
Michal Smialek

10 September 1996

This dissertation is a part requirement for the MSc in Advanced Software Engineering.
Abstract

The work presents Discovery, a new third-generation method for object-oriented software development. The method is presented together with a case study which tests the method’s techniques, notations and process. Four development phases of Discovery - Requirements, Analysis, Design and Implementation are presented in detail. A development process is proposed which incorporates these four phases. For each of the steps in the process, a set of techniques is presented. Already existing techniques are used to form a new logical development path.

A set of activities which use relevant techniques and notations, is proposed for each development phase. A format of deliverable for each activity is suggested. Several new notations are introduced and existing notations used to denote important elements of the deliverables. A method for reuse of already delivered systems with their architectures is proposed. A company-wide solution for architecture and component reuse is suggested.

The process and techniques of Discovery are used to analyse an In-Flight Cabin Services system. Important aspects of the system requirements are described. Examples of deliverables are shown.
Acknowledgements

I would like to thank people at GEC Marconi and of the Excalibur Scholarship that they made possible for me to spend this year on fruitful studies. I would like to thank EASAMS Ltd. for supplying me with documentation necessary to perform the important case study part of this thesis. I would like to dedicate this work to my wife Grazyna and to the new member of our family.
# Table of Contents

1. **Introduction** ..................................................................................................................... 1  
   1.1. Motives ............................................................................................................................ 1  
   1.2. Objectives ....................................................................................................................... 2  
   1.3. IFCS Case Study ............................................................................................................. 2  

2. **Background** ...................................................................................................................... 3  

3. **Method Overview** .............................................................................................................. 5  

4. **Requirements Phase** ....................................................................................................... 7  
   I. **Notation and Activities** .................................................................................................. 7  
      4.1. Domain and Design Experts ......................................................................................... 7  
      4.2. Communication between Experts ............................................................................... 8  
      4.3. Vocabulary .................................................................................................................. 8  
          4.3.1. Actors Vocabulary ................................................................................................. 8  
          4.3.2. Noun Phrase Vocabulary .................................................................................... 9  
          4.3.3. Verb Phrase Vocabulary ..................................................................................... 9  
      4.4. Task and Narrative Models ......................................................................................... 9  
          4.4.1. List of Scenarios .................................................................................................. 10  
          4.4.2. Specification of Scenarios .................................................................................... 10  
      4.5. Requirements Cross-Checks ...................................................................................... 11  
      4.6. Summary of Activities - Requirements Document ................................................... 11  
          4.6.1. Preparation of the Requirements Statement (Activity R1) - option .................... 11  
          4.6.2. Construction of the Vocabularies (Activity R2) ................................................. 11  
          4.6.3. Definition of the Task Model (Activity R3) ......................................................... 12  
          4.6.4. Definition of the Narrative Model (Activity R4) ............................................... 12  
          4.6.4. Requirements Cross-check (Activity R5) ............................................................ 12  
      4.7. Reuse of Requirements ............................................................................................... 12  
          4.7.1. Software Architecture Library ............................................................................. 13  
          4.7.2. Reuse Process ...................................................................................................... 13  
   II. **Case Study** .................................................................................................................... 14  
      4.8. Requirements Statement - Deliverable R1-1 ............................................................... 14  
          4.8.1. In-Flight Cabin Services system ...................................................................... 14  
          4.8.2. Shopping Application ......................................................................................... 15  
          4.8.3. Faxgram Application .......................................................................................... 15  
          4.8.4. Prize Draw Application ....................................................................................... 15  
          4.8.5. Quiz Application .................................................................................................. 15  
          4.8.6. News Application .................................................................................................. 15  
          4.8.7. Survey Application .............................................................................................. 16  
          4.8.8. Car/Hotel Reservation Application .................................................................... 16  
      4.9. Actors Vocabulary - Deliverable R2-1 ........................................................................ 16  
      4.10. Noun Phrase Vocabulary - Deliverable R2-2 ............................................................. 16  
          4.10.1. Credit Card Noun Phrases .................................................................................. 17  
          4.10.2. Shopping Noun Phrases ..................................................................................... 17  
          4.10.3. Prize Draw Noun Phrases .................................................................................. 18  
          4.10.4. News Noun Phrases ........................................................................................... 18  
      4.11. Verb Phrase Vocabulary - Deliverable R2-3 ............................................................... 18  
          4.11.1. Credit Card Verb Phrases .................................................................................. 19  
          4.11.2. Shopping Verb Phrases ...................................................................................... 19  
          4.11.3. Prize Draw Verb Phrases ................................................................................... 20
4.11.4. News Verb Phrases ................................................................. 20
4.12. List of Scenarios - Deliverable R4-1 ........................................... 20
  4.12.1. Credit Card Scenarios ....................................................... 21
  4.12.2. Shopping Scenarios ......................................................... 22
  4.12.3. Prize Draw Scenarios ....................................................... 24
  4.12.4. News Scenarios .............................................................. 25

5. Analysis Phase ............................................................................ 26
I. Notation and Activities ................................................................. 26
  5.1. Interaction Model ................................................................. 26
  5.2. Collaboration Model ............................................................ 28
  5.3. Auxiliary Models ................................................................. 30
  5.4. Analysis Cross-Checks ......................................................... 31
  5.5. Summary of Activities - Analysis Document ............................ 31
    5.5.1. Preparation of the Interaction Model (Activity A1) ............... 31
    5.5.2. Preparation of the Collaboration Model (Activity A2) .......... 32
    5.5.3. Preparation of the Auxiliary Models (Activity A3) ............. 32
    5.5.4. Analysis Cross-check (Activity A4) .................................. 32
  5.6. Reuse of Analysis ............................................................... 32
II. Case Study .................................................................................. 33
  5.7. Interaction Model - Deliverable A1-1 ....................................... 33
  5.8. Collaboration Model - Deliverables A2-1, A2-2 ....................... 41

6. Design Phase ................................................................................ 45
I. Notation and Activities ................................................................. 45
  6.1. Object/Class Model .............................................................. 45
  6.2. Reuse of Software Components ............................................. 48
    6.2.1. Components Library ....................................................... 49
    6.2.2. Reuse Steps ................................................................. 49
  6.3. Summary of Activities - Design Document ............................. 50
    6.3.1. Preparation of the Object/Class Model (Activity D1) .......... 50
    6.3.1. Reuse of Software Components (Activity D2) ............... 50
II. Case Study .................................................................................. 50

7. Implementation Phase ................................................................... 52
I. Notation and Activities ................................................................. 52
  7.1. Feature Layer ....................................................................... 52
  7.2. Method Layer ....................................................................... 53
  7.3. Behaviour Model ................................................................. 55
  7.4. Visibility Model ..................................................................... 55
  7.5. Summary of Activities- Implementation Document ............... 56
    7.5.1. Definition of the Feature Layer (Activity I1) .................... 56
    7.5.2. Definition of the Method Layer (Activity I2) ................. 56
    7.5.3. Construction of the Behaviour Model (Activity I3) ...... 56
    7.5.4. Construction of the Visibility Model (Activity I4) ..... 56
II. Case Study .................................................................................. 57

8. Summary and Conclusions ........................................................... 59
References ....................................................................................... 60
Appendix. Project Diary ................................................................. 62
# Table of Figures

- Figure 1. Process of Discovery. ................................................................. 6
- Figure 2. Interaction Diagram - notation. ............................................... 27
- Figure 3. Collaboration Graph - notation. .............................................. 28
- Figure 4. CRC Form - notation. .............................................................. 29
- Figure 5. Classes - notation. ................................................................. 46
- Figure 6. Inheritance relationships. ...................................................... 46
- Figure 7. Association and aggregation. ................................................ 47
- Figure 8. Simplification of communication. .......................................... 48
- Figure 9. Class with a feature part. ....................................................... 53
- Figure 10. Features - notation. ............................................................ 53
- Figure 11. Method sheets, invariants - notation. ................................... 54
- Figure 12. Method Trace Diagram - notations. ..................................... 55
1. Introduction.

1.1. Motives.

The main area of interest of the present work is the object-oriented paradigm of software development. In the recent years a vast number of new software analysis and design methods have emerged that use this paradigm. Many organisations consider changing their software development lifecycle to be conformant with the object-oriented technology. Many projects have already been successfully developed with the help of an object-oriented method and using an object-oriented programming language. To justify introducing and defining a new method of object-oriented software development we need to answer two basic questions. The first question is: “why an object-oriented method?”, and the second question is: “why a new method?”

Answering the first question is certainly out of scope of this work. However, we can mention the most important reason for introducing object-orientation, which is its ability to deal with complexity. Software is undoubtedly the most complex element of computer systems. The problem of dealing with this complexity has been addressed as far as during the famous, first conference on software engineering organised by NATO in 1968 [Buxton, 1976]. The term used throughout this conference is “software crisis”. This crisis or maybe rather “chronical affliction” (see [Pressman, 1994], p. 12) has not been resolved since that time. Many software projects are not delivered on time, have overspent their budget, and the resulting systems are of unsatisfactory quality. We can also give various examples of spectacular failures caused by software errors. Among the recent ones is the collapse of the Ariane 5 rocket [Owen, 1996]. Unfortunately, we are unable to solve all the current problems by simply applying object-orientation to existing systems. In this sense, the object-oriented paradigm is no “silver bullet” [Brooks, 1987]. However, we can now try to avoid the problems by constructing “better software” in the future. Better software means software produced quickly, cheaply, and what is most important, of good quality - which includes good maintainability. The object-oriented paradigm claims to have solutions to provide these features and many arguments can be found in the existing literature (see references in Chapter 2).

Currently there are at least ten significant methods or methodologies in the area, and this is the point where we should answer the second question asked at the beginning: why another method? To answer it we have to note that this vast number of existing methods shows just how immature the whole field is. The researchers and practitioners still try to present new methods (with the method presented in this work not being an exception). Tendencies to unify the methods are yet weak, although existent ([Open, 1996], [UML, 1996]). Many methodologies are influenced one by another. Many use similar or same techniques, but in very different phases of the development cycle. This all leads to a significant confusion in the software industry and reluctance in shifting to the new technology.

In the author’s opinion, the main challenge at present is to show a clear and uniform method of software development for object-oriented technology. The method should be clear enough to convince current software developers about the new paradigm. It should promote those aspects of the new technology that would prevent poor quality software, with the emphasis on reuse, easy maintenance and enhancement. Such a method should use techniques that are as simple as possible, and which are used at proper stages of the software development cycle. The techniques should take into account the psychology of developers. Hence, a stress has to be laid on the use of abstraction, which deals with complexity and keeps the number of elements dealt with at a time within reasonable boundaries. Finally, a clearly defined technical process should be defined in order to eliminate ambiguities and offer guidance especially for less experienced developers.

The methods that have the above features can be classified as third generation o-o development methods. This is in contrast to the existing first and second generation methods. The classification can be found in [Simons, 1996] (see also Chapter 2). Here we can mention that the first generation techniques can be criticised for their naive design structures and process, over-reliance on static data modelling, and late process modelling. The second generation methods tend to have over-rich notations, include techniques indiscriminately, and lack any clear guiding technical process.
1.2. Objectives.

Considering the arguments presented above we can see the need to introduce a new method which would contain a uniform and clear combination of already existing techniques. Such a method was proposed by Dr. Tony Simons (see: [Simons, 1996]) and was called Discovery to emphasise the importance of discovery processes in any software development method. Discovery uses appropriate techniques to focus the developer’s attention only on a limited number of system elements at each stage of development. The key features of the method include a set of precise notations, and a clearly defined technical process. The method is currently under development and in this work we will try to test it in real life and propose certain solutions that are based on this experience.

Initially, the main objective of this work was thus to apply Discovery to a real-life project and produce a case study report. The resulting report was to be a good basis for further development and more precise specification of the method. However, in the current state, Discovery possesses a set of techniques, but lacks in notations and process. Hence, there was introduced another objective, which was to propose these two elements. The propositions should be conformant with outline defined in [Simons, 1996] and [Simons, 1994], and should offer “semi-automatic mechanisms leading to discovery of all the necessary concepts and smooth transition from stage to stage” [Simons, 1994]. Conformance with Discovery’s basic assumptions means that the notations should be as simple as possible not to distract the developer from actually developing a system. The developer should concentrate on using a notation not on learning it constantly (compare: [Booch, 1994]). The process should may not necessarily be “seamless” as it is proposed by some other methods [Walden, 1994], but there should be a precisely defined sequence of activities which lead to the final system.

The rest of this work conforms with the above objectives. According to them, the work is divided into two parts which complement each other. The first part describes the method and proposes notations and activities of the process. It also describes the techniques included so far in Discovery and those newly proposed in the present work. The second part shows the case study and gives detailed examples for the suggested techniques and notations. Here, the reader may also follow the Discovery’s process by comparing gradually built models of the case study project and analysing transitions from stage to stage.

