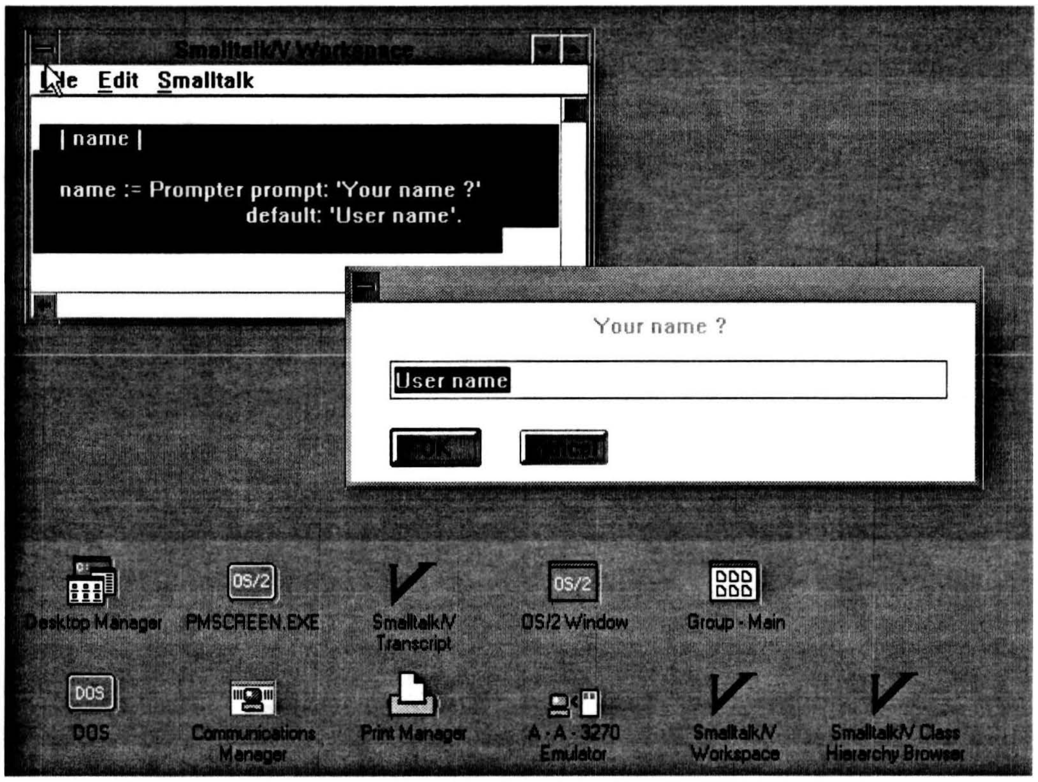
Class Prompter

Instances of class Prompter are dialog boxes which allows an application to pose a question and solicit an answer from the user. Depending on which class method is used to invoke a Prompter, the output from the Prompter can be either the entered string from the user or the object that is the result of evaluating the entered string.

```
| dialog |

dialog := Prompter prompt:'Your name?' default:'User name'
```

Figure 29. Prompter example



```
| name |

name := Prompter prompt: 'Your name ?'
                default: 'User name'.
```

```
User name
```

Figure 30. Prompter example

This first example shows a dialog box that asks the name of the user. The default name is 'User name', but can be any other name. It is also possible to give it no name, only the '' are then needed.

Class Dictionary

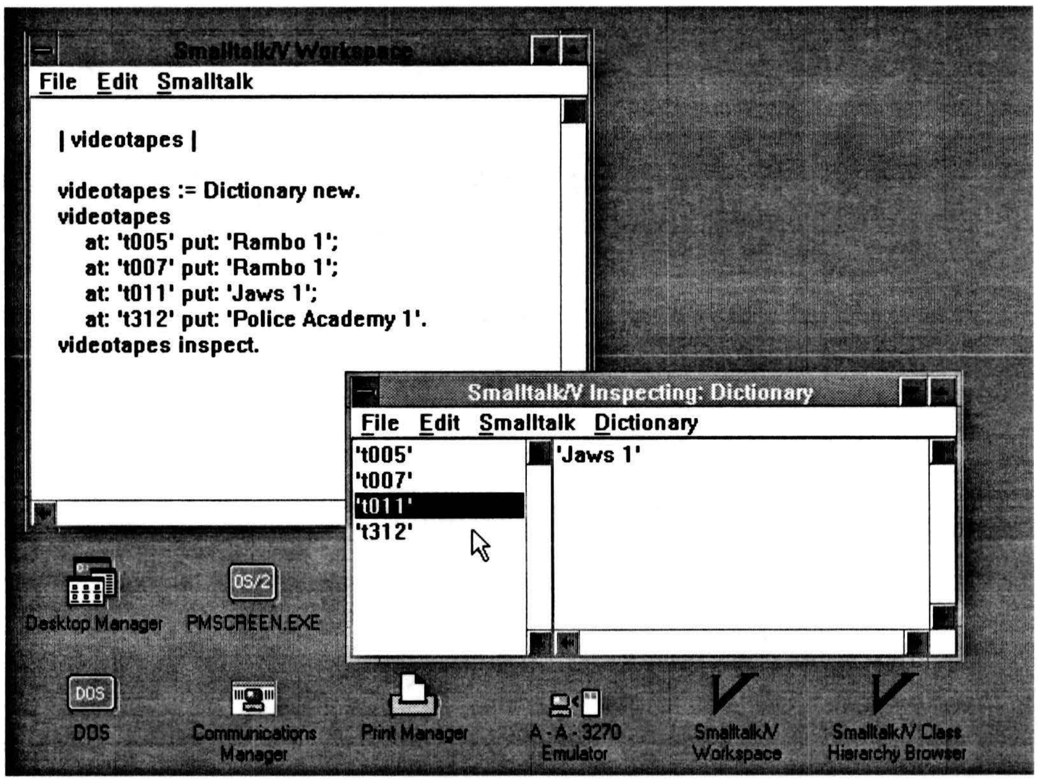
One of the system classes of Smalltalk/VPM is class Dictionary. This class is a direct subclass of class Set, and a direct superclass of class IdentityDictionary.

Figure 31.

```
Collection
  Set
    Dictionary
      IdentityDictionary
```

Instances of class Dictionary are dictionaries. A dictionary is a collection of key/value pairs. The keys in a dictionary are unique, whereas values may be duplicated. A Dictionary may be searched either by key or by value.

In the following example, the keys identify videotapes with a specific movie.



```
| videotapes |

videotapes := Dictionary new.
videotapes
   at: 't005' put: 'Rambo 1';
   at: 't007' put: 'Rambo 1';
   at: 't011' put: 'Jaws 1';
   at: 't312' put: 'Police Academy 1'.
videotapes inspect.
```

```
't005'        'Jaws 1'
't007'
't011'
't312'
```

Figure 32. Dictionary example

The example shows duplicated values, which is allowed. The keys are unique.

Class Dictionary has the following instance methods:

- at :aKey
  Answers the value of the key/value pair whose key equals aKey from the receiver.
- at:aKey put:anObject
  If the receiver contains the key/value pair whose key equals aKey, replace the value of the pair with anObject. Else add the key/anObject pair.
- includes: anObject
  Answer true if the receiver contains the key/value pair whose value equals anObject, else answer false.
- includesKey: aKey
  Answer true if the receiver contains aKey, else answer false
- keys
  Answer a Set containing all the keys in the receiver.
- values
  Answer a Bag containing all the values of the key/value pairs in the receiver.

Each instance of class Dictionary is a collection of key/value pairs. The keys in a dictionary are unique, whereas values may be duplicated. A Dictionary may be searched either by key or by value.

Class Dictionary provides some interesting methods that can be used.