The above two parts can be found in Chapters 4 to 7, which describe consecutive phases of Discovery: Requirements Phase, Analysis Phase, Design Phase, and Implementation Phase. Each of these chapters includes a general description of notation and activities (Part I) and the case study example (Part II) for an appropriate phase. Distributing the case study will allow the reader to verify and assess the elements proposed in parts I immediately after reading relevant description. It is also the author’s belief that such a layout will make it easier to understand the propositions from the first reading, because a direct example is always more powerful for explaining new notions than just a general description. The remaining chapters contain description of other methods (Chapter 2), overview of the process (Chapter 3) and summary of the work with conclusions (Chapter 8).

1.3. IFCS Case Study.

The case study presented in the present work is based on the development of an In-Flight Cabin Services (IFCS) system. The system requirements were taken from a recent requirements specification document prepared by EASAMS Ltd. for one of the major airlines. The main purpose of this system is to provide entertainment, news, shopping and other facilities to passengers on long-haul flights. Each of the passengers has an individual terminal with a special keypad to operate all the features of the system. Many of the included options require payment by credit card, transmitting shopping orders, etc., and thus the system has to communicate with the ground to fulfil its responsibilities. The present system is a variant of a system created by EASAMS for British Airways. The previous system already functions in some of the BA’s Boeing 777 aircrafts. The current system, however, will add to the functionality of the one already developed. More detailed description can be found in section 4.8.

The key advantage of the IFCS project is that it was prepared as a response to real-life requirements. The project is currently at the stage of initial proposals and requirements specification at EASAMS. Thus, there was no pressure to apply company-specific routines and procedures. As for any new method, it would be difficult to persuade applying Discovery in a later stage of system development. The models developed in this work were proposed to EASAMS, and it is possible that they will be used during later development stages at EASAMS. In such case they could become an even better test-field for the new method than the work presented here.
2. Background.

It is still less than a decade since the introduction of first real object-oriented methods and we already can choose from a vast number of them. As it was noted in the introduction, there are at least ten methods worth mentioning. It is not the purpose of this work to assess and present these methods in detail. However, it is necessary to give a brief description, since some of them were the basis and source of techniques for Discovery. For a detailed analysis of the field the reader can be referred to [Brinkkemper, 1995], which was the basis for the current description. A more formal comparison of some methods can be found in [Fichman, 1992] and [Monarchi, 1992].

According to [Simons, 1996], we will divide the described methods into two generations (there is still no third generation method). We will concentrate on listing the main techniques used in the presented methods.

Techniques of the first generation methods.

1. **Shlaer-Mellor method** [Shlaer, 1988], [Shlaer, 1992], [Shlaer/Mellor, 1996].
   - Information model - entity-relationship diagrams showing the information structure of the system.
   - State transition diagrams - description of objects’ lifecycles.
   - Object communication model and Object access model - presentation of the system’s asynchronous communication structure.
   - Data flow diagrams - representation and communication between units of processing.
   - Class diagrams, Dependency diagrams, Inheritance diagrams - presentation of the system’s class structure.

2. **Coad-Yourdon method** [Coad, 1990a], [Coad, 1990b].
   - Data model - a diagram showing the static data structure and communication between objects together with the inheritance relationships between them.
   - Layering - division of the system into transparency-like layers, that show increasingly detailed view of the system.
   - Object state diagrams - a form of finite-state machine description of an object.

3. **Object Modelling Technique** [Rumbaugh, 1991].
   - Object model - a diagram showing the static structure of intra-object relationships, a kind of entity-relationship diagram.
   - Dynamic model - a set of diagrams describing dynamic behaviour of single objects.
   - Functional model - description of system computations using data flow diagrams.

4. **Responsibility Driven Design** [Wirfs-Brock, 1990]
   - Class-Responsibility Collaborator (CRC) cards - definition of class’s responsibilities and classes that it collaborates with to fulfil these responsibilities.
   - Subsystem cards - similar to CRC cards, but applied to subsystems of classes.
   - Class hierarchy graphs - diagrams showing the inheritance relationships.
   - Collaboration graphs - diagrams showing the collaborations between classes.

Techniques of the second generation methods.

1. **Booch’s method** [Booch, 1994], [Rational, 1996].
   - Object diagrams - description of static relationships between existing objects including visibility and synchronisation aspects.
Chapter 2. Background.

• Class diagrams - presentation of relationships between classes with cardinality, class utility, persistence and visibility aspects.
• State transition diagrams - finite-state machine approach to defining the dynamic behaviour of objects.
• Interaction diagrams - dynamic capture of the system scenarios.

2. Fusion method [Coleman, 1994], [Fusion, 1996].
• Object models - static structure of the system’s data based on extended entity-relationship notation.
• Interface model - description of the input and output communication in terms of events and caused changes.
• Object interaction graphs - definition of message sequences for system’s operations.
• Visibility graphs - reference structure of objects in the system.
• Class descriptions - a list of methods, and some attributes for each class.
• Inheritance graphs - graphs showing the generalisation-specialisation structure of the system.

• Object/Class model - static representation of object and class relationships.
• Class specifications - formal description of classes/objects.
• Service specification - definitions of object/class services by using formal contracts with pre- and postconditions, and by showing inside flow of control.
• Event model - visualisation of message handling between objects/classes.
• Object charts - finite-state machine like diagrams showing the internal object/class behaviour.
• Inheritance model - visualisation of the inheritance hierarchy.
• Complexity management - includes layering and division into sheets.

• Use case model - definitions of actors and use cases (scenarios), which describe complete courses of events between the user and the system.
• Analysis model - description of the static object structure of the system.
• Object interaction diagrams - description of use-cases in terms of communicating objects.
• State transition graphs - description of the internal object behaviours using use-case stimuli.

If we analyse this short listing of methods with their techniques we can see that there is no clear and uniform terminology in the object-oriented society. Many similar or even same techniques are named differently by different authors. This is another reason for confusion when applying these methods in real life projects. Moreover, every new method tries to introduce new techniques and ‘improve’ notations, which additionally obscures clarity of the initial models. Especially the Booch’s method leads in this field. This forms a field for a new method to introduce simple notations by referring back to the original method inventors. Such approach can be found in Discovery and is used throughout the following chapters.

The Discovery method consists of two main Stages, which divide into four Phases. The two Stages are Analysis and Design, which is a traditional division in software engineering. The Analysis Stage is divided into Requirements Phase and Analysis Phase, while the Design Stage has its Design and Implementation Phases. Each of the stages has a different approach, and places emphasis on different aspects of the final system. However, all the stages form a logical path which leads from the initial requirements coming from the Problem Domain, up to the final specification of the system which can be expressed by the Object/Class Model together with its four supporting models (see Fig. 1).

In the Requirements Phase, the whole effort is concentrated on capturing the user’s requirements in a way which may serve for future formal verification of the final system. An emphasis is set on establishing common vocabulary for discussion, and on precise expression of the functionality of the user’s domain of activities. The developers do not direct the discussion while interviewing the clients. It is rather the client that describes the procedures of his/her everyday business.

The Analysis Stage changes the emphasis from creating textual narrative descriptions into building models of the problem domain. The developer expresses his/her view of the user’s problem domain in a set of diagrams and graphs. At the same time, primary collaborating objects are discovered. These diagrams and objects are reflected back to the client which verifies them with the actual requirements.

With the Design Phase we start considering the target implementation environment. This is done by structuring the models developed during the analysis. The structuring is made by capturing subsystems with high cohesion of collaborations. The subsystems can be formed into inheritance and containment structures. At the same coarse-grained objects of the problem domain are divided into objects specific for the target system environment. If required, the resulting model may be modified to suit the existing software component libraries.

In the final Implementation Phase we define the contents of objects/classes established in the Design Phase. We do this by developing four supporting models, where the attribute, behavioural, visibility and service/method solutions are specified.

The present work constitutes a proposition for the process (or development path) of Discovery. This development path will be expressed as a series of deliverables. These deliverables are transformed from their predecessors by applying specific techniques. This gives us an unambiguous process of the method, which is shown on Fig. 1. On this figure, deliverables are denoted by rectangles, and arrows denote techniques applied to transform from one deliverable to another. Double head arrows denote the fact that two deliverables can be built concurrently. The appropriate deliverables and techniques are described in the following chapters.
Chapter 3. Method Overview.

Requirements Phase

- **Problem Domain**
  - Non-directive interviewing
  - Verbal expression of user's experience
  - **Requirements Specification**
    - Optional

- **Task Model**
- **Vocabularies** (Actor, Noun, Verb)
- **Narrative Model**
  - Non-directive interviewing
  - Verbal expression of user's experience
  - **Vocabulary SVDPI analysis**

Analysis Phase

- **CRC Forms**
  - CRC Analysis
  - CRC object discovery
  - Graph representation

- **Interaction Model**

Design Phase

- **Object/Class Model**
  - Simplification of communication patterns
  - Generalisation
  - Aggregation
  - **Components Library**
  - Selection of components

Implementation Phase

- **Feature Layer**
- **Method Layer**
- **Behaviour Model**
- **Visibility Model**

*Figure 1. Process of Discovery.*
4. Requirements Phase.

1. Notation and Activities.

Majority of the object-oriented methods use a mixed approach to software development. Some of the performed activities are top-down, some are bottom-up. The same approach will be used in Discovery. The top-down part of the method will be concentrated in the requirements phase. In this phase we thus try to reach the “bottom” of the problem domain, and the following phases go again from this bottom to the top. A note has to be made that the “top” may be reached only for a specific, ready system. Real problem domains usually have no defined top or rather - have many tops. In other words, systems that model certain problem domains should be able to change their top functionality inside these domains easily (see [Meyer 1988]).

Another difference between the requirements phase and other phases is that usually this phase cannot be reiterated after performing some other phases. In many cases the requirements phase will end with signing the requirements documentation and it will not be possible to make any changes. It can be suggested that the document signed is a document prepared according to the outline in section 4.6 (Requirements Document). This document is composed of all the deliverables described in this chapter. Thus, these deliverables grouped in one document and signed may constitute a formal contract between the user and the developer. This will assure both parties that the requirements are precisely defined after a thorough study. This will also allow for fairly mechanical verification of the final system to satisfy the specified requirements.

An important feature of the requirements phase specified in this work is the emphasis on precision. This aspect of the requirements capture is rather under-valued in other methods. An exception here is the method by Jacobsen et al. [Jacobsen, 1992], which uses a “scenario” (or use case) model of requirements. However, even this method does not define any formal notation for precise requirements’ specification.

Another important aspect of the requirements phase described here is the possibility of its reuse. This phase allows for preparation of a reusable requirements specification with discovered and precisely defined elements (abstractions) of the problem domain. Some of these abstractions can be reused when constructing a system in the same or similar problem domain.

The primary abstractions that allow to deal with complexity of the problem domain are Actors (see section 4.3) and Scenarios (see 4.4). The systems are analysed from the point of view of the system users (human or non-human) - i.e. the Actors. Each of the Actors may interact with the system according to defined Scenarios. The main part of the Requirements Document is a definition of Scenarios for the individual Actors. Such a structure of the document allows to concentrate only on one part of the system at a time.

Finding Actors and Scenarios is the main task for the Domain and Design Experts (see section 4.1) during the requirements phase. With the use of abstraction this job can be done more efficiently. At first the concentration is focused on finding the top level entities - the Actors. Then we concentrate on finding Scenarios for each individual Actor. Finally we define individual Scenarios using the Vocabularies of the problem domain (see section 4.3).

4.1. Domain and Design Experts.

The requirements phase is the phase where the interaction between the client/user and the developer is most frequent. At this introductory stage of developing a system we have thus two important parties involved, which usually have contradictory backgrounds.

The first party is the client (or user). This party should have all the necessary knowledge about the problem domain. Hence we will call this party the Domain Expert. The Domain Expert may be just one person, or several people may form a group which will constitute a source of domain expertise for the
software developers. The background of the Domain Expert is the (hopefully thorough) knowledge of the area where the software system shall be implemented.

The second party is the developer, i.e. usually a person or a group which are experts in the design of software systems. We will call this party the Design Expert.

4.2. Communication between Experts.

Often problems arise when the Domain Expert tries to communicate various aspects of the problem domain to the Design Expert. Similar problems awake when the Design Expert tries to inform the Domain Expert about the characteristics and limitations of the software systems. However, much more important is the first group of problems. The Domain Experts are usually trained in basic aspects of computer systems, and are aware of the constraints. On the other hand it is impossible to train the Design Expert in all the possible problem domains taken from real life. Thus it is very important for the Design Expert to acquire necessary knowledge about the current problem domain from the Domain Expert.

The first activity of the Requirements Phase is thus to establish a proper communication between the Experts. The basis for this and the first deliverable should be a textual Requirements Statement (deliverable R1-1, see section 4.6.1) prepared by the Domain Expert. We will not specify any guidelines for writing such a statement. This preliminary activity of the Domain Expert is just the starting point for the whole Requirements Phase. The document that results from this activity is usually the one that the software developer receives from the client even before signing an agreement or before a decision is made on feasibility of the whole project.

After establishing the basis for communication the Experts should perform several other activities which will eventually result in preparing the Requirements Document (see section 4.6). These activities are presented in sections 4.2-4.5. The guiding technique that should be used during these activities is Non-directive Interviewing as proposed in [Gibson, 1990]. The application of this technique depends on psychological skills of the Design Expert. It is very important to avoid pressurising the Domain Expert into any implied thought-modes. We should discover as many elements of the problem domain as possible. This should be done without suggesting any solutions stipulated by the Design Expert’s knowledge about similar software systems.

In the sections describing the Requirements Phase we assume that the communication between the Experts is established on a non-directive basis. This communication allows to prepare deliverables as specified in each of the steps.

4.3. Vocabulary.

The second deliverable (or group of deliverables) of the requirements phase is the Vocabulary. The Vocabulary is composed of three parts: the Actors Vocabulary (deliverable R2-1), the Noun Phrase Vocabulary (R2-2) and the Verb Phrase Vocabulary (R2-3).

4.3.1. Actors Vocabulary.

The Actors Vocabulary is usually the shortest. It contains definitions of all the "Actors" existing in the problem domain. The Actors (see [Jacobsen, 1992]) can be defined as all the roles that can be played by entities that are outside and that interact with the system. Usually those entities are simply the users of the system. However, we have to be more general with the definition of Actors, because many computer systems receive outside interactions from entities that are not human, e.g. other computer systems. Additionally we have to be aware that the users may often play several roles in the system, and thus a single user may have to be described by several Actors. A typical example is of a user that plays a role of a clerk and at the same time has a role of maintaining the system as a system administrator. Each of such roles has to defined by different Actors.

Additionally we have to note that the Actors are the only entities in the problem domain whose behaviour is non-deterministic. All the other entities elicited in the requirements phase should have a
precisely specified behaviour, and this may be a main criterion for distinguishing the Actors from other entities.

Preparation of the Actors Vocabulary consists in finding the Actors in the problem domain during communication between the Experts. Each of the Actors has to be listed, and a definition given by the Domain Expert. All the future activities of the requirements phase will be based on proper definition of the Actors.

### 4.3.2. Noun Phrase Vocabulary.

The Noun Phrase Vocabulary defines all the entities of the problem domain not specified previously as Actors. The simplest technique that can be used to extract Noun Phrases is a linguistic analysis of the Requirements Statement. We should find all the nouns together with accompanying phrases (adjectives, etc.) and decide whether they constitute a separate entity in the problem domain.

Linguistic analysis is a very unreliable technique, as the Requirements Statement might not describe all of the elements of the problem domain. Thus, it is important to elicit nouns with the use of non-directive interviewing. Here we should also elicit phrases that surround and include the discussed nouns. The phrases may include adjectives and other words that describe the nouns. For instance, if we find a noun "car" in the problem domain, we may also find words "new", "used" that describe a car. Thus we result in Noun Phrases: "car", "new car", "used car".

All the phrases should be thoroughly discussed and precisely defined. This latter task has to be done mainly by the Domain Expert. Thus, the Design Expert should not influence the Domain Expert here. The first, textual definition should be given by the Domain Expert, and then it should be corrected during a non-directive conversation between the Experts.

In larger systems, the Noun Phrases should be classified in groups. Each of the groups should cover certain aspect of the problem domain and be named appropriately. This activity will ease the future activity of defining Scenarios.

### 4.3.3. Verb Phrase Vocabulary.

The Verb Phrase Vocabulary defines all the actions performed by the Actors and performed inside the system (or more precisely - actions performed by or on the entities defined by Noun Phrases). The main techniques used here are taken from the construction of the Noun Phrase Vocabulary. The Verbs can be defined during the linguistic analysis and communication between Experts as above. This time however we seek for verbs not nouns. The elicted verbs should be matched with appropriate Noun Phrases or/and Actors. Each of the Verbs in the vocabulary should thus have a list of matching Nouns from the Noun Phrase Vocabulary. This means that one Verb may describe operations on different Nouns. The semantics of the actions depicted by a Verb should be common for all the associated Nouns. Thus, a common definition of a Verb Phrase should be prepared. However, some individual descriptions for each Noun are acceptable.

### 4.4. Task and Narrative Models.

The Task and the Narrative Models we present here are the core of the whole requirements specification. It is the main objective of the requirements phase to discover and narrate system tasks. The Task Model defines all the main tasks of the proposed system. The tasks should be named and described briefly. There can also be prepared a task priority matrix, which would specify the importance of tasks and order of their delivery. In this work we will not give any specific notation and techniques for the Task Model. The specification of this important part of the final deliverable Requirements Document is left to future development of the method.

The Narrative Model of the requirements phase is based on Actors and Scenarios (narrations of system tasks). Here we assume that the Actors are already listed and defined during previous activities. At this point we have to list and specify Scenarios (scripts, use cases [Jacobsen, 1992]) for each of the Actors.
4.4.1. List of Scenarios.

The construction of the Narrative Model starts from elicitation of all the system tasks. The Requirements Statement may be used and analysed during this activity, but the main source of information should come from non-directive interviews. To find system tasks we have to find all the sequences of events which are performed by or on Actors. These sequences usually can be described as single interactions of an Actor with the system (equivalent to system tasks). System tasks have to be thus named and defined. It is common that the interactions can be found already in the Verb Vocabulary. In such case the definition from the Vocabulary is taken. In other cases, the Vocabulary has to be amended, i.e. the definition of the task/interaction added.

After we find a list of tasks for each of the Actors found in step 4.3.1, we can consider reiterating the procedure of finding the Actors. This procedure can be recommended if one Actor has more tasks than about 7 ± 2. In such case it can be suggested to divide the Actor into several separate entities, or in other words, split a complex role into at least two simpler roles. This split can become beneficial when specifying Scenarios, and when writing user manuals. From the point of view of a user it allows to simplify the process of learning the system. From the point of view of the development team it enables to handle all the Scenarios of one Actor by a single developer, and keep the quantity of abstractions in reasonable boundaries.

The above activities lead to specification of a final list of tasks for every Actor in the Actor Vocabulary. After completing the lists, these interactions are treated as future Scenarios. At this point in the requirements phase we should have a complete list of Actors, a complete list of Scenarios for each of the Actors, and two additional auxiliary Vocabularies (Noun and Verb). With these elements we can now start the final part of the task model, i.e. the specification of Scenarios.

4.4.2. Specification of Scenarios.

Specification of a Scenario consists of a name of an Actor(s) involved, the name of this Scenario, and its textual description. The first parts of the Scenario specification are already available. Now, when preparing the textual description we should make an extensive use of the Noun and Verb Vocabularies.

Each Scenario description should define a single course of events during an interaction of an Actor with our system. This course of events should be specified as precisely as possible. This precision should mean that we do not use any names that are not yet defined elsewhere or which may have several interpretations. Thus, a Scenario description should consist of several concise sentences which use verbs and nouns already defined during previous activities. A restriction that should be set during the present activity is that all the names of entities and actions should be already present in one of the three Vocabularies. If during this step we find an entity or action that is not defined there, it means that we should reiterate the activity of finding Nouns and Verbs for this particular fragment of the system.

It is important to use a proper notation for the Scenario description. The notation should accent the presence of all the formally defined elements. The description is prepared in textual form, and thus the accents should be made by marking certain fragments of text. All the occurrences of an Actor name in text should be typed in a different font, e.g. Car Seller. All the Noun Phrases from the Vocabulary should be bolded (e.g. Used Car, New Car), and all the Verb Phrases - underlined (e.g. Enter Data, Change Registration). Such marking will allow for easy visual verification of the Scenario description.

It is important that the Scenario description sentences are built with a restricted grammar that would enforce the discovery process. Such a grammar was proposed in [Graham, 1995] (see also [Simons, 1996]). This grammar allows to write sentences in the following format:

Subject, Verb, Direct-object, [Preposition, Indirect-object]

hence its name: SVD(PI). Such a format has an effect of restricting the way in which the interactions can be described. Applying this grammar means that "over-complex system interactions must be reconsidered if they do not meet the restrictions of SVDPI. The developer is continuously forced to invent new vocabulary until system interactions can be described in simple terms satisfying SVDPI" (see [Simons, 1996]).

All computer systems should be designed to handle abnormal courses of events. As we mentioned before, it has to be assumed that the Actors’ behaviour is unpredictable, and they may interact with the system in an unusual manner. Such exceptional courses of events should be also considered when defining Scenarios. Thus, a Scenario definition should describe not only the main, normal course of events but also contain separate scripts for exceptional cases. All the deviations from the normal sequence should be listed as a part of Scenario description. The course of events for abnormal cases should be described in the same manner as for normal sequences.

The resulting deliverable of a Scenario should contain the following elements:
1. The name of the main Actor participating in this Scenario.
2. The name of the Scenario.
3. Narrative description of event sequence. The narrative description should be divided into several paragraphs. The first paragraph should describe the normal sequence of events, and can be entitled "Main track". The other paragraphs should narrate the exceptional courses of events ("Alternate track" 1, 2, ...). Each of such tracks should be named appropriately. The name should reflect the reason for the exception, etc.

4.5. Requirements Cross-Checks.

The requirements phase should capture all the functional features of the future system. Thus it is very important to make sure that the resulting Requirements Document (see section 4.6) complies with the real user requirements. This final document should be consistent, i.e. the Narrative Model has to be cross-checked with the Requirements Statement, and also discussed with the Domain Expert in direct conversation.

It is rather not possible to automate the process of checking the Requirements Document. These checks have to be based on the experience of the Domain Expert, and communicational capabilities of both Experts. However, we can perform a certain kind of mechanical cross-check assuming that the Requirements Statement is complete and exact. In order to check the Requirements Document we have to walk through all the Scenario descriptions. When reading the Scenarios we should cross-out the sentences of the Requirements Statement that are covered by the read parts of the Scenarios. This process will ensure eventually that the Requirements Statement is met by the Scenarios. At the end, if we find a sentence that is not crossed-out, we should reiterate the process of defining the relevant Scenario description. It has to be noted that after completing the above activity we cannot be sure about the correctness of our specification of requirements.


The previous sections contain description of activities during the requirements phase. Now we will summarise and systematise these activities. We will also specify the contents of the Requirements Document which is an effect of the listed activities. Each of the activities listed here will have an appropriate deliverable which will constitute a part of the overall document.

4.6.1. Preparation of the Requirements Statement (Activity R1) - option.

**Techniques**
1. Textual expression of Domain Expert’s experience.

**Deliverable**
1. Requirements Statement (R1-1): A basis for all other activities. Textual document prepared by the Domain Expert, describing the required system functionality and other non-functional requirements.

**Notation**
1. Text.

4.6.2. Construction of the Vocabularies (Activity R2).

**Techniques**
2. Non-directive interviewing.

**Deliverables**
1. Actor Vocabulary (R2-1): Definitions of all the Actors of the problem domain.
3. Verb Phrase Vocabulary (R2-3): Definition of all the actions that can be performed in the problem domain.

Notation
1. Text. List of appropriate elements (Actors, Nouns, Verbs) with their textual definitions.
2. Lists ordered alphabetically. In larger systems, lists divided into conceptual groups.

4.6.3. Definition of the Task Model (Activity R3).

Techniques
1. Non-directive interviewing.

Deliverable
1. Task Model (R3-1) - list of system tasks with task priority matrix.

Notation
Not yet determined.

4.6.4. Definition of the Narrative Model (Activity R4).

Techniques
1. Non-directive interviewing.
2. Vocabulary analysis.

Deliverables
1. Narrative Model (R4-1): list of Scenarios, i.e. all the Actor interactions with the system for each of the Actors, together with their detailed specifications.

Notation
1. Text in SVDPI format. List of all the Scenarios with narrative descriptions.
2. Entities taken from the Vocabularies highlighted in text by different fonts:
   - Actors - monotype font with size 2 points larger than the rest of text, e.g. Car Seller;
   - Noun Phrases - bolded, e.g. New Car;
   - Verb Phrases - underlined, e.g. Enter Data.
3. Narrative description (for a single Scenario):
   - Each course of events constitutes a separate entitled paragraph.
   - The first paragraph (entitled: Main track) describes the normal course of events. The other paragraphs (entitled: Alternate track 1,2,...) describe abnormal cases and are entitled appropriately.

4.6.4. Requirements Cross-check (Activity R5).

Techniques
1. Scenario walk-through.

4.7. Reuse of Requirements.

The object-oriented paradigm stresses the need of design and code reuse. However, also the reuse of requirements can be suggested. With the model of requirements presented here, it is possible to implement this process. Each of the Requirements Documents can be treated as a model of a single problem domain with certain boundaries set by the initial Requirements Statement. Such a model can be later reused for systems that serve in similar or extended problem domains. The requirements can be reused when changing (adding to) the functional characteristics of the respective system. In a software company, the reuse of requirements can be formalised by introducing a special library where the Requirements Documents would form an "interface" for choosing the right system framework. Such a library will be described later in this section.