- videotapes includesKey: 't011'
  Answers true if MovieTapes contains the key 't011'. The answer will be true.
- videotapes values
  Answers a bag containing all the values of MovieTape. The answer is: Bag(Rambo 1, Rambo 1, Jaws 1, Police Academy 1)
- MovieTapes keys

Answers a set containing all the values of MovieTape. The answer is: Set (t005, t007, t011, t312)

- at:'t05'
  Answers the value which corresponds with the key 't005'
- at:'t08' put:'Jaws 1'

Class Window and ApplicationWindow

Class Window is an abstract class which provides the common data and protocol that are inherited by all the window subclasses.

Figure 33. Class Window and some of its subclasses

```
Window
   ApplicationWindow
      ClassBrowser
      ClassHierarchyBrowser
      Inspector
        Debugger
        DictionaryInspector
      MethodBrowser
      TextWindow
   ControlWindow
   DialogBox
   SubPane
```

Class ApplicationWindow is a subclass of class Window, it is also an abstract class and provides the common protocol for all application windows. It includes methods for opening application windows and for coordinating the subpanes of an application window.

Appendix I: Object oriented design for 'Tiger Video'

In this appendix, object the transition is demonstrated for the case study 'Tiger Video'. Appendix F contains the description of this case study.

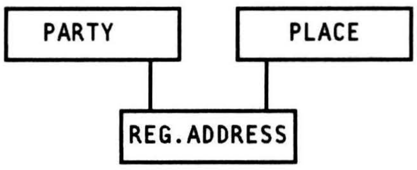
Step 1: Identify dependency structures

The dependency structures are described in appendix A.

These structures can simply be identified. Some of the structures overlap. The 'i' means that this entity is an independent entity ***within*** the dependency structure. The 'd' means that this entity is the dependent entity ***within*** the dependency structure.

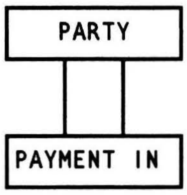
- Simple Structure (S)
  1. PARTY(i) and PLACE(i) with REGISTERED ADDRESS(d)

Figure 34. Simple Structure 1



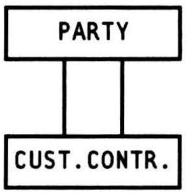
- Double dependency (D)
  1. PARTY(i) with PAYMENT IN(d)

Figure 35. Double dependency Structure 1



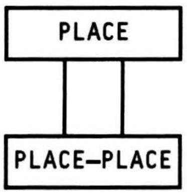
  2. PARTY(i) with CUSTOMER CONTRACT(d)

Figure 36. Double Dependency Structure 2



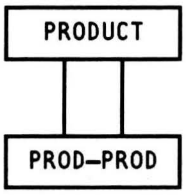
- Involution (I)
  1. PLACE(i) with PLACE-PLACE(d)

Figure 37. Involution Structure 1



  2. PRODUCT(i) with PRODUCT-PRODUCT

Figure 38. Involution Structure 2



- Generalisation (G)
  In the entity diagram of Tiger Video, there are three generalisation structures: PRODUCT, PARTY and PLACE.
  1. PLACE: Street, City, District
  2. PARTY: Customer, Supplier, Employee

Figure 39. Generalisation structure 1

  3. PRODUCT: Compact Disc, Video Tape

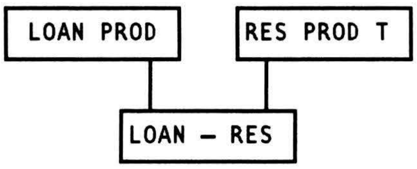
Figure 40. Generalisation structure 2

(These structures are shown in the following diagram.)

1. PERSON(i) with SUPPLIER(n) and EMPLOYEE(n) and CUSTOMER(n)
2. PRODUCT(i) with VIDEOTAPE(n) with MOVIETAPE(n)

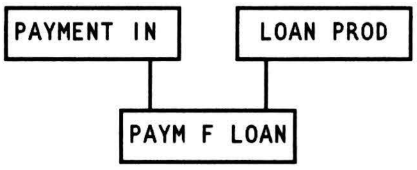
- Reconciliation (R)
  1. LOAN OF PRODUCT(i) and RESERVE PRODUCT TYPE(i) with LOAN AGAINST RESERVATION(d).

Figure 41. Reconciliation 1



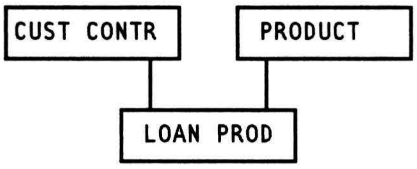
2. PAYMENT IN(i) and LOAN OF PRODUCT(i) with PAYMENT AGAINST LOAN.

Figure 42. Reconciliation 2



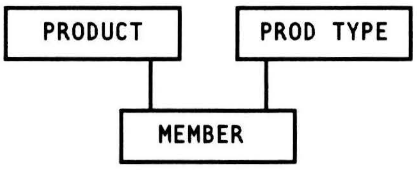
3. CUSTOMER CONTRACT(i) and PRODUCT(i) with LOAN OF PRODUCT(d).

Figure 43. Reconciliation 3



- Membership (M)
  1. PRODUCT(i) and PRODUCT TYPE(i) with MEMBER(d)

Figure 44. Membership Structure 1



To have a clear overview, I made the entity - dependency structure diagram. This diagram shows the involvement of entities within dependency structures. For each entity can easily be seen in which dependency structures it is involved. For each dependency structure can easily be seen which entities it contains.

```
14 PAYMENT FOR LOAN............
13 LOAN AGAINST RESERVATION....
12 RESERVATION OF PRODUCT TYPE.
11 LOAN OF PRODUCT.............
10 MEMBER OF PRODUCT TYPE......
09 CUSTOMER CONTRACT...........
08 PAYMENT IN..................
07 PRODUCT-PRODUCT.............
06 PLACE-PLACE.................
05 REGISTERED ADDRESS..........
04 PRODUCT TYPE................
03 PRODUCT.....................
02 PLACE.......................
01 PARTY.......................
```

id.	Description	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
S01	Simple structure 1	i	i			d											
02	Simple structure 2																
03	Simple structure 3																
D01	Double dependency 1	i							d								
02	Double dependency 2	i								d							
03	Double dependency 3																
I01	Involution 1		i				d										
02	Involution 2			i				d									
G01	Generalisation 1	i															
02	Generalisation 2			i													
03	Generalisation 3																
R01	Reconciliation 1											i	i	d			
02	Reconciliation 2								i			i			d		
03	Reconciliation 3			i						i		d					
M01	Membership 1			i	i						d						
02																	
03																	

Figure 45. Entity - dependency structure matrix

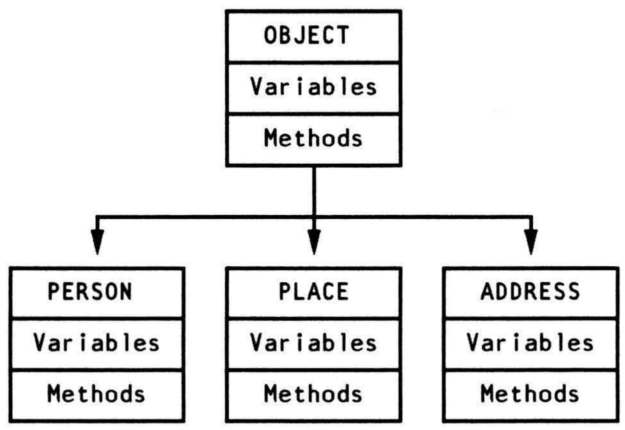
Step 2: Define inheritance structures

For some dependency structures, new classes/entities have to be defined.

Simple Structure

The Simple Structure does not imply any inheritance. Therefore, this structure will be transformed into the following class hierarchy.