The analysis of requirements reuse should be a starting point when a company creates a system in the same or similar problem domain as for a system already created before. In most methods of object oriented
development the suggested procedure is to start the reuse from the level of library classes (see section 6.2), which can be done only in the later design stage. This means that reuse is introduced only just before the end of the software development process (not including maintenance). To overcome this weakness, the present method proposes techniques for reuse that can be implemented right from the beginning. The inclusion of reuse to all the stages of the software development means that we can reuse not only the models of some of the stages, but also the process of moving from stage to stage.

4.7.1. Software Architecture Library.

As it was mentioned, to formalise the mechanisms of reuse, a software company may introduce a special library which we will call the Software Architecture Library (SA Library, SAL). This Library would be "indexed" through the Requirements Documents prepared according to the present chapter. On the level introduced in this section we may assume that the SA Library contains just the Requirements Documents, however it should actually contain the documentation of the whole development processes. Thus, later in this work we will name the elements of the SA Library as Software Architectures (SA) or simply Architectures. Each of the Software Architectures should contain four Levels (corresponding with the Discovery’s four Phases):

- Requirements Level,
- Analysis Level,
- Design Level, and
- Implementation Level.

Each of the four Levels listed above contain just the appropriate final documents of consecutive Phases. As an example of the above terminology we can state that a suitable Requirements Document forms the Requirements Level of some Software Architecture. This terminology is used to prevent confusion later in this work and to distinguish between the Requirements (or other) Document of a newly developed system and the Requirements Document of a system found in the SA Library.

The structure of the Library depends on the needs of the development company and will not be precisely specified here. However, similar problem domain descriptions should be grouped according to the subject. Every Software Framework that is based on a previously developed Framework should have a link with this predecessor. The link should describe the changes made (enhancements, changed functionality, etc.). The library would thus have a clustered structure, where each of the clusters consists of several SFs and forms a graph of links. Obviously, each of the Frameworks should have a descriptive name which would allow for introductory identification of the modelled problem domain.

4.7.2. Reuse Process.

When applying the reuse and using the SA Library we have to be careful not to violate the important guiding rule of non-directive interviewing. Thus it is vital not to start the reuse process too soon in order not to impose any presumptions on the Domain Expert. Under this assumption we could simply state that we should forbid reusing the Requirements Level, as this will certainly force the Domain Expert into some previous solutions which may not really suit the current needs. In fact, we can suggest not to reuse the requirements when developing totally new systems, perhaps for new users, especially when they are not certain about their needs. This suggestion should be applied even if we have already developed a system in the same or similar problem domain. In this worst case, the only part of the previous system’s Requirements Level that can (and should) be considered for reuse during the Requirements Phase is the Vocabulary.

Reusing the old Vocabulary is vital if we intend to reuse the remaining Levels of an Architecture already existing in the Library. At the same time the Design Expert should not impose specific Vocabulary phrases on the Domain Expert. Thus, the old Vocabulary should be at first accessible only to the Design Expert. The old Phrases can be introduced and substitute new Phrases only after the Domain Expert defines the Vocabulary for the new system.

In many cases we don’t have to be so restrictive in reusing the requirements. This is especially so when we intend to create a new version of the existing system. In such case we are actually forced into using previous concepts and solutions, and non-directive interviewing can be applied only to the changed or new parts of the system. Reuse of requirements allows us to conform with the previous vocabulary and continue previous communication between Experts.

When we intend to reuse the requirements we should start right after finishing activity R1-1. The Design Expert, together with the Domain Expert should browse the SF Library seeking for the Requirements Frameworks that would describe a problem domain similar to the currently considered one.
The level of reuse depends on the similarity between the current problem domain and the one found in the SF Library. If our considered system will be an enhancement of a previously developed one, the Experts should seek for the Requirements Framework of the previous system. In this most convenient case the previous Requirements can be used entirely. The only activity left in the requirements phase is to extend the list of Scenarios and define these additional courses of events.

Another case is when we find in the Library a Requirements Framework describing the same problem domain, but with a different functionality. In this situation more changes have to be made to the found Requirements Framework. Most probably all the three Vocabularies will remain the same, although perhaps some additional entities may have to be introduced. The main changes will have to be made to the list of Scenarios, and to the Scenario descriptions. The level of changes depends on the difference between the functionalities of the two considered systems. However, it is very probable that many of the old Scenarios could be reused entirely or with slight modifications.

It is probable that the Experts will find in the Library a Requirements Document that covers only part of the problem domain of the new system. In this case the changes to the old Document will be the biggest, although some level of reuse will be still possible. Usually many of the Actors and other Vocabulary entities may be used in the new system. Also some of the Scenarios might find their place there.

After finding the most similar Requirements in the SF Library and eliciting the elements that can be reused, we have to write the final Requirements Document for the new system. The reused elements have to be clearly marked. This marking will be used later when linking the elements reused in the requirements with appropriate classes or contracts/methods of the Analysis and Design phases (see 5.6, 6.2). The suggested notation is to place the name of the reused entity or scenario in bolded curly brackets, e.g. [New Car]. After transferring the reused elements to the new Requirements Document we should follow activities R2 and R3 to complete the Vocabularies and Scenarios. An important activity here are the cross-checks. The final Requirements Document of the new system should be carefully cross-checked with the new Requirements Statement. This will ensure that the reused elements actually fulfill the user requirements, and the newly added elements fill (or change) the boundaries of the problem domain as requested by the user.

The reuse mechanism described in this section is just the starting point for the reuse of the whole Software Architecture. The necessary links from the reuse of requirements to the reuse of analysis, design and implementation will be made in the following chapters. It can be added that the method of reuse as defined in this work is also suitable and should be applied during the maintenance of the software systems. In many cases maintenance consists in incremental development and systematic slight changes of the system functionality. This means that each of the mentioned maintenance steps should include changes or additions to the list of Scenarios, definitions of entities and other changes as described in this section. At the requirements level we can treat a maintenance step as a development of a new system with the reuse of almost entire old Requirements Document. Obviously, not all the maintenance activities should involve changing the Requirements Document. Some of them that involve the correction of bugs or changing some hardware-dependent issues can be performed at lower levels (implementation, design).

II. Case Study.

4.8. Requirements Statement - Deliverable R1-1.

The Requirements Statement we will present here is based on an internal document prepared by EASAMS Ltd. The description presented below is an abbreviated version of this document. The abbreviations had to be done due to a comparatively large volume of the EASAMS document, and because of certain confidentiality and copyright constraints.

4.8.1. In-Flight Cabin Services system.

The main purpose of the system is to provide leisure activities to the passengers of long-haul flights. The system consists of several parts that cooperate together to form the overall structure. The main three parts are:
- Flight Attendant Applications;
- Passenger Applications;
• Automated Applications (in the Cabin File Server).
Here we will analyse just the functionality of the Passenger Applications’ part. It is assumed that these applications may cooperate with the other parts of the system which are capable of communicating with the ground, performing system administration, keeping record of all the transactions, etc.

The system contains the following passenger applications: Shopping Application, Faxgram Application, Prize Draw Application, Quiz Application, News Application, Survey Application, Car/Hotel Reservation Application.

4.8.2. Shopping Application.

The Shopping Application should allow the passenger to purchase goods from the airline’s hardcopy catalogue distributed in the aircrafts. The orders would be transmitted to the ground and processed by a third-party Shopping Fulfilment House (SFH). The ordered goods would be delivered to the delivery address specified by the passenger. Payments could be made by a credit card and a Credit Card Organisation (CCO) should be involved in authorising the transaction.

- **Information capture** - The passenger will select items from a scrolling list matching the product names and descriptions from the hard-copy catalogue. Multiple choices can be made.
- **Submission of order** - After choosing all the goods the passenger is prompted for a delivery address, shown the total transaction value and asked to swipe a card. The credit card details are validated with the CCO. The order is then transmitted to the SFH, which should send an acknowledgement.

4.8.3. Faxgram Application.

The Faxgram is a basic but cost-effective fax facility enabling a simple text-only message delivery to any fax in the world.

- **Creating a message** - Before the passenger creates a message, a credit card has to be swiped and a credit limit determined. The passenger may enter and edit a short piece of text.
- **Submitting a message** - After entering the message, the passenger hits the transmit button, and is prompted for the telephone number. Next, the message is transmitted to the ground and sent to the appropriate fax number. An acknowledgement message is transmitted to the passenger after a successful or unsuccessful transmission.

4.8.4. Prize Draw Application.

This application allows the passengers to enter into a limited prize draw which operates across all the airline’s aircrafts. Each ticket purchase will be processed in a central ground system.

- **Entering the draw** - The passenger will first be informed about the rules, prizes, etc. Then the application will wait for the passenger to swipe a card and enter name and address. These details are transmitted to the ground system which assigns a random number from the available numbers. The number is then displayed to the passenger. The passenger will be informed by mail about the draw results.

4.8.5. Quiz Application.

The Quiz Application offers simple quizzes that can be played by the passengers for amusement purposes only.

- **Taking a quiz** - The system would show an image/graphic or text question and the passenger should select one of the three presented answers. The categories for a quiz can be selected by the passenger or chosen randomly by the computer.


This feature provides passengers with access to a range of news topics updated successively.
Chapter 4. Requirements Phase.

**Access to news service** - After entering the News Application, the passenger is presented with four main topic areas. When a topic is selected, a sub-menu is displayed. The sub-menu shows a number of headlines. After a headline is selected the system transmits the corresponding information from the ground. On receiving the complete information, the headline is highlighted which indicates to the passenger that it is ready to be read. The list of headlines will be kept in the Cabin File Server and updated during flight.

4.8.7. Survey Application.

Passenger surveys are used for marketing research purposes. The passengers will be encouraged to take a survey and express their impressions on various aspects of the airline’s performance.

**Taking a survey** - A passenger will be presented with a few (up to 3) pages of questions. Each question will have a line of happy/unhappy faces, and the passenger can make his/her choice by touching an appropriate face. The passenger has an opportunity to review the answers before ‘ completing’ the survey. After this the survey will be stored in a "sealed" file.


This application allows the passengers to make on-line reservations of cars and hotels through worldwide car and hotel reservation systems. The passengers can specify their precise needs, and the availability information is returned in response, with a possibility to make a booking.

**Entry to application** - Before entering the application the passenger is asked to swipe a credit card. Then, the passenger is requested to provide information about the city, room, car type, etc. All the information will be entered in a fixed-field form. The request is sent to the ground, and the ground service replies with the availability information. This information is displayed to the passenger.

**Making a reservation** - The passenger may choose the car/hotel from those available. Before making a reservation, the user is asked to confirm the data he/she has entered when preparing an availability request. After receiving a confirmation, the application displays it to the passenger together with the booking reference number.


There were found the following five Actors that interact with the Passenger Applications (PA).

- **Cabin File Server (CFS)** - the main entity that the PA interact with in order to receive/send data to the ground.
- **Credit Card Organisation (CCO)** - the entity that the PA interact with to authorise the passengers’ credit cards. Cabin File Server intervenes in these interactions.
- **Passenger** - one of the flight participants; the main user of the system.
- **Shopping Fulfilment House (SFH)** - the entity that the Shopping Application interacts with in order to place shopping orders. Cabin File Server intervenes in these interactions.
- **Telephone Application (TA)** - the entity that the Faxgram Application interacts with in order to receive information about the phone dialling codes (international and local).


There were found 137 noun phrases in the problem domain. Below they will be divided into groups one for each of the applications plus a group for credit card phrases. Due to constraints in the size of this work, only some of the groups (Shopping, Prize Draw and News) were presented to illustrate the usage and construction of vocabularies.
Abbreviations:
- CC - credit card; CCA - credit card authorisation; CCT - credit card transaction.

Attention Message - a message informing the Passenger that the application is waiting for a CC swipe.
Authorisation Code - a code issued by the CCO after a successful credit card authorisation.
Bell - a device capable of producing sound effects.
CC (Credit Card) Swipe Prompt - a message to the Passenger asking to swipe a Credit Card.
CCA Failure Message - a message to the Passenger informing that the authorisation process has failed and asking to swipe again (another card etc.).
CCA Reply - reply received from the CCO after sending a CCA Request; contains Authorisation Code, and Credit Limit.
CCA Request - request to authorise a Credit Card, sent to the CCO; contains Credit Card Details.
CCA Time-Out Message - a message to the Passenger informing that the application was waiting for a CCA Reply too long.
CCT Failure Message - a message to the Passenger informing that the transaction process has failed.
CCT Request - request to perform a Credit Card Transaction, sent to the CCO; contains the Credit Card Details, Authorisation Code, and Transaction Sum.
CCT Reply - reply received from the CCO after sending a CCT Request; contains Confirmation Code.
CCT Time-Out Message - a message to the Passenger informing that the application was waiting for a CCT Reply too long.
Credit Card (Details) - data contained in a physical credit card and obtained via swipe.
Credit Card Transaction - a payment performed using a Credit Card.
Credit Limit - amount available to the Passenger as indicated by CCO in a reply to CCA Request.
First Time-Out Signal - a signal from a timing device informing that a specified time period has elapsed for the first time.
Second Time-Out Signal - a signal from a timing device informing that a specified time has elapsed for the second time.
Swipe Acknowledgement Message - a message to the Passenger informing that the swipe was successful.
Swipe Error Message - a message to the Passenger informing that the swipe was unsuccessful, and asking to swipe again.
Swipe Time-Out Message - a message to the Passenger informing that the application was waiting for a CC swipe too long.
Transaction Sum - an amount of a Credit Card Transaction.

4.10.2. Shopping Noun Phrases.

Abort Option - option allowing to abandon a cancellation operation.
Accept Option - option allowing to accept an Order and continue with order procedure.
Add Option - option allowing to add an Item to the Order.
Cancel Option - option allowing to cancel an operation.
Confirm Option - option allowing to confirm an operation.
Delete Item Option - option allowing to delete an item from the Order.
Delivery Address - the purchaser’s address at the time of delivery.
Exit Option - option allowing to exit the application or an option.
Item - an entry denoting a product from the hardcopy catalogue.
Item Delete Options - a set of on-screen choice options; contains: Confirm Option, Abort Option.
Item Selection Options - a set of on-screen choice options; contains: Confirm Option, Cancel Option.
Long Time-Out Signal - a signal from a timing device informing that a specified long time period has elapsed.
Order Cancel Options - a set of on-screen choice options; contains: Confirm Option, Abort Option.
Order Confirmation Options - a set of on-screen choice options; contains: Confirm Option, Cancel Option.
Order (Details) - a set of ordered Items.
Order List - a scrolling list of Order Items.
Order Options - a set of on-screen choice options; contains: Accept Option, Cancel Option, Revise Option.
Order Reply Message - a message to the Passenger informing about the results of his/her shopping order request.
Order Revision Options - a set of on-screen choice options; contains: Delete Item Option, Exit Option.
Product/Supplier Code - a unique code representing the given product of a supplier.
SFH Request - request to fulfil an Order, sent to an SFH; contains a list of Product/Supplier Codes.
SFH Reply - reply received from the SFH after sending an SFH Request.
SFH Time-Out Message - a message to the Passenger informing that the application was waiting for an SFH Reply too long.
Shopping Card Transaction - a kind of Credit Card Transaction with behaviour specific to Shopping.
Shopping Instructions - information about the contents and manner of use of the shopping service.
Shopping Item List - a scrolling list of Items which allows to choose one or more of them.
Shopping Options - a set of on-screen choice options; contains: Start Option, Exit Option.
Start Option - option allowing to start the application.
Total Order Value - a sum of prices of all the items ordered during a shopping session, together with Delivery Charges and Duties/Taxes.

4.10.3. Prize Draw Noun Phrases.

Exit Option - option allowing to exit the application or an option.
Long Time-Out Signal - a signal from a timing device informing that a specified long time period has elapsed.
Passenger Address - the Passenger’s name and place of residence, where the information with draw results will be sent to.
Passenger Address Prompt - a prompt asking the Passenger to enter his/her address (see: Passenger Address).
Passenger’s Draw Number - a unique, random number given by the prize draw ground system in response to the Ticket Purchase Request: Passenger’s draw entry number.
Prize Draw Instructions - information describing the prize draw, its prizes, statutory rules, draw expire date, with compliance and promotional data.
Prize Draw Options - a set of on-screen choice options; contains: Start Option, Exit Option.
Prize Draw Time-Out Message - an information to the Passenger informing that the application was waiting for a Passenger’s Draw Number too long.
Start Option - option allowing to start the application.
Ticket Purchase Request - a data packet containing the Passenger Address with a request to enter the prize draw.


Accept Option - option allowing to confirm reading a Headline Story.
Exit Option - option allowing to exit the application or an option.
Headline - one of the news headlines
Headline List - a list of current news Headlines.
Headline Story - a text of a news story under a given Headline.
News Options - a set of on-screen choice options; contains: Exit Option.
Subject - one of the news subjects.
Subject List - a list presenting four available news Subjects.


There were found 155 verb phrases. They will be divided in the same manner as the noun phrases. Similarly, only Shopping, Prize Draw and News verb phrases will be shown.
4.11.1. Credit Card Verb Phrases.

Display Attention Message - show the Attention Message.
Display CC Swipe Prompt - show the CC Swipe Prompt.
Display CCA Failure Message - show the CCA Failure Message.
Display CCA Time-Out Message - show the CCA Time-Out Message.
Display Swipe Error Message - show the Swipe Error Message.
Display Swipe Time-Out Message - show the Swipe Time-Out Message.
Form CCA Request - prepare a CCA Request packet in an appropriate format.
Form CCT Request - prepare a CCT Request packet in an appropriate format.
Read Credit Card Details - swipe and receive Credit Card Details.
Receive CCA Reply - transmit a CCA Reply packet from the CCO.
Receive CCT Reply - transmit a CCT Reply packet from the CCO.
Receive Credit Card Details - transmit the Credit Card Detail from a card reader.
Record Authorisation Code - extract from the CCA Reply and keep the Authorisation Code.
Record Credit Card Details - keep the information contained physically in a Credit Card.
Record Credit Limit - extract from the CCA Reply and keep the Credit Limit.
Register Credit Card Transaction - cause the CCO to register the Credit Card Transaction.
Record Registration Code - keep the Registration Code in memory.
Send CCA Request - transmit the CCA Request packet to the CCO.
Send CCT Request - transmit the CCT Request packet to the CCO.
Sound Bell - cause the Bell to give a sound.
Display Swipe Acknowledgement Message - show the Swipe Acknowledgement Message.
Swipe Credit Card - the Passenger swipes a Credit Card through a card reader.
Validate Credit Card Details - check that the Credit Card Details' format is valid.
Verify Credit Card - authorise the Credit Card with the Credit Card Organisation

4.11.2. Shopping Verb Phrases.

Add Item to Order - add an Item to the current Order list.
Calculate Total Order Value - add the Item prices and add Delivery Charges with Duties/Taxes.
Confirm Order - the Passenger confirms that he/she has ordered all the goods.
Display Delivery Address Prompt - show the Delivery Address Prompt.
Delete Item - erase an Item from the Order.
Display Item Delete Options - show the Item Delete Options.
Display Item Selection Options - show the Item Selection Options.
Display Order - show part of the Order on the screen, prepare for scrolling.
Display Order Options - show the Order Options.
Display Order Cancel Options - show the Order Cancel Options.
Display Order Confirmation Options - show the Order Confirmation Options.
Display Order Revision Options - show the Order Revision Options.
Display Order Reply Message - show the Order Reply Message.
Display SFH Time-Out Message - show the SFH Time-Out Message.
Display Shopping Item List - shows part of the List on the screen, prepare for scrolling.
Display Shopping Options - show the Shopping Options.
Display Total Order Value - show the Total Order Value on the screen.
Form SFH Request - prepare an SFH Request transmission packet in appropriate format.
Form Shopping Card Transaction - prepare a Credit Card Transaction using appropriate shopping data.
Input Delivery Address - key in the Delivery Address.
Place Order - send Order Details to the SFH.
Prepare Order - choose Items to be ordered from the SFH.
Present Shopping Instructions - allow the Passenger to browse the Shopping Instructions.
Receive SFH Reply - transmit an SFH Reply packet from the SFH.
Record Delivery Address - keep the Delivery Address in memory.
Revise Order - scroll through an Order List and possibly delete Items.
Select Item from Order List - choose one of the Items from an Order List.
Select Item from Shopping Item List - choose one of the Items from a Shopping Item List.
Select Abort Option - choose the on-screen Abort Option.
Select Accept Option - choose the on-screen Accept Option.
Select Add Option - choose the on-screen Add Option.
Select Cancel Option - choose the on-screen Cancel Option.
Select Confirm Option - choose the on-screen Confirm Option.
Select Delete Item Option - choose the on-screen Delete Item Option.
Select Revise Option - choose the on-screen Revise Option.
Select Start Option - choose the on-screen Start Option to start the application.
Select Exit Option - choose an on-screen Exit Option to exit the application or an option.
Send SFH Request - transmit the SFH Request packet to the SFH.

4.11.3. Prize Draw Verb Phrases.

Display Passenger’s Draw Number - show to the Passenger his/her draw entry number.
Display Prize Draw Options - show the Prize Draw Options.
Display Prize Draw Time-Out Message - show the Prize Draw Time-Out Message.
Display Passenger Address Prompt - show the Passenger Address Prompt.
Form Prize Draw Card Transaction - prepare a Cred. Card Transaction using appropriate prize draw data.
Form Ticket Purchase Request - prepare the Ticket Purchase Request using appropriate format.
Input Passenger Address - key in the Passenger Address.
Receive Passenger's Draw Number - transmit the Passenger’s Draw Number from the Cabin File Server.
Present Prize Draw Instructions - allow the Passenger to browse the Prize Draw Instructions.
Record Passenger Address - keep the Passenger Address in memory.
Send Passenger Address - pass the Passenger’s Address to the cabin file server with a request to enter a prize draw
Send Ticket Purchase Request - transmit the Ticket Purchase Request to the Cabin File Server.

4.11.4. News Verb Phrases.

Display Headline Story - show the current Headline Story.
Display News Options - show the News Options.
Present Headline List - show the current Headline List on the screen.
Present Subject List - show the Subject List on the screen.
Receive Headline List - transmit a Headline List for the current Subject from the Cabin File Server.
Receive Headline Story - transmit a Headline Story for the current Headline from the Cabin File Server.
Select Headline - choose one of the Headlines from the Headline List.
Select Subject - choose one of the Subjects from the Subject List.


With this deliverable we propose a notation for structuring scripts which is in accordance with guidelines given in section 4.4.2. Each scenario is treated as a ‘story’ with one beginning, but with different endings. Each ‘ending’ is denoted by a separate track of a scenario. The places, where the ‘story’ may have a different ending are denoted by indented title of the alternate ending in angle brackets, e.g. <Alternate track 1 - order cancelled>. Each track (‘story’) ends with a ‘happy’ or ‘sad’ ending, which is denoted by exit marks, e.g. <Exit =OK>, <Exit =Failure>, etc. When a track description contains sub-scenarios, it is denoted by the symbol of that sub-scenario in curly brackets, e.g., {S1}. If a track repeats certain actions, it can be expressed by placing a <Repeat> symbol at the end of this track, which returns to a <#> symbol which should be placed somewhere in the scenario. In the author’s opinion it shouldn't be very difficult for the user (Domain Expert) to understand this notation, especially when a ‘story telling’ analogy is explained.

Scenario C1 - Application reads the Credit Card Details.
Actors - Passenger, Credit Card Organisation.

Main track.
Application displays a CC Swipe Prompt.
<Alternate track 1 - time-out occurs>
Passenger swipes a Credit Card.
Application receives Credit Card Details.
<Alternate track 2 - card validation is unsuccessful>
Application validates Credit Card Details. =OK=
Application records Credit Card Details.
Application displays a Swipe Acknowledgement Message.
<Exit =OK=>

Alternate track 1 - time-out occurs.
Application receives a First Time-Out Signal.
Application sounds a Bell.
Application displays an Attention Message.
<Alternate track 3 - time-out occurs second time>
Application validates Credit Card Details. =OK=
Application records Credit Card Details.
Application displays a Swipe Acknowledgement Message.
<Exit =OK=>

Alternate track 2 - card validation is unsuccessful.
Application validates Credit Card Details. =Error=
Application displays a Swipe Error Message.
<Exit =Error=>

Alternate track 3 - time-out occurs second time.
Application receives a Second Time-Out Signal.
Application sounds a Bell.
Application displays a Swipe Time-Out Message.
<Exit =TimeOut=>

Scenario C2 - Credit Card Organisation verifies Credit Card Details.
Actor - Credit Card Organisation.

Main track.
Application forms a CCA Request.
Application sends the CCA Request to the Credit Card Organisation.
<Alternate track 1 - time-out occurs>
<Alternate track 2 - authorisation fails>
Application receives a CCA Reply from the Credit Card Organisation. =OK=
Application records Authorisation Code.
Application records Credit Limit.
<Exit =OK=>

Alternate track 1 - time-out occurs.
Application receives a Long Time-Out Signal.
Application displays a CCA Time-Out Message.
<Exit =TimeOut=>

Alternate track 2 - authorisation fails.
Application receives a CCA Reply from the Credit Card Organisation. =Failure=
Application displays a CCA Failure Message.
<Exit =Failure=>
Scenario C3 - Credit Card Organisation registers a **Credit Card Transaction**.
Actor - Credit Card Organisation.

Main track.

Application forms a **CCT Request**.
Application sends the **CCT Request** to the Credit Card Organisation.

<Alternate track 1 - time-out occurs>
<Alternate track 2 - registration fails>

Application receives a **CCT Reply** from the Credit Card Organisation. =OK=
Application records **Registration Code**.
<Exit =OK=>

Alternate tracks similar to those of Scenario C2.

---

Scenario C4 - Application cancels a **Credit Card Transaction**.
Actor - Credit Card Organisation.

Main track.

Application forms a **CCC Request**.
Application sends the **CCC Request** to the Credit Card Organisation.