Figure 46. Class structure of Simple Structure

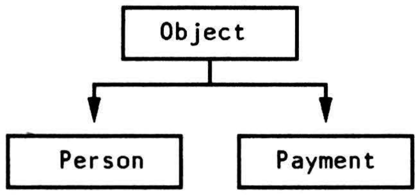


- Simple Structure 1
  PARTY, PLACE and REGISTERED ADDRESS become direct subclasses of class Object.

Double dependency

This structure cannot be modelled with inheritance.

Figure 47. Inheritance structure from double dependence



The entity PAYMENT is double dependent on the entity PERSON.

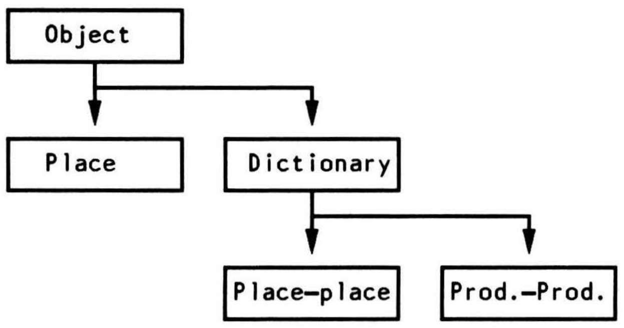
- Double dependency 1
  PARTY and PAYMENT IN become direct subclasses of class Object.
- Double dependency 2
  PARTY and CUSTOMER CONTRACT become direct subclasses of class Object.

Involution structure

This structure cannot be modelled with inheritance between the independent(i) and the dependent(d) entity within the involution structure. The corresponding inheritance structure is the same as the previous two.

But there is inheritance when you use class Dictionary from Smalltalk. Using the Dictionary class from Smalltalk may give some inheritance, which can be the benefit for all involution structures.

Figure 48. Inheritance structure from involution



- Involution 1
  PLACE becomes a direct subclass of class Object. PLACE-PLACE becomes a subclass of class Dictionary.
- Involution 2
  PRODUCT becomes a direct subclass of class Object. PRODUCT-PRODUCT becomes a direct subclass of class Dictionary.

Generalization structure

Build a hierarchical structure of the entities that are generalised. The entity PARTY includes CUSTOMER, SUPPLIER and EMPLOYEE. These entities have to be identified explicitly.

Figure 49. Inheritance structure from generalised entity PARTY

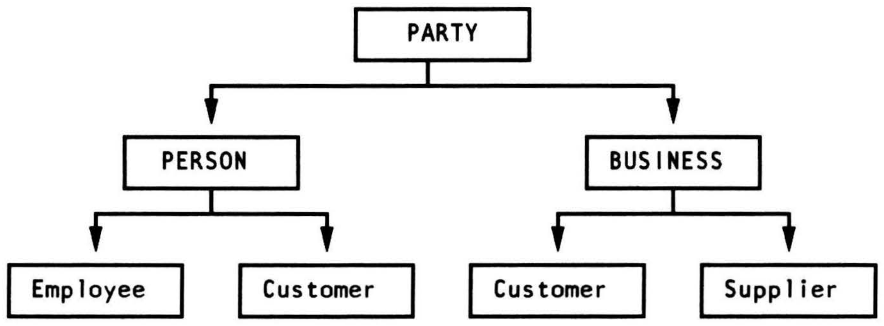


Figure 4 shows the class Customer multiple dependent on class Person and class Business. In this figure can be seen that class Customer is a subclass of two classes: class Person and class Business. This structure needs multiple inheritance!

When the environment where the applications will be implemented does not support multiple inheritance, it is preferable to erase the classes Person and Business. The classes EMployee, Customer and Supplier are then direct subclasses of class Party.

- Generalisation 1
  PARTY becomes a direct subclass of class Object. Subclasses of class PARTY will be the classes Customer, Employee and Supplier.
- Generalisation 2

  PRODUCT becomes a direct subclass of class Object. Classes Videotape, Movietape, etc. become subclasses of class PRODUCT.

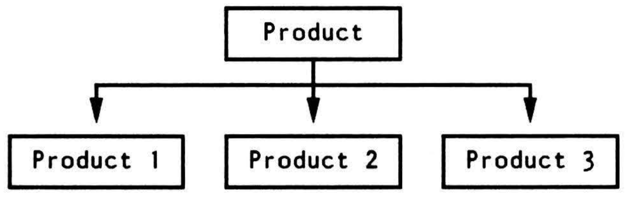
Reconciliation structure

The result is the same as the simple structure.

- Reconciliation 1
  LOAN OF PRODUCT and RESERVE PRODUCT TYPE become direct subclasses of class Object. LOAN AGAINST RESERVATION becomes a direct subclass of class Object, or a direct subclass of class Dictionary/Reconciliation.
- Reconciliation 2
  PAYMENT IN and LOAN OF PRODUCT become direct subclasses of class Object. PAYMENT FOR LOAN becomes a direct subclass of class Object, or a direct subclass of class Dictionary/Reconciliation.
- Reconciliation 3
  PRODUCT and CUSTOMER CONTRACT become direct subclasses of class Object. LOAN OF PRODUCT becomes a direct subclass of class Object, or a direct subclass of class Dictionary/Reconciliation.

Membership structure

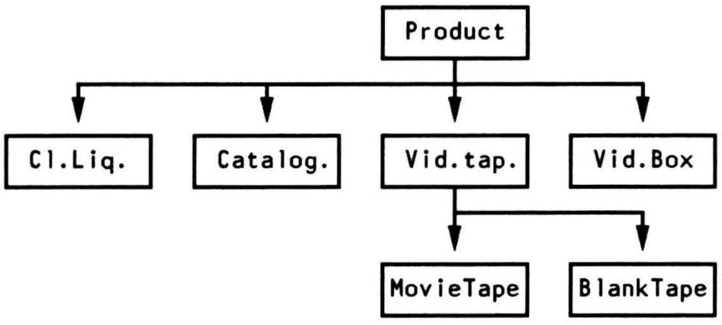
Figure 50. Inheritance from membership structure



The entity (or class) MEMBER is erased because class Product 1 as a subclass of class Product already indicates the membership.

Make attributes of the entity PRODUCT TYPE class variables of the class Product.

Figure 51. Inheritance structure of entity PRODUCT



From the three entities, which are part of a membership structure, the entity MEMBER will disappear. The entity PRODUCT TYPE and PRODUCT will be transformed to the structure PRODUCT with the individual products. Rules are needed, which are similar to generalization.

- Membership structure
  Class PRODUCT will be a direct subclass of class PRODUCT. Class VideoTape will be a direct subclass of class PRODUCT, and class Movietape will be a direct subclass of class Videotape.

Conclusion

We see that there are no conflicting class structures within this case study!

Step 3: Transform attributes into instance variables

For every entity, which is mapped against a class, the attributes will identify the instance variables. An instance variable should be put into the highest possible class. The attributes which are identified as 'derived' attributes within BSDM, should be defined as derived within an OO environment too. The attributes of the classes should be put as high as possible in the hierarchy.

The classes with their instance variables:

1. PARTY (abstract)
   Name, Person/Business, Initials, Contact person, Phone number.
2. PLACE
   Name, suffix, type.
3. PRODUCT
   Name, description, purchase price, purchase date, obsolete date, current state.
4. PRODUCT TYPE
   Name, description, supplier, current price, current delivery time.
5. REGISTERED ADDRESS
   Start date, premises, ZIP code, end date.
6. PLACE-PLACE
   Super, sub, start date, end date.
7. PRODUCT-PRODUCT
   Original product, derived product, start date, end date.
8. PAYMENT IN
   Amount, Date, Reason
9. CUSTOMER CONTRACT
   Title, Description, Start date, Date cancelled.