<Alternate track 1 - time-out occurs>
<Alternate track 2 - cancellation fails>

Application receives a **CCC Reply** from the Credit Card Organisation. =OK=
Application records **Cancellation Code**.
<Exit =OK=>

Alternate tracks similar to those of scenario C2.

---

### 4.12.2. Shopping Scenarios.

**Scenario S1** - Passenger does **Shopping**.
Actor - Passenger, Credit Card Organisation, Shopping Fulfilment House.

Main track.

Application presents the **Shopping Instructions**.
Application displays **Shopping Options**.

<Alternate track 1. shopping abandoned>

Passenger selects the **Start Option**.
<Alternate track 2. order cancelled>

[S2] Passenger prepares an **Order**. =OK=
<#> <Alternate track 3. time-out occurs>
<Alternate track 4. failed to read credit card>

[C1] Application reads **Credit Card Details**. =OK=
<Alternate track 5. time-out occurs>
<Alternate track 6. failed to verify credit card>

[C2] Credit Card Organisation verifies the **Credit Card**. =OK=
<Alternate track 7. order not confirmed>

[S4] Passenger confirms an **Order**. =OK=
<Alternate track 8. order not placed>

[S5] Application places the **Order** with the Shopping Fulfilment House. =OK=
Application forms a **Shopping Card Transaction**.
<Alternate track 9. transaction not registered>

[C3] Credit Card Organisation registers a **Credit Card Transaction**. =OK=
<Exit =OK=>

Alternate track 1 - shopping abandoned.
Passenger selects the **Exit Option**.
<Exit =OK=>

Alternate track 2 - order cancelled.

[S2] Passenger prepares an **Order**. =Cancel=
<Exit =OK=>
Chapter 4. Requirements Phase.

Alternate track 3 - time-out occurs.
   {C1} Application reads Credit Card Details. =TIMEOUT=
   <Exit =OK=>
Alternate track 4 - failed to read credit card.
   {C1} Application reads Credit Card Details. =Failure=  
   <Repeat #>
Alternate track 5 - time-out occurs.
   {C2} Credit Card Organisation verifies the Credit Card. =TIMEOUT=
   <Exit =OK=>
Alternate track 6 - failed to verify credit card.
   {C2} Credit Card Organisation verifies the Credit Card. =Failure=  
   <Repeat #>
Alternate track 7 - order not confirmed.
   {S4} Passenger confirms an Order. =Failure=  
   <Exit =OK=>
Alternate track 8 - order not placed.
   {S5} Application places the Order with the Shopping Fulfilment House. =TIMEOUT=
   <Exit =OK=>
Alternate track 9 - transaction not registered.
   {C3} Credit Card Organisation registers a Credit Card Transaction. =TIMEOUT=
   <Exit =OK=>

Scenario S2 - Passenger prepares an Order.
Actor - Passenger.

Main track.
Application displays a Shopping Item List.
Application displays Order Options.
<#> Passenger selects an Item from the Shopping Item List.
   <Alternate track 1. order is cancelled>
   <Alternate track 2. order is accepted>
   <Alternate track 3. order revised>
Passenger selects the Add Option.
Application displays Item Selection Options.
   <Alternate track 4. item not confirmed>
Passenger selects the Confirm Option.
Application adds the Item to the Order.
   <Repeat #>
Alternate track 1 - order is cancelled.
   Passenger selects the Cancel Option.
Application displays Order Cancel Options.
   <Alternate track 5. cancellation not confirmed>
Passenger selects the Confirm Option.
   <Exit =Cancel=>
Alternate track 2 - order is accepted.
   Passenger selects the Accept Option.
   <Exit =OK=>
Alternate track 3 - order revision.
   Passenger selects the Revise Option.
   {S3} Passenger revises the Order.
   <Repeat #>
Alternate track 4 is similar to track 2.
Scenario S3 - Passenger revises an Order.
Actor - Passenger.

Main track.
Application displays the Order.
Application displays Order Revision Options.
># Passenger selects an Item from the Order List.
<Alternate track 1. exit revised>
Passenger selects the Delete Item Option.
Application displays Item Delete Options.
<Alternate track 2. deletion not confirmed>
Passenger selects the Confirm Option.
Application deletes the Item.
<Repeat #>
Alternate track 1 - exit revision.
Passenger selects the Exit Option.
<Exit =OK=>
Alternate track 2 - deletion not confirmed.
Passenger selects the Abort Option.
<Repeat #>

Scenario S4 - Passenger confirms an Order.
Actor - Passenger

Main track.
Application prompts for Delivery Address.
Passenger inputs the Delivery Address.
Application records the Delivery Address.
Passenger revises the Order.
Application calculates the Total Order Value.
Application displays the Total Order Value.
Application displays Order Confirmation Options.
<Alternate track 1. order not confirmed>
Passenger selects the Confirm Option.
<Exit =OK=>
Alternate track 1 is obvious.

Scenario S5 - Application places an Order with the Shopping Fulfilment House.
Actor - Shopping Fulfilment House (SFH).

Main case.
Application forms an SFH Request.
Application sends the SFH Request to the Shopping Fulfilment House.
<Alternate track 1. time-out occurs>
Application receives an SFH Reply from the Shopping Fulfilment House.
Application displays an Order Reply Message.
<Exit =OK=>
Alternate track 1 - time-out occurs.
Application receives a Long Time-Out Signal.
Application displays an SFH Time-Out Message.
<Exit =Timeout=>


Scenario P1 - Passenger enters Prize Draw.
Actors - Passenger, Credit Card Organisation.
Main track.
Application presents the Prize Draw Instructions.
Application displays the Prize Draw Options.
<Alternate track 1. prize draw abandoned>
Passenger selects the Start Option.
<#>  <Alternate track 2. time-out occurs>
<Alternate track 3. failed to read credit card>
{C1} Application reads Credit Card Details. =OK=
<Alternate track 4. time-out occurs>
<Alternate track 5. failed to verify credit card>
{C2} Application verifies the Credit Card with the Credit Card Organisation. =OK=
Application displays the Passenger Address Prompt.
Passenger inputs the Passenger Address.
Application records the Passenger Address.
<Alternate track 6. ticket purchase not sent>
{P2} Application sends the Passenger Address to the Cabin File Server. =OK=
Application forms a Prize Draw Card Transaction.
<Alternate track 7. transaction not registered>
{C3} Application registers a Credit Card Transaction with the Credit Card Organisation.
<Exit =OK=>
Alternate tracks similar to those of Scenario S1.

Scenario P2 - Application sends the Passenger Address to the Cabin File Server.
Actor - Cabin File Server.

Main track.
Application forms a Ticket Purchase Request.
Application sends the Ticket Purchase Request to the Cabin File Server.
<Alternate track 1. time-out occurs>
Application receives a Passenger’s Draw Number
Application displays the Passenger’s Draw Number
<Exit =OK=>
Alternate track 1 - time-out occurs.
Application receives a Long Time-Out Signal.
Application displays a Prize Draw Time-Out Message.
<Exit =TimeOut=>


Scenario N1 - Passenger reads News.
Actors - Passenger, Cabin File Server.

Main track.
<#> Application presents a Subject List.
Application displays News Options.
<Alternate track 1. news abandoned>
Passenger selects a Subject.
Application receives a Headline List from the Cabin File Server.
Application presents the Headline List.
<Alternate track 1. news abandoned>
Passenger selects a Headline.
Application receives a Headline Story from the Cabin File Server.
Application displays the Headline Story.
<Alternate track 1. news abandoned>
Passenger selects the Accept Option.
<Repeat #>
Alternate track 1 - news abandoned.
Passenger selects the Exit Option.
<Exit =OK=>
5. Analysis Phase.

1. Notation and Activities.

In the Discovery method, the Analysis Phase is a direct continuation of the Requirements Phase. However, this new phase introduces a change of approach in the system development. From now on, detailed "engineering oriented" techniques are used. The Domain Expert still plays an important role but now the Design Expert is the active party. The Analysis Phase is based around Design Expert’s building of models of overall system structure and behaviour, which are reflected back and verified by the Domain Expert.

As we can see, there appears a difference in communication between Experts. The first phase tries to help the Domain Expert to formulate his view of the problem domain without any presumptions based on the Design Expert’s knowledge about software systems. Now, in the second phase, appropriate modelling techniques are used to achieve common and more formal understanding of the system by the Experts. The main postulate is that the analysis models have to be still understandable by the Domain Expert, which should verify them with his/her view of the problem domain. Additionally, the final Analysis Document as specified in section 5.5 should be formally cross-checked to be compliant with the Requirements Document.

The main task of the Analysis Phase is to discover appropriate objects from the problem domain. Most of the existing methods leave this task to the intuition and experience of the system analysts. Instead, they concentrate on defining fancy notations to describe the found objects and relations between them. The process of Discovery’s Analysis Phase defined in the following sections tries to overcome this weakness.

It has to be emphasised that the approach of Discovery to the requirements analysis is behaviour-centred not data-centred. This means that objects are elicited from the problem domain by grouping actions and not by grouping attributes. Thus, the most important of the analysis models described below is the Collaboration Model, where objects are defined by their responsibilities and not by their data. The Data Model which uses entity-relationship diagrams is treated as a secondary model, which can be created for data-oriented applications.

5.1. Interaction Model.

Construction of the Interaction Model constitutes a link between the Requirements Document and the Analysis Phase. During this development step, the Design Expert should seek for objects that can be elicited from the requirements Scenarios. The discovery of objects is based on the analysis of system tasks described by Scenarios.

For each of the Scenarios an Interaction Diagram (ID) is built. The ID notation is similar to that introduced by Jacobson [Jacobson, 1992] for his Object Interaction Diagrams, although that in his OOSE method it was used in the design ("construction") stage. In order to build an Interaction Diagram, the given Scenario is written one clause at a time (per line) along the y-axis of the diagram (see Fig. 2). The y-axis represents the system boundary (interface). Each Scenario clause is now treated as an event (stimulus) that should be handled by one of the objects (not yet defined).

By analysing one stimulus at a time we should try to elicit the first pool of objects. This first pool comes thus directly from the problem domain. These obvious choices for objects can be found already in the Noun Phrase Vocabulary. Each of the stimuli from the analysed Scenario should have a specified client and a specified server entity of that action. If the stimulus comes from an Actor, this Actor should become the client of that stimulus, and some object should become a server. If an Actor receives a stimulus from the system, the situation is opposite. The same situation is for the stimuli that are wholly contained inside the system. In this case one of the objects, the one that requests a service, is the client, and another object is the server.
### Figure 2. Interaction Diagram - notation.

The objects that are elicited from the problem domain for a given Scenario should be listed along the x-axis (Fig. 2) of the Interaction Diagram. Each object has a vertical line drawn along the entire diagram until its bottom. The stimuli are drawn as horizontal arrows leading from one object line to another. The stimuli from the Actors (external entities) are denoted by arrows leading from a relevant Scenario clause to an object line and crossing the y-axis (the system boundary). The stimuli coming to the Actors cross the system boundary in the opposite direction.

It is common that we cannot find a proper Noun Phrase in the problem domain to describe an object that should be a client of an ID stimulus. This means that an object that does not belong to the problem domain should be discovered (an “esoteric” object). The new objects should be defined to handle groups of stimuli, and so that they would describe a reasonable abstract behaviour. At this moment some intuition and experience of the developer has to be applied in order to find proper object abstractions.

The above process means that the objects we find are naturally based on their behaviour. This is due to the fact that we analyse the problem domain from the point of view of actions, not data. The discovery process is initiated with every clause of the Scenarios. Each of these clauses is treated as a stimulus, and an object that we define to be the server of that stimulus is then assigned a responsibility for handling it and similar stimuli.

If during analysis of a Scenario we discover an object that we believe is suitable for being a server for a particular stimulus (or stimuli), we should try to define the general responsibilities of that object. The definition of responsibilities of an object is described in section 4.2., and this is the moment where the Interaction Model is intersected with the Collaboration Model (see next section). Thus, the processes of the two models’ construction should be conducted in parallel and both of the processes should support each other.

Each of the object that we find and define its responsibilities should be taken aside and listed. Of course, the responsibilities of an object are yet defined very generally. They should be specified more precisely after the analysis of other Scenarios (see next section). Later in the analysis of Scenarios we may find that a yet undiscovered object should be responsible for handling a stimulus that fits into the range of responsibilities of some already defined object. In such case we can simply take that existent object and use it in the Interaction Model as it was described above. At the same time we can amend the object definition with the additional responsibility. This way we can build the Collaboration Model along with the Interaction Model.

Although the process of discovering and defining objects described above consists in building two parallel models, we may note that the transitions between construction of models are to a fair extent automatic. When we cannot find any reasonable object in the Collaboration Model to handle the current stimulus in a Scenario, the only activity we can do is to discover and define some new object. This way we have to shift from the construction of the Interaction Model to the construction of the Collaboration Model. Also, finding a stimulus that is suitable to be handled by an existing object shifts us to the construction of responsibilities in the Collaboration Model.

An important aspect of the models constructed in this first part of analysis is that all the elicited objects have certain behaviour. The analysis of Scenarios and Interaction Diagrams assures us that all the objects placed on the x-axis of the diagrams behave in some way by being a server or a client for at least one stimulus.
The final deliverable of the Interaction Model is a set of Interaction Diagrams. At least one diagram should be defined for each Scenario. The "Main track" paragraph of a Scenario description should be defined by a separate diagram, and possible alternate tracks should be described appropriately by other diagrams. The Interaction Model’s deliverable can be finalised only after several iterations and intersections with the Collaboration Model. Only after certain amount of analysis proper objects can be elicited and placed along the IDs’ x-axes. The next section describes this process in more detail.

5.2. Collaboration Model.

As it was already noted in section 4.1., the Collaboration Model is not constructed directly from the Requirements Document. Instead, the objects of the model are discovered during the construction of the Interaction Model. However, in the previous section we did not give any guidelines for finding the objects’ boundaries. We did not specify how broad should be the responsibilities handled by a single object. The element that allows us to assess the size of objects and keep their responsibilities in reasonable quantities, is the CRC form - the basic component of the Collaboration Model.

First we will define the notation of the present model. With this notation we will build a Collaboration Graph which is a final deliverable of this step, heavily influenced by the Interaction Model from the previous step. To denote the Collaboration Graph we will use the simplest possible form of a directed graph. The objects found during analysis are placed in the nodes of this graph. Each of the nodes is a rectangle with a name of an object (Fig. 3)

![Collaboration Graph](image)

**Figure 3. Collaboration Graph - notation.**

The arrows between nodes specify the graph’s direction. The object at node where the arrow starts is a client object for some stimuli sent to the object placed at the end node of this arrow. When an arrow has two ends (e.g. between Object1 and Object2 on Fig. 3), it means that both objects are mutually clients and servers for some stimuli sent between them. A set of stimuli between objects we will call a “bond”. The bonds should have their ‘binding capacities’ specified. A capacity should be denoted by a number besides an arrow head (see Fig. 3). The numbers represent the number of stimuli sent from the client object to the server object, and are placed at the server end of an arrow.

The semantics of bonds is similar to the meaning of “contracts” taken from Responsibility Driven Design [Wirfs-Brock, 1990]. A bond between two objects can be described as a set of responsibilities that one of the objects is obliged to fulfil in response to the other object’s stimuli. A bond is thus a link between two objects with a list of responsibilities delegated by one of the objects to the other. The idea of bonds allows the developers to move to a higher level of abstraction when considering inter-object communication. Bonds differ slightly from RDD’s contracts in that they do not describe partial communication between objects. There is always maximum one bond between two objects (in one direction, or in both) with the number of responsibilities (binding capacities) specified. Thus, we do not abstract over only some of the intra-object stimuli, but over all of them. This can be justified by the fact that in Discovery we have already limited the number of such stimuli to 7 (usually this number is smaller), and thus we do not need to introduce some intermediate levels of abstraction.

The Collaboration Graph shows just the overall structure of the system’s behavioural decomposition. We can just see the object names and their general collaborations. The lower, more specific level of visualisation in the Collaboration Model is constructed with the use of CRC forms/cards. This technique uses again a very simple notation. It was introduced by Beck and Cunningham [Beck, 1989] and later used

---

1 The terminology is changed in respect to RDD because the word “contract” is also used throughout the literature in the context of method specification with the use of formal assertions (pre- and postconditions) which will be used in the Implementation Model.
in RDD [Wirfs-Brock, 1990] as a low-tech way of capturing objects. The original technique uses 6" by 4" file index cards to record each object (or class) with its responsibilities and collaborators (hence its name - CRC). In our slight modification of CRC we will use CRC forms, which will allow to include up to 9 (7+2) responsibilities for each class (object). Also the number of collaborators is limited to 9. A general shape of the CRC form is shown on Fig. 4.

<table>
<thead>
<tr>
<th>Name: Object 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
</tr>
<tr>
<td>1. First responsibility - description ...</td>
</tr>
<tr>
<td>2. Second responsibility - description ...</td>
</tr>
<tr>
<td>3. ... (other responsibilities)</td>
</tr>
<tr>
<td>4.</td>
</tr>
<tr>
<td>5.</td>
</tr>
<tr>
<td>6.</td>
</tr>
<tr>
<td>7.</td>
</tr>
<tr>
<td>8.</td>
</tr>
<tr>
<td>9.</td>
</tr>
</tbody>
</table>

*Figure 4. CRC Form - notation.*

The top line of a CRC card/form contains the object/class name identical to that found in the Collaboration Graph. The left side of the CRC form is reserved for a list of services provided by the object/class, i.e. its responsibilities which are equivalent to stimuli sent to it. Each of the responsibilities should be named and described. This includes the internal responsibilities which are denoted by "turn-around" arrows in the Interaction Model. We can also note that the number of responsibilities listed may not be equal to the sum of binding capacities of all the object's bonds in the Collaboration Graph. There can be internal responsibilities, but also some of the stimuli sent from different objects may be handled by the same responsibility.

The other objects that an object uses to fulfil some of its responsibilities are called collaborators. The collaborator objects should be listed on the right side of the CRC form. The list of collaborators should be again in accordance with the Collaboration Graph. The list should contain only the objects that are servers for the current object.

The above lists allow to include up to 9 elements (responsibilities or collaborators). This number is a definite maximum, but an additional line is drawn after the 7th element. This line indicates that after listing some elements below it, the developer should start thinking about splitting an object into several other objects.

A set of CRC cards describing all the found classes together with a complete Collaboration Graph forms the final deliverable of the Collaboration Model. However, it has to be stressed that the Collaboration Model should not be built in the order given when defining the notation. A proper process of objects' discovery and CRC form definition is given below. This process interleaves with the process of building the Interaction Model (section 4.1).

The discovery of objects starts thus in the Interaction Model. The Collaboration Model’s function in this process is to amplify the developer’s capabilities and to assure proper level of abstraction. The objects found in the Interaction Diagrams should be successively listed on the CRC forms. The set of cards should grow after analysing consecutive Scenarios. Together with the growth in the number of CRC forms, the lists of responsibilities and collaborators for each of the objects should be amended. This process is also described in section 4.1.

During the analysis of Scenarios, the number of responsibilities assigned to different objects may eventually exceed 9. It takes place often during the initial stage of analysis, when the objects taken from the problem domain are yet very coarse-grained. This is the moment where the discovery process is evoked by the constraints of the CRC forms. These constraints are specified to limit the number of abstractions handled by a single object. This should enable the developers to keep within the psychological barrier of short-term human memory. When the number of responsibilities of a given object exceeds this barrier - we have to discover some other object (or objects) that would handle this excess. Otherwise the developed system would become incomprehensible and very difficult to design and maintain.

With the above process, the complexity of the system is decentralised and hidden behind the abstraction barrier of objects with reasonably distributed responsibilities. The only problem left is to define proper boundaries for the discovered objects. The new objects that come from the split of objects found during the Interaction Diagram analysis have to describe reasonable abstractions. This process should involve the presence of the Domain Expert and proper communication with the Design Expert. This time
however, the Design Expert should be the active party when defining new abstractions. This approach is justified by the fact that the new objects usually are not directly present in the problem domain.

There can be applied various methods of splitting an existing object. The easiest method is to assign some of its responsibilities to some other existing objects. However, care has to be taken in order not to create "hybrid" objects that combine responsibilities without any consistent abstraction which could bind them. Such hybrids, created extensively, can easily ruin the final analysis model. When we find a group of responsibilities that does not fit naturally into any existing other object, we should rather create a new one. The same remark can be applied to defining objects during the Scenario analysis in step 4.1.

Splitting an object does not necessarily mean that the analysed object should be deleted from our model. We can leave some responsibilities within the old object, and transfer only the rest to some other objects. Simplifying the list of responsibilities may also mean that we should specify them more generally. Such specification would allow to join some responsibilities into a more general one. However, this should not cause a loss in the model’s precision. The joined responsibilities should be assigned to some other objects which would collaborate with our current object to fulfill the more general responsibility.

After distributing the responsibilities of an “overloaded” object among other objects we result with a modified list of objects. The Collaboration Graph also changes accordingly. Usually this will cause the need to change the Interaction Model. Obviously, we need to change the objects placed in some of the Interaction Diagrams. The procedure of modification and shifting between models is thus iterative and gradual. Each change in the Collaboration Model causes changes in the Interaction Model and vice-versa. A CASE tool which would automate this process of mutual modifications would be of great benefit.

5.3. Auxiliary Models.

The construction of the two models presented in the previous sections is obligatory in Discovery. However, for some of the systems these two models may be insufficient. Some other models are then needed to express the results of analysis. Thus, Discovery introduces three additional models that can be optionally constructed. These are the Data Model, the Process Model and the State Model.

Here we will just describe the first of the models in more detail. The Data Model should be constructed if the problem domain contains extensive enough amounts of data. This usually involves creating a database and using the relational database theory. In order not to lose the benefits of this established technology we can construct a model that would allow us to capture the data structure of our system. In Discovery we will use Chen’s entity-relationship diagrams (ERDs) [Chen, 1976] to denote the Data Model. This model was already taken as a basis for various models in other OO methods. However, most of these methods used ERDs to express their main “Object Model” (see [Rumbaugh, 1991] and [Coleman, 1994]) which was used to find all of the developed system’s objects. In such an approach, the entities were equivalent to objects, and also certain enhancements to denote the semantics of object relationships were added.

In Discovery we take just the pure Chen’s notation and enrich it only with cardinality and optionality constraints. We should use this model to allow for normalisation of data tables in relational databases that we intend to create in our system. It has to be stressed that the Data Model should be built only as a link to the database, not as a way to capture objects. It is often that after normalisation, the objects of the Collaboration Model may be fragmented over several entities.

We may also note that the Data Model presented in the present version of Discovery can be the starting point to the development of a model for Object-Oriented Databases (or perhaps more correctly - Objectbases [Plant, 1992]) in the future. Object Oriented Database technology is yet in its early period of development (see e.g. [Atkinson, 1989]), but it is already promising as a natural extension of the general OO technology.

The remaining two auxiliary models are based on techniques taken from [Jackson, 1983], and [Harel, 1987]. The Process Model uses Jackson Structured Diagrams to capture the overall system architecture for more complex systems. The State Model is constructed for systems with time-dependent or event-driven features, and uses Harel State Charts. A more detailed description of applying these techniques in Discovery can be found in [Simons, 1996].
5.4. Analysis Cross-Checks.

The main purpose of the Analysis Phase is to represent the requirements in a set of models, as opposed to the narrative, textual representation of the Requirements Phase. The representation reflected in the Analysis Document (see section 4.5) should thus be verified with the preceding Requirements Document. This process of cross-checking is actually already present in the main activities described in sections 4.1 and 4.2. The Discovery’s process, applied properly, automatically ensures that all the elements of the Requirements Document have their reflection in appropriate elements of the Analysis Document. However, it is beneficial to cross-check this in order to eliminate any human-generated errors and omissions.

The main technique used during the cross-checking step is a simple walk-through. Here we should bring forward the Interaction Model which is the main link between the requirements and analysis. The suggested procedure is to walk through all the IDs and check them with the requirements Scenarios. We should make sure that every Scenario has one ID for its main course and one ID for every exceptional case. Following that we should consider each ID alone, and assure that all the Scenario clauses have corresponding stimuli with arrows leading from one object (or “outside world”) to another.

After checking the correctness and completeness of the “Scenario part” we should also check the "object part” of the Interaction Model. This activity ensures in turn completeness (but not necessarily correctness) of the Collaboration Model. By walking through the IDs we should once again ensure that each of the stimuli has appropriate client and server objects. We should also ensure that these stimuli are reflected in corresponding responsibilities placed in the server object’s CRC form. Simultaneously we should ensure that appropriate collaborator is placed on a collaborator list in the server object of the stimulus.

The above activities should ensure that our analysis models agree with the Requirements Document and that they don’t lack in any information from the previous phase. These activities would not be necessary if an appropriate CASE tool would be applied. It is important to note that such a tool used for building the Interaction and Collaboration Models should not just allow to automate diagram drawing, and to ensure the correctness of notation. It should also offer the capability to cross-check the two analysis models each with another and with the Requirements Document. The possibilities for such an automated tool are broad, but are outside of scope of this work.

5.5. Summary of Activities - Analysis Document.

The current section summarises the activities described in sections 4.1-4.4. Each of the activities has a resulting deliverable, and all these deliverables form the Analysis Document. As it was already described, activities A1 and A2 should be performed simultaneously. The remaining activities can be performed after constructing the Interaction and Collaboration Models.

5.5.1. Preparation of the Interaction Model (Activity A1).

Techniques
1. Diagrammatic analysis of the requirements’ Scenarios.
2. Object discovery based on the Verb Phrase Vocabulary and Domain Expert’s experience.

Deliverable
1. Interaction Model (A1-1) - one or more IDs for each of the Scenarios from the requirements (for normal and exceptional courses of events).