10. MEMBER OF PRODUCT TYPE
    Not a class.
11. LOAN OF PRODUCT
    Date of issue, issue of condition, date of return, return condition, expected return date, actual return date, additional period, additional rental charge, repeat period, repeat rental charge, penalty period, penalty charge.
12. RESERVE PRODUCT TYPE
    Date-time of reservation, requested reservation date, requested reservation period.
13. LOAN AGAINST RESERVATION
    Date-time, end date.
14. PAYMENT FOR LOAN
    Date of allocation, date cancelled.
15. Object
16. PARTY/Employee Bank account,
17. PARTY/Customer
    Current amount of tapes, credit status, credit limit, Number of passport, number of driver's licence.
18. PARTY/Supplier
19. PRODUCT/Videotape
20. Videotape/Movietape
21. Dictionary/Reconciliation
22. Dictionary/Involution

Remarks

- Class PARTY is the abstract superclass for the classes Employee, Supplier and Customer.
- The attribute 'current amount of tapes' will be an instance variable of the class Customer, because only customers can have an amount of tapes. But if an employee can loan tapes, without registered as a customer, then this attribute should be specified for class PERSON, but overridden for class Supplier. You can also specify this attribute for the classes Customer and Employee.
- Credit status, see 'current amount of tapes'.
- Bank account is used for paying the salary to employees and paying the supplier for delivered tapes.
- PAYMENT IN and PAYMENT OUT can be subclasses of Payment.
- Customer Contract can be the subclass of class Contract. Other subclasses are Supplier contract and Employee contract.

Step 4: Identify methods

Task 1: Determine create methods

Accept PARTY	PARTY	originate PARTY
	PLACE	(originate PLACE)
	REGISTERED ADDRESS	originate REG.ADDR.
	PLACE–PLACE	(originate PLACE–PLACE)
	CUSTOMER CONTRACT	originate CUST.CONTR.
Recognize PLACE	PLACE	originate PLACE
	PLACE–PLACE	(originate PLACE–PLACE)
Change REG. ADDRESS	PARTY	–
	PLACE	(originate PLACE)
	REG.ADDRESS	originate PLACE–PLACE
	PLACE–PLACE	(originate PLACE–PLACE)
Reserve PRODUCT TYPE	PRODUCT	–
	PRODUCT TYPE	–
	PRODUCT–PRODUCT	–
	CUSTOMER CONTRACT	–
	MEMBER OF P.T.	–
	LOAN OF PRODUCT	–
	RESERV. OF P.T.	originate RES.OF P.T.

Task 2 :Identify process attribute interactions

For every business process a process attribute matrix is made. This matrix shows the entities and attributes that are needed for that business process.

The process attribute matrix of the business process Accept PARTY is depicted in the following figure.

Accept PARTY	PARTY (O)	Name	
		Person/Business	
		Initials	
		Passport nr.	
		Driver's licence nr.	
		Phone number	
		Credit limit	
		Credit status	
		Current Amount of T.	
	REGISTERED ADDRESS (O)	Start date	
		End date	
		Number	
	PLACE (▶)	Name	
		Suffix	
		Type	
	PLACE–PLACE	Start Date	
		End date	

Figure 52. Process Attribute Table for Accept PARTY

This matrix shows which entities are originated, changed or referenced by the business process Accept PARTY. It also shows which attributes are needed. The business process Accept PARTY originates occurrences of PARTY and REGISTERED ADDRESS.

Task 3: Identify attribute methods

Accept PARTY	PARTY (O)	Name Initials Passport nr. Driver's licence nr. Phone number Credit limit Credit status Current Amount of T.	assign Name assign Initials assign Passp.nr. assign D.l.nr. assign Phone nr.
	REGISTERED ADDRESS (O)	Start date End date Number	
	PLACE (▸)	Name Suffix Type	
	PLACE–PLACE	Start Date End date	

Figure 53. Process Attribute Table for Accept PARTY

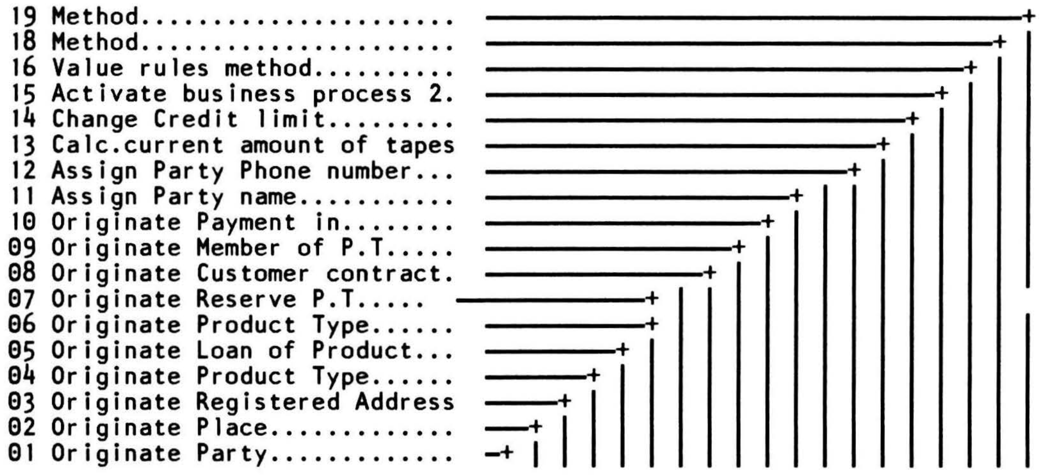
We need methods for creating new objects. We want to create customers, videotapes, reservations, loans, payments, addresses, places, etc as objects. We want to add the values to the instance variables.

In OO terminology, this means that objects of the classes PARTY and REGISTERED ADDRESS must be created. These methods are indicated by a 'c' in the PMG matrix.

The process attribute matrix also shows that values of some attributes have to be assigned, like Name, Passport nr., Credit limit, etc..

When the credit limit of a customer changes, this is done by a change method. When a registered phone number contains an error, this error is corrected by a change method.

The calculations of derived attributes are done by calculate methods. An example is the calculation of the current amount of tapes.



```
19 Method......................
18 Method......................
16 Value rules method..........
15 Activate business process 2.
14 Change Credit limit.........
13 Calc.current amount of tapes
12 Assign Party Phone number...
11 Assign Party name...........
10 Originate Payment in........
09 Originate Member of P.T.....
08 Originate Customer contract.
07 Originate Reserve P.T.....
06 Originate Product Type......
05 Originate Loan of Product...
04 Originate Product Type......
03 Originate Registered Address
02 Originate Place.............
01 Originate Party.............
```

PMG id.	Description	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
01	Accept PARTY	o		o					o				a			b	v
02	Register/Recognize PLACE		o														
03	Change REGISTERED ADDRESS			o													
04	Reserve PRODUCT TYPE																
05	Establish LOAN OF PRODUCT					o											
06	Add PRODUCT																
07	Add PRODUCT TYPE																
08																	
09																	
10																	
11																	
12																	

Figure 54. Process methods group matrix (PMG matrix)

------------------ EINDE EINDE ---------------------------

Te business processes for Tiger Video are:

1. Accept PARTY
2. Recognize PLACE
3. Change REGISTERED ADDRESS
4. Reserve PRODUCT TYPE
5. Establish LOAN OF PRODUCT
6. Add PRODUCT
7. Add PRODUCT TYPE

The process Accept PARTY In this process the entities PARTY, REGISTERED ADDRESS and CUSTOMER CONTRACT are originated. Which attributes are originated?

Figure 55. Process Attribute Table for Accept PARTY

Accept PARTY	PARTY (0)	Name	0
		Person/Business	0
		Initials	0
		Passport nr.	0
		Driver's licence nr.	0
		Phone number	0
		Credit limit	0
		Credit status	0
		Current Amount of T.	x
	REGISTERED ADDRESS (0)	Start date	0
		End date	x
		Number	0
	PLACE (▶)	Name	▶
		Suffix	▶
		Type	▶
	PLACE-PLACE	Start Date	▶
		End date	x

Figure 56. Process Attribute Table for Change REGISTERED ADDRESS

Change REGISTERED ADDRESS	REGISTERED ADDRESS (0)	Start date (old)	x
		End date (old)	0
		Number (old)	x
		Start date (new)	0
		End date (new)	x
		Number (new)	0
	PLACE (▶)	Name	▶
		Suffix	▶
		Type	▶
	PLACE-PLACE	Start Date (old)	x
		End date (old)	0

Because a trigger is needed to start a business process, there must be a relation between computer processes which are part of that business process and the trigger. I make the following assumption:
A trigger is needed to start a computer process.
For example: A person asks for enrollment -- > Accept PARTY
The next step is to identify the methods which are part of that process.

Accept PARTY

1. PARTY new
2. Add PARTY Name
3. Add PARTY Initials
4. Add PARTY Passport number
5. Add PARTY Driver's licence
6. Add PARTY Phone number
7. REGISTERED ADDRESS new
8. Add REGISTERED ADDRESS Start date

9. Add REGISTERED ADDRESS Number

IF PLACE unknown THEN

1. PLACE new
2. Add PLACE name
3. Add PLACE suffix
4. Add PLACE Type

IF PLACE-PLACE unknown THEN

1. PLACE-PLACE new
2. Add PLACE-PLACE Start date

Change REGISTERED ADDRESS

1. Add REGISTERED ADDRESS Start date
2. REGISTERED ADDRESS new
3. Add REGISTERED ADDRESS Start date
4. Add REGISTERED ADDRESS Number

IF PLACE unknown THEN

1. PLACE new
2. Add PLACE name
3. Add PLACE suffix
4. Add PLACE Type

IF PLACE-PLACE unknown THEN

1. PLACE-PLACE new
2. Add PLACE-PLACE Start date

Add PRODUCT TYPE

1. PRODUCT TYPE new
2. Add PRODUCT TYPE Name
3. Add PRODUCT TYPE Description
4. Add PRODUCT TYPE Current Price
5. Add PRODUCT TYPE Current Delivery time

= = = = = = = = = = = = = = EINDE =

Step 5: Establish relationships between objects

Identifying the methods is not the same as design the method implementation.

For example, we have to define the creation of a new customer. Then we have to add the values of the attributes, and to save these values.

Which processes need 'create object' messages?

1. Accept PARTY
   We might want to create new parties and addresses. (Although Smalltalk sees everything as an object, I want to have a less extreme approach).

Add PRODUCT TYPE	PRODUCT TYPE (O)	Name Description Current price Curr. delivery time	O O O O

2. andere business processes !!!

Figure 57. Process Attribute Table for Change REGISTERED ADDRESS

Change REGISTERED ADDRESS	REGISTERED ADDRESS (O)	Start date (old) End date (old) Number (old) Start date (new) End date (new) Number (new)	x O x O x O
	PLACE (▶)	Name Suffix Type	▶ ▶ ▶
	PLACE–PLACE	Start Date (old) End date (old)	x O

Within a business process, it is clear when there is a method needed. For example, accepting a new customer will certainly need the methods new or new:. And it will also need the methods to give values to the attributes Name, Age, Birth Date etc.

The B.O.M. structure can be implemented as a dictionary. For example, a district as key, with some streets as values. The MEMBER OF PRODUCTTYPE can be implemented as superclass - subclass.

Smalltalk

A message specifies which operation is desired, but not how that operation should be carried out. The receiver, the object to which the message was sent, determines how to carry out the requested operation. Messages insure the modularity of the system because they specify the type of operation desired, but not how that operation should be accomplished.

The set of messages to which an object can respond is called its interface with the rest of the system. The only way to interact with with an object is through its interface. A crucial property is that its private memory can be manipulated only by its own operations. A crucial property of messages is that they are the only way to invoke an object's operations.

All instances of a class have the same message interface since they represent the same kind of component.

Each method in a class tells how to perform the operation requested by a particular type of message. When that type of message is sent to any instance of the class, the method is executed. Programming in the Smalltalk system consists of creating classes, creating instances of classes, and specifying sequences of message exchanges among these objects. The memory available to an object is made up of variables. Most of these variables have names. Each variable remembers a single object and the variable's name can be used as an expression referring to that object. Private variable names are accessible only to a single object, like instance variables (amount, name, index, etc.). Shared variables can be accessed by more than one object (Amount, Name, Index, etc.). The object referred to by a variable is changed when an assignment expression is evaluated. A message expression describes a receiver, selector, and possibly some arguments. A message selector is a name for the type of interaction the sender desires with the receiver.

'20' factorial gives an error, because it does not make any sense, it does not belong to its interface / data incorrect!

Different kinds of objects can respond to the same message in different ways.

Implementation in Smalltalk/VPM

EXAMPLE 'Objects' An individual tape will be an object. Objects are created by the message new. p123 := MovieTape new. p456 := MovieTape new. p789 := MovieTape new.

Most of the instance variables correspond with BSDM attributes. For example, the class hierarchy: Subclass of OBJECT: PRODUCT (leverancier) Subclass of PRODUCT: VIDEOTAPE (playing time, video system) Subclass of VIDEOTAPE: MOVIETAPE (title, quality)

EXAMPLE 'Attribute'. How does Smalltalk assign and show the value of the attributes? Assigning attributes: leverancier := 'JUMBO Video' Showing attributes : 'caret' leverancier

EXAMPLE 'Window'

A Window can have one or more panes. Windows are specified in a subclass of ApplicationWindow. The panes are specified within the method 'open', using the method framing:ratio:.

EXAMPLE 'Collection' Collections are the basic data structures, used to store objects in groups. Within Smalltalk, class Collection is an abstract class, defining the common protocol for all its subclasses, like string, array and dictionary.

Smalltalk does not support the idea of roles. In reality, a customer is an occurrence of party, and an occurrence of Customer. But in Smalltalk, a customer is not an occurrence of PARTY.

An object can have methods which will not do anything with its data!

Chapter 7: Combining BSDM and the OO-paradigm.

When we want to combine BSDM with the object oriented paradigm, we should have the advantages of BSDM, and the advantages of the object oriented approach.

How can we combine (the advantages) of BSDM and the OO approach?

There are some possibilities for combining them:

1. Not OO, fully BSDM
   This is the current situation. Applications will be implemented, using third and fourth generation languages.
2. OOP instead of conventional programming.

Here, BSDM will be combined with OOP. The problem which will arise is that BSDM outline and detailed design do not fit well with OOP.

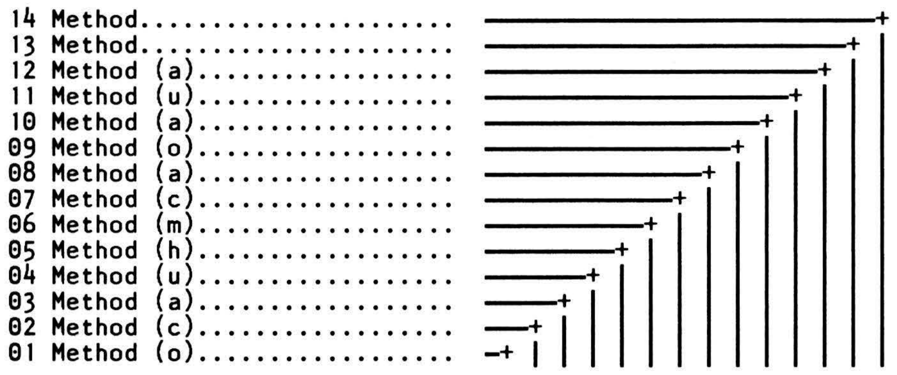
3. Implement BASIS AC's, using OOP.
   This is a very interesting approach. But the object oriented paradigm will only be used in the programming phase.
4. IBS/DM Business Model, followed by OOA, OOD and OOP.
   This approach starts with BSDM Business Modelling. The OOD will be based on this Business Model, using already gathered information.
5. Object oriented Business Model, followed by OOD and OOP.
   Starting with BSDM Business Modelling, the transition to an object oriented business model will be made. An OO BM is a business model, which contains objects, classes and methods. Based on this OOBM, an OO design will be made, which will be used for OOP. Probably, the basic idea of modelling the WHAT of the business, cannot be realized in an OO model, which is oriented towards the HOW. Dynamical aspects are contradictorily to the nature of the business.
6. Not BSDM, fully OO.
   This approach goes fully OO, forgetting BSDM.

In this chapter, the fourth approach for combining them will be described: BSDM Business modelling, followed by OOD and OOP.

This approach builds upon the BSDM Business Model, that contains a lot of useful information (as we will see). Within IBM, Business Models are currently being made. If we can use already gathered information, then we should do that. Perhaps this approach is finally not the best solution. But I think that this approach is the best way to come to that best solution.

If IBM really wants to develop OO applications, then it should be able to use already gathered information. Because there are currently business modelling activities, within IBM, it is interesting to know to what extent the business models can be used. If there might be good reasons for developing OO applications, there should be a convenient transition to this approach. Not only for the people who work with it. But also for the organization IBM. A new development approach means an organizational change, which might be very large. The business model contains information about the data and the processes. Both are needed in an object oriented approach.

The overview of PMG's are given in a process methods group matrix. See Figure 58 on page 70.



```
14 Method....................
13 Method....................
12 Method (a)................
11 Method (u)................
10 Method (a)................
09 Method (o)................
08 Method (a)................
07 Method (c)................
06 Method (m)................
05 Method (h)................
04 Method (u)................
03 Method (a)................
02 Method (c)................
01 Method (o)................
```

PMG id.	Description	01	02	03	04	05	06	07	08	09	10	11	12	13	14
01	Business Process	o		a							a		a		
02	Business Process		c				m		a						
03	Business Process	o				h									
04	Business Process			a				c							
05	Business Process									o					
06	Business Process				u							u			
07	Business Process														
08	Business Process														
09	Business Process														
10	Business Process														
11	Business Process														
12	Business Process														

Figure 58. Process methods group matrix (PMG matrix): Such a method is indicated by a 'c' in the matrix. For example, Business Process 2 needs assign method 8. Such a method is indicated by an 'h' in the matrix. For example, Business Process 3 needs change method 5. Such a method is indicated by an 'a' in the matrix. Such a method is indicated by an 'u' in the matrix. For example, Business Process 4 needs calculate method 6. Such a method is indicated by an 'm' in the matrix. For example, Business Process 2 needs monitor method 6.

Although the multi covered entities might suggest problems, the case study demonstrates that the resulting inheritance structures do not conflict with each other. During BSDM business modelling, the business processes are divided into originate-, change-, and reference- processes.

- Originate processes imply create methods.
- Change processes imply methods that change values of attributes.
- Reference processes imply methods that references to values of attributes.

This matrix may become very confused, when many business processes and methods are related to each other. One of the ways to make such a large diagram surveyable is to have automated support with highlighted columns and rows.

When an object is created, it contains the instance variables without values.

Methods are needed to assign values to instance variables (attributes).

Assigned values are retrieved by methods

Different methods can be grouped into one method.

When there are restrictions to the values, this can be implemented in the methods.

For example: colorblue, colorred, colorgreen. Three methods that can assign three values to the attribute color.

ONDERZOEKSRAPPORT BSDM - OOD/OOP

August 21st, 1990

P.J.T. Lambregts (Student T.U.E.)

IBM, EBSAT
Uithoorn, The Netherlands

Table of Contents

Inleiding

Maandag 11 september 1989 is gestart met het afstudeerprojekt 'Object oriented development with BSDM' in het kader van de bedrijfskunde studie aan de Technische Universiteit te Eindhoven. Dit rapport is het onderzoeksrapport dat bij dit afstudeerprojekt hoort. De afstudeeropdracht is uitgevoerd bij de afdeling EBSAT (EMEA Business Systems Architecture and Technology) van IBM in Uithoorn. Deze afdeling onderzoekt methoden en hulpmiddelen voor systeemontwikkeling binnen IBM EMEA (Europe, Middle East and Africa), op basis waarvan advies wordt uitgebracht aan het Europese hoofdkantoor in Parijs. Het afstudeerprojekt is afgerond in augustus 1990.

Aanleiding tot onderzoek

De aanleiding van het onderzoek ligt ten grondslag aan twee belangrijke ontwikkelingen voor IBM. De ene ontwikkeling betreft de invoering van de systeemontwikkelingsmethode BSDM (Business Systems Development Method). Sinds december 1987 is BSDM verplicht voor de automatiseringsafdelingen binnen IBM EMEA (Europe, Middle East and Africa). Met de invoering van deze methode gaan grote investeringen gepaard.
De andere ontwikkeling is de toegenomen aandacht in de literatuur en in de praktijk voor object georiënteerde systeemontwikkeling. Velen denken dat deze aanpak het systeem ontwikkelingsproces kan verbeteren.

Voor EBSAT zijn er tal van vragen op dit gebied:

- Biedt de object georiënteerde aanpak gewenste oplossingen?
- Is verdere studie van de object georiënteerde aanpak nodig?
- Als inderdaad blijkt dat de object georiënteerde aanpak te prefereren valt boven de huidige wijze van werken, op welke wijze kan dan het beste geprofiteerd worden van deze aanpak?
- Sluiten beide benaderingen elkaar uit, of kunnen ze worden gecombineerd?

Vele van dit soort vragen kunnen nog niet goed beantwoord worden.

Conclusie

Centraal binnen het afstudeerprojekt staat de integratie van de IBM systeem ontwikkelingsmethode BSDM met het object georiënteerde paradigma. Hiertoe is gedurende de uitvoering van de opdracht een initieel systeem ontwikkelingsproces gedefinieerd dat beide benaderingen integreert.
De belangrijkste conclusie is dat de integratie van BSDM met het object georiënteerde paradigma mogelijk is en veelbelovend.

Rapportage

Gedurende het afstudeerprojekt zijn verschenen:

- Literatuur onderzoek {Lit. 1. on page 11}.

- Afstudeerrapport {Lit. 2. on page 11}.
- Appendices {Lit. 3. on page 11}.
- Onderzoeksrapport {Lit. 4. on page 11}.

Behalve het onderzoeksrapport zijn de rapporten geschreven in het Engels. Rapportage in het Engels betekent een veel grotere toegankelijkheid binnen IBM wereldwijd. Daarbij komt dat de directe manager een Engelsman is en voor de afstuderende het leereffect groter is.

Het afstudeerprojekt

Voorbereiding

Het eerste contact voor deze afstudeeropdracht vond plaats in mei 1989 tussen de bedrijfsbegeleider Ir.J.de Lint en de afstuderende. Hieruit volgde een globale opdrachtbeschrijving. Op basis van dit contact en de globale opdrachtbeschrijving werd goedkeuring gevraagd door de afstuderende aan de vakgroep BISA. Na goedkeuring van Prof.Dr.T.M.A.Bemelmans kon het afstudeerprojekt van start gaan.

Opdrachtformulering en begeleiding

Maandag 11 september 1989 werd begonnen met de afstudeeropdracht. De eerste weken werden in beslag genomen door bestudering van het probleemgebied, dat na twee en een halve maand resulteerde in de opdrachtformulering die goedgekeurd werd door de eerste begeleider Ir.J.Trienekens, de bedrijfsbegeleider Ir.J.de Lint en de afstuderende.
Gedurende het afstudeerprojekt is Dr.Ir.P.Bots (Technische Universiteit Delft) de tweede begeleider geworden. Het contact met de de tweede begeleider is tot stand gekomen op basis van bestaande contacten tussen de EBSAT manager H.Jurriaans en Prof.Dr.H.Sol van de Technische Universiteit Delft.
De derde begeleider is Prof.Dr.T.M.A.Bemelmans.

Opdrachtbeschrijving

1. Modelleer een case studie met BSDM en evalueer BSDM.
2. Beschrijf het object georiënteerde paradigma en de voordelen en nadelen van object georiënteerde systeem ontwikkeling.
3. Beschrijf in welke mate BSDM specificaties kunnen worden gebruikt voor object georiënteerde systeem ontwikkeling
4. Bouw een demonstratie model van het ontwikkelingsproces met Smalltalk/VPM, gebruik makend van de case studie.
5. Evalueer the opdracht, the case, het demonstratie model en het ontwikkelingsproces.

Volgens de opdrachtbeschrijving wordt eerst BSDM en het object georiënteerde paradigma afzonderlijk bestudeerd voordat er een (mogelijke) integratie plaatsvindt.

Plan van aanpak en resultaat

Voor elke taak in de plan van aanpak wordt het doel omschreven en het resultaat.

1. Bestuderen van BSDM.
   Het doel van deze taak is BSDM kennis vergaren en inzicht verkrijgen in deze methode.
   Hiervoor is met name gebruik gemaakt van de BSDM manuals en de BSDM experts binnen EBSAT.

Het resultaat is een beschrijving van BSDM met een evaluatie van BSDM.

2. Uitwerken van een case studie.
   Het doel van deze taak is om de kennis van BSDM te vergroten, het inzicht in deze methode te verruimen, en het beschikbaar maken van een uitgewerkte case studie dat kan worden gebruikt voor demonstraties in het kader van dit onderzoek.
   Hiervoor is met name gebruik gemaakt van de BSDM manuals en de BSDM experts binnen EBSAT.
   Het resultaat is een case studie betreffende een videotheque waarvan een (gedeeltelijk) business model is gemaakt.
3. Bestuderen van het object georiënteerde paradigma.
   Het doel van deze taak is om kennis vergaren van het object georiënteerde paradigma en inzicht te verkrijgen in dit paradigma.
   Hiervoor is gebruik gemaakt van literatuur op dit gebied, zowel boeken als artikelen. Daarbij zijn er vele discussies gevoerd met deskundigen op dit gebied, zowel binnen als buiten EBSAT. Buiten EBSAT met name op het congres 'De object georiënteerde aanpak: Modegril of must?' van 6 april 1990 te Utrecht.
   Het resultaat is het rapport 'Object oriented systems development' {Lit. 1. on page 11} dat tevens de bijzondere opdracht is.
4. Leren programmeren in Smalltalk/VPM.
   Het doel van deze taak is om ervaring op te doen met een belangrijke object georiënteerde taal, waarmee een beter begrip ontstaat van het object georiënteerde paradigma.
   Hiervoor wordt gebruik gemaakt van Smalltalk/VPM, release 1.0 op een IBM Model/80 met 8 Mb geheugen.
   Het resultaat is enige ervaring met een belangrijke object georiënteerde taal op basis waarvan een beschrijving is gemaakt van Smalltalk/VPM en enkele Smalltalk/VPM systeemklassen.
5. Integreren van BSDM en het object georiënteerde paradigma.
   Het doel van deze taak is te onderzoeken in hoeverre BSDM en het object georiënteerde paradigma geïntegreerd kunnen worden.
   Hiervoor is gebruik gemaakt van experts en literatuur op beide gebieden.
   Het resultaat is een initieel ontwikkelingsproces, genaamd BSDM - OOD/OOP dat beide benaderingen integreert, de beschrijving van de sterke en zwakke punten van dit ontwikkelingsproces en een overzicht van de aandachtsgebieden.
6. Ontwikkelingsprocess voor de case studie.
   Het doel van deze stap is om de integratie van BSDM en het object georiënteerde paradigma te demonstreren aan de hand van de case studie en te komen tot verbeteringen van BSDM - OOD/OOP.
   Het resultaat is een uitwerking van de case studie voor de eerste vier fasen van BSDM - OOD/OOP.
7. Smalltalk implementatie
   Het doel is om te demonstreren dat BSDM - OOD/OOP werkt. Maar voordat dit gedaan kan worden zal eerst moeten worden nagedacht over het ontwerpen en implementeren van voorgedefinieerde klassen. Pas dan wordt een dergelijke applicatie waardevol. Tijdens het afstudeerprojekt is er geen applicatie geïmplementeerd dat gebaseerd is op BSDM - OOD/OOP.
8. Conclusies en aanbevelingen
   Het doel van deze taak is om op basis van de bevindingen conclusies en aanbevelingen te doen die als input dienen voor (besluitvorming omtrent) verder onderzoek. Het resultaat is een overzicht van de sterke en zwakke punten van BSDM - OOD/OOP, de aandachtsgebieden, conclusies en aanbevelingen voor verder onderzoek.

Evaluatie afstudeerprojekt

In dit hoofdstuk worden geëvalueerd: De opdracht, de case studie, het gebruik van Smalltalk/VPM binnen de opdracht, BSDM - OOD/OOP en de conclusie.

Opdracht

De opdracht is door mij zeer positief ervaren. Het onderzoeksgebied is zeer interessant en ik heb daar veel van geleerd. Het is geen eenvoudige opdracht, maar wel een opdracht waar je een behoorlijke dosis creativiteit in kwijt kan. De opdracht is m.i. wel erg ruim, zodat je er niet aan ontkomt om bepaalde zaken minder diep te onderzoeken en te belichten dan je eigenlijk wilt. Bovendien is voor deze opdracht een langere stage periode nodig.
Alles overziend, ben ik zeer tevreden over de opdracht die ik heb kunnen uitvoeren.

Case studie

De case studie is erg belangrijk geweest. Het was een middel voor mij om BSDM beter te begrijpen. Toepassen van een systeem ontwikkelingsmethode verhoogt het inzicht in de methode in vergelijking met alleen het bestuderen van de methode. Verder is de case studie gebruikt om BSDM - OOD/OOP te demonstreren, dat verhelderend heeft gewerkt bij een aantal mensen.

Gebruik van Smalltalk/VPM

Het werken met Smalltalk/VPM is van grote waarde voor de kennis van en het inzicht in het object georiënteerde paradigma. Zonder het experimenteren met Smalltalk (of een andere object georiënteerde programmeertaal) zou het inzicht zeker minder geweest zijn. Het experimenteren met Smalltalk heeft zeker de waarde van het werk verhoogd.

BSDM - OOD/OOP

BSDM - OOD/OOP is een eerste poging om BSDM te integreren met het object georiënteerde paradigma. Dat betekent dat BSDM - OOD/OOP verder zal moeten worden uitgewerkt om het geschikt te maken voor applicatie ontwikkeling op grote schaal. Theoretisch gezien bevat BSDM - OOD/OOP erg veel concepten van BSDM en het object georiënteerde paradigma. Praktisch gezien kan het verder worden uitgewerkt en gebruikt voor experimentele toepassingen.
Ik ben erg tevreden over BSDM - OOD/OOP. BSDM en het object georiënteerde paradigma kunnen beter worden geïntegreerd dan aanvankelijk werd gedacht. Bovendien heeft nog niemand de integratie van beide benaderingen in vergelijkbaar detail beschreven als nu is gedaan.

Demonstratie applicatie

Er is geen demonstratie model gemaakt, gebaseerd op BSDM - OOD/OOP. Het maken van een demonstratiemodel zou gebaseerd moeten zijn op een verder uitgewerkte versie van BSDM - OOD/OOP, en zou gebruik moeten maken van een verzameling voorgedefinieerde classes. Met name de user interface classes en eventueel de business en applicatie classes. Pas dan is zo'n demonstratie applicatie zinvol. Het implementeren van een aantal classes met instance variables en methods is geen probleem. Maar het gaat om de weg waarlangs de classes, instance variables en methods worden gevonden. Wanneer BSDM - OOD/OOP verder is uitgewerkt zal een demonstratie applicatie zeker waardevol zijn.

Conclusie

De conclusie valt positief uit voor EBSAT. De tegenovergestelde conclusie, namelijk dat BSDM en het object georiënteerde paradigma slecht kunnen worden geïntegreerd, werd (in het begin van het projekt) niet uitgesloten. De conclusie is erg belangrijk. Het betekent dat IBM een goede uitgangspositie heeft voor het ontwikkelen van object georiënteerde applicaties. Wat betreft de methodische ondersteuning biedt BSDM - OOD/OOP momenteel meer dan volledig object georiënteerd ontwikkelen (dat biedt zeer weinig).

Wetenschappelijkheid

Op basis van 'Herkenningspunten van wetenschappelijk werk' {Lit. 5. on page 11} wordt het afstudeerwerk geëvalueerd naar de mate van wetenschappelijkheid.

1. De eis van openbaarheid.
   Op grond van de beschreven gegevens moet het mogelijk zijn de belangrijkste stappen in het onderzoek op hun juistheid na te trekken en te beoordelen.

   Uitgaande van de taken in de plan van aanpak, wordt voldaan aan deze eis.

   Taak 1: Bestuderen van BSDM.
   De resultaten van deze stap zijn vastgelegd in hoofdstuk 4 en in de appendices A, B en C.
   Taak 2: BSDM case studie.
   De resultaten van deze stap zijn vastgelegd in appendices D, E en F.
   Taak 3: Bestuderen van het object georiënteerde paradigma.
   De resultaten van deze stap zijn vastgelegd in het verslag betreffende het literatuuronderzoek en hoofdstuk 5 van het afstudeerrapport.
   Taak 4: Bestuderen van Smalltalk/VPM.
   De resultaten van deze stap zijn vastgelegd in appendix G en H.
   Taak 5: Integreren van BSDM en het object georiënteerde paradigma.
   De resultaten van deze stap zijn vastgelegd in hoofdstuk 6.
   Taak 6: Het ontwikkelingsproces demonstreren voor de case studie.
   De resultaten van deze stap zijn vastgelegd in appendix I.
   Taak 7: Implementeren van Smalltalk applicatie.
   Deze stap is niet als zodanig uitgevoerd.
   Taak 8: Conclusies en aanbevelingen.
   De resultaten hiervan zijn vastgelegd in hoofdstuk 7 en 8.

2. De eis van voorlopigheid.
   De tweede eis betreft het voorlopige karakter van de verkregen kennis. Een onderzoek bouwt altijd voort op bestaand werk en levert slechts voorlopige kennis en (vaak veel) nieuwe vragen op. Het bestaande werk in het kader van dit onderzoek betreft met name BSDM en het object georiënteerde paradigma. De opgedane kennis zal verder uitgewerkt moeten worden en nieuwe aandachtsgebieden en vragen zijn vermeld in hoofdstuk 7.

3. De eis van objectiviteit.
De onderzoeker is verplicht hetgeen in de vakliteratuur wordt aangetroffen feitelijk weer te geven. Het weergeven van feiten, gegevens, en van een heersende mening in de vakliteratuur geschiedt los van de uiteenzetting of weergave van eigen interpretaties of conclusies. In hoofdstuk 3, betreffende BSDM is een onderscheid gemaakt tussen een beschrijving van BSDM en de evaluatie van BSDM. In hoofdstuk 5 wordt ingegaan op de theorie van het object georiënteerde paradigma. De conclusies zijn apart weergegeven. BSDM - OOD/OOP is beschreven in hoofdstuk 6 en geëvalueerd in hoofdstuk 7.

4. De eis van precisie.
In de scriptie dient op verstandelijke wijze te worden gekozen tussen de vormen van inhoudelijke presentatie van de methode en de resultaten van onderzoek. Zonder criteria is moeilijk te beoordelen in welke mate aan deze eis is voldaan. 'Verstandelijk' is niet gedefinieerd.

5. De eis van betrouwbaarheid.
In hoeverre worden dezelfde conclusies en resultaten bereikt indien het onderzoek door een ander met dezelfde vraagstelling zou worden herhaald. Dat de persoonlijke visie van de auteur een -soms zeer- belangrijke rol in het onderzoek speelt doet niet af aan het terechte verlangen dat de gevonden resultaten en conclusies uit het onderzoek een betrouwbare weergave dienen te zijn van het voor het onderwerp beschikbare materiaal. Of aan deze eis is voldaan valt moeilijk te zeggen.

6. De eis van geldigheid.
Met de eis van geldigheid wordt bedoeld dat een wetenschappelijke redenering in de scriptie volgens regels van de logica in elkaar zit en dat de argumentatie correct is. Het geeft weer de mate waarin de resultaten en conclusies uit het onderzoek een juiste afspiegeling zijn van hetgeen in de literatuur over het onderwerp is geschreven.
De resultaten van het afstudeeronderzoek zijn niet tegenstrijdig aan de heersende opvattingen in de literatuur op dit gebied.

7. De eis van effectiviteit.
De voorwaarde dat de resultaten van onderzoek toepasbaar moeten zijn en dus ook zodanig geformuleerd dat er iets mee gedaan kan worden. Aan deze eis wordt zeer goed voldaan. BSDM - OOD/OOP kan verder worden uitgewerkt en toegepast.

Afgezien van eis 4 en 5, voldoet dit onderzoek aan de eisen die hierboven zijn gesteld.

Samenvatting van evaluatie

Hoewel er nog veel aktiviteiten ondernomen moeten worden en ik bepaalde aspecten verder uit had willen werken is er een duidelijk resultaat van het projekt. Momenteel wordt bestudeerd wat EBSAT met dit werk verder gaat doen.

Ik ben tevreden over mijn afstudeerprojekt, het is een geslaagd projekt. Het onderzoeksgebied is zeer interessant, het resultaat is een (kleine) stap vooruit op het gebied van object georiënteerde systeemontwikkeling, het resultaat is nuttig voor EBSAT, de ondersteuning vanuit EBSAT was groot, en ik heb een zeer leerzame en ook plezierige afstudeerperiode achter de rug.

Gebruikte hardware en software

Gedurende het afstudeerprojekt is gebruik gemaakt van de volgende hardware en software.

Hardware

- PS/2 Model 80
  IBM personal system 2 met 8Mb geheugen en 70Mb vaste schijf.
- IBM 3090
  IBM mainframe.

Software

- OS/2, versie 1.2
  Operating system voor de PS/2, versie 1.2.
- DOS versie 4.0
  Operating system voor PC's.
- Smalltalk/VPM
  Object georiënteerde programmeeromgeving dat werkt onder OS/2 Presentation Manager.
- Freelance
  Software voor het maken van sheets.
- XEDITG
  Editor op de host.
- VM
  Operating systeem voor 3090.
- Foils 5.0
  Software op de 3090 voor het maken van sheets.
- PCASSI
  Software voor het file transport tussen de host en de PS/2.

Literatuur

1. ***Lambregts, P.J.T.***
   Object oriented systems development
   Verslag literatuuronderzoek (Bijzondere opdracht), Augustus 1990.

2. ***Lambregts, P.J.T.***
   Object oriented development with BSDM
   Afstudeerrapport, Augustus 1990.

3. ***Lambregts, P.J.T.***
   Appendices BSDM - OOD/OOP
   Bijlage van Afstudeerrapport, Augustus 1990.

4. ***Lambregts, P.J.T.***
   Onderzoeksrapport BSDM - OOD/OOP.
   Augustus 1990.

5. ***Wessels, Mr B.***
   Juridische vaardigheden
   Kluwer, tweede druk, 1984.