Notation
1. Interaction Diagram - a diagram representing the interactions between objects during the course of events in a Scenario. Objects are listed along the x axis. Each object is represented by a vertical line expanding from the top to the bottom. Scenario clauses (stimuli) are listed along the y axis. Stimuli are depicted by horizontal arrows coming from an object line to another object line. See Fig. 2.
5.5.2. Preparation of the Collaboration Model (Activity A2).

Techniques
1. Class-Responsibilities-Collaborations analysis.
2. Graph representation of the object collaborations.
3. Object discovery based on the CRC form constraints.

Deliverable
1. Collaboration Graph (A2-1) - a directed graph containing object names and links between them.
2. CRC Forms (A2-2) - a set of forms which expand the Collaboration Graph nodes.

Notation
1. Collaboration Graph - each object is depicted as a rectangle containing the object’s name. Collaborations (bonds) between object are shown as arrows starting from the client objects and ending at the server objects, with numbers showing the binding capacities. See Fig. 3.
2. CRC form - a questionnaire containing the object/class name, a list of its responsibilities and a list of its collaborators. The lists are limited to 9 entries. A separator line is drawn after the 7th entry to indicate the need of splitting the object. See Fig. 4.

5.5.3. Preparation of the Auxiliary Models (Activity A3).

Techniques
1. Entity-relationship analysis.
2. Process analysis.
3. State analysis.

Deliverables
1. Chen Entity-Relationship Diagram (A3-1).
3. Harel State Diagram (A3-3).

Notation
As described by relevant authors (Chen, Jackson, Harel).

5.5.4. Analysis Cross-check (Activity A4).

Techniques
1. Interaction Diagram walk-through.

5.6. Reuse of Analysis.

The reuse of the analysis models should naturally follow the reuse of the requirements models. The most important element here is to elicit the parts of the Analysis Document that come directly from the reused fragments of the Requirements Document. With the models introduced in this work this process should not be difficult and may be done almost automatically. This is due to the fact that both documents are based on the same abstract notions of Actors and Scenarios.

As it was suggested in section 4.6., the reused elements of the Requirements Document should be marked by placing their names in curly brackets. This technique is used just for better visualisation, but we should not under-value such simple notations. With this marking method it is already clear which of the Actors, Scenarios, Nouns or Verbs are to be reused during the analysis phase. Now we just have to find the elements of the Analysis Document that correspond with the above notions. To maintain the elements reused at the requirements level it is very important to have a clear mapping from the requirements to the analysis. Unlike for other methods, the Discovery’s sequence of models and notations offers the means to record this mapping, which we will describe below.
We assume that during the requirements phase there was found a Software Framework in the SF Library with its problem domain close enough to our new system’s problem domain. In section 4.6, we have already described the Requirements Framework level. That level was used to find a proper Framework for our new system. Now we have to reveal the analysis level, i.e. the Analysis Framework. As it was already described, this level of a Software Framework contains simply an appropriate Analysis Document.

We begin the reuse process on the analysis level by mapping all the reused elements found in the Requirements Document of the current project. If the Software Framework we use for the reuse purposes is prepared correctly, this mapping should proceed fairly mechanically. The Analysis Framework should already contain all the Interaction Diagrams that correspond with appropriate Scenarios in the Requirements Framework. Thus we just have to elicit the IDs that correspond with the Scenarios placed in curly brackets in our current Requirements Document. After this activity we should result with an initial pool of IDs directly reused from the previously developed framework.

The above activity should be followed by the construction of the initial Collaboration Model. We should find all the objects that are present in the first pool of reused IDs. By analysing the IDs we should also elicit the responsibilities of the found objects that are suitable for our new system. An important remark here is to take care when discarding of CRC cards and parts of their contents that we think will not be suitable for reuse in the current system. We should not forget that the objects and their responsibilities may perhaps be of some use when analysing the changed or new Scenarios.

There can be taken two approaches to the construction of a partially reused Collaboration Model. The first approach is to start from an empty "workspace" and to add the reused CRC cards successively when we find corresponding objects in the IDs. The CRC cards should initially have an empty list of responsibilities and collaborators. The lists should be filled with these elements copied from the Analysis Framework’s CRC cards also on a successive basis. The second approach to the construction of the Collaboration Model is just opposite to the described above. Here we leave the entire model in our "workspace" and discard all the classes/objects and responsibilities that are not suitable for our current system. The second approach is better for small, incremental developments.

The next activity would be to append the set of Interaction Diagrams with new ones and those that need certain changes in comparison to the previous model. With this activity we return to the normal course of activities as described in sections 4.1. and 4.2. However, during the construction of the Interaction and Collaboration models we have to consider the reused elements from the requirements and elements prospective for reuse from the analysis models. We should not distinguish between new Scenarios and partially changed ones. The construction of new IDs should be done successively according to the list of Scenarios in the Requirements Document. A parallel process of finding new objects and thus amending the Collaboration Model should also proceed as described in previous sections. During this process we should turn our attention to the Noun and Verb Vocabularies. Unlike for ID construction, now we should look for entries not marked as reused. These new entries of the vocabularies may be helpful when discovering new objects and their responsibilities.

II. Case Study.


In the Interaction Model of our case study we will present Interaction Diagrams for only those applications that had their Scenarios presented in the Narrative Model (section 3.12).
<table>
<thead>
<tr>
<th>Diagram C1 - Application reads the Credit Card Details.</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Main track.</strong></td>
<td><strong>Credit Card Reader</strong></td>
</tr>
<tr>
<td>Application displays a CC Swipe Prompt.</td>
<td></td>
</tr>
<tr>
<td>{Start Timer.}</td>
<td></td>
</tr>
<tr>
<td>Passenger swipes a Credit Card.</td>
<td></td>
</tr>
<tr>
<td>Application receives Credit Card Details.</td>
<td></td>
</tr>
<tr>
<td>{Stop Timer.}</td>
<td></td>
</tr>
<tr>
<td>Credit Card Organisation verifies Credit Card Details.</td>
<td></td>
</tr>
<tr>
<td>Application records Credit Card Details.</td>
<td></td>
</tr>
<tr>
<td>Application displays a Swipe Acknowledgement Message.</td>
<td></td>
</tr>
<tr>
<td>&lt;Exit&gt;</td>
<td></td>
</tr>
<tr>
<td>Alternate track 1.</td>
<td><strong>Credit Card Reader</strong></td>
</tr>
<tr>
<td>Application receives a First Time-Out Signal.</td>
<td></td>
</tr>
<tr>
<td>Application sounds a Bell.</td>
<td></td>
</tr>
<tr>
<td>Application displays an Attention Message.</td>
<td></td>
</tr>
<tr>
<td>(same as main track)</td>
<td></td>
</tr>
<tr>
<td>&lt;Exit&gt;</td>
<td></td>
</tr>
<tr>
<td>Alternate track 2.</td>
<td><strong>Credit Card Reader</strong></td>
</tr>
<tr>
<td>Application validates Credit Card Details.</td>
<td></td>
</tr>
<tr>
<td>Application displays a Swipe Error Message.</td>
<td></td>
</tr>
<tr>
<td>&lt;Exit&gt;</td>
<td></td>
</tr>
<tr>
<td>Alternate track 3.</td>
<td><strong>Credit Card Reader</strong></td>
</tr>
<tr>
<td>Application receives a Second Time-Out Signal.</td>
<td></td>
</tr>
<tr>
<td>Application sounds a Bell.</td>
<td></td>
</tr>
<tr>
<td>Application displays a Swipe Time-Out Message.</td>
<td></td>
</tr>
<tr>
<td>&lt;Exit&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Diagram C2 - Credit Card Organisation verifies Credit Card Details.

Main track:
- Application forms a CCA Request.
- Application sends the CCA Request to the Credit Card Organisation.
- {App. starts Timer.}
- Application receives a CCA Reply from the Credit Card Organisation.
- {App. stops Timer.}
- Application records Authorisation Code.
- Application records Credit Limit.
- <Exit>

Alternate track 1.
- CCA Request Message Timer

Diagram C3 - Credit Card Organisation registers a Credit Card Transaction.

Main track:
- Application forms a CCT Request.
- Application sends the CCT Request to the Credit Card
Diagram S1 - Passenger does **Shopping**

Main track.

<table>
<thead>
<tr>
<th>Shopping Options</th>
<th>Shopping Instructions</th>
<th>Shopping Options</th>
<th>Shopping Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application presents the Shopping Instructions.</td>
<td>Application displays Shopping Options.</td>
<td>Passenger selects the Start Option.</td>
<td>{S2} Passenger prepares an Order.</td>
</tr>
<tr>
<td>{C1} Application reads Credit Card Details. =OK=</td>
<td>{C2} Credit Card Organisation verifies the Credit Card.</td>
<td>{S4} Passenger confirms an Order.</td>
<td>{S5} Application places the Order with the Shopping Fulfilment House.</td>
</tr>
<tr>
<td>Application forms a Shopping Card Transaction.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>{C3} Credit Card Organisation registers a Credit Card Transaction.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alternate track 1.</td>
<td>Shopping Options</td>
<td>Shopping Options</td>
<td>Shopping Options</td>
</tr>
<tr>
<td>Passenger selects the Exit Option.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note: the remaining tracks do not add new types of stimuli</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Diagram S2 - Passenger prepares an Order.

Main track:

Application displays a Shopping Item List.
Application displays Order Options.
Passenger selects an Item from the Shopping Item List.
Passenger selects the Add Option.

Application displays Item Selection Options.
Passenger selects the Confirm Option.
Application adds the Item to the Order.

<Repeat>

Alternate track 1.

Passenger selects the Cancel Option.
Application displays Order Cancel Options.
Passenger selects the Confirm Option.

<Exit>

Alternate track 2.

Passenger selects the Accept Option.

<Exit>

Alternate track 3.

Passenger selects the Revise Option.
{S3} Passenger revises the Order.

<Repeat>

Alternate track 4 is similar to track 2.
Diagram S3 - Passenger revises an Order.

Main track. | Order | Order Revision Options | Item Delete Options
---|---|---|---
Application displays the Order. | | | |
Application displays Order Revision Options. | | | |
Passenger selects an Item from the Order List. | | | |
Passenger selects the Delete Item Option. | | | |
Application displays Item Delete Options. | | | |
Passenger selects the Confirm Option. | | | |
Application deletes the Item. | | | |
<Repeat> | | | |

Alternate track 1 | Order | Order Revision Options
---|---|---
Passenger selects the Exit Option. | | | |
<Exit> | | | |

Alternate track 2. | Order | Item Delete Options
---|---|---
Passenger selects the Abort Option. | | | |
<Repeat> | | | |

Diagram S4 - Passenger confirms an Order.

Main track. | Delivery | SFH | Messager | Order | Credit Card
---|---|---|---|---|---
| Order | Address | Request | Conf. Options | Transaction |
Application prompts for Delivery Address. | | | | | |
Passenger inputs the Delivery Address. | | | | | |
Application records the Delivery Address. | | | | | |
Passenger revises the Order. | | | | | |
Application calculates the Total Order Value. | | | | | |
Application displays the Total Order Value. | | | | | |
Application records the Total Order Value. | | | | | |
Application displays Order Confirmation Options. | | | | | |
Passenger selects the Confirm Option. | | | | | |
<Exit> | | | | | |
Alternate track 1 is obvious.

Diagram S5 - Application places an Order with the Shopping Fulfilment House.

Main case.

<table>
<thead>
<tr>
<th>Application forms an SFH Request.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application sends the SFH Request to the Shopping Fulfilment House.</td>
</tr>
<tr>
<td>{Start Timer.}</td>
</tr>
<tr>
<td>Application receives an SFH Reply from the Shopping Fulfilment House.</td>
</tr>
<tr>
<td>{Stop Timer.}</td>
</tr>
<tr>
<td>Application displays an Order Reply Message.</td>
</tr>
</tbody>
</table>

<Exit>

Alternate track 1 similar to those in Credit Card diagrams.
### Chapter 5. Analysis Phase.

#### Diagram P1 - Passenger enters Prize Draw

<table>
<thead>
<tr>
<th>Main track</th>
<th>Prize Draw</th>
<th>Credit Card Reader</th>
<th>Passenger Address</th>
<th>Credit Card Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application presents the Prize Draw Instructions.</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>Application displays the Prize Draw Options.</td>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
<td><img src="image7" alt="Diagram" /></td>
<td><img src="image8" alt="Diagram" /></td>
</tr>
<tr>
<td>Passenger selects the Start Option.</td>
<td><img src="image9" alt="Diagram" /></td>
<td><img src="image10" alt="Diagram" /></td>
<td><img src="image11" alt="Diagram" /></td>
<td><img src="image12" alt="Diagram" /></td>
</tr>
<tr>
<td>{C1} Application reads Credit Card Details.</td>
<td><img src="image13" alt="Diagram" /></td>
<td><img src="image14" alt="Diagram" /></td>
<td><img src="image15" alt="Diagram" /></td>
<td><img src="image16" alt="Diagram" /></td>
</tr>
<tr>
<td>{C2} Application verifies the Credit Card with the Credit Card Organisation.</td>
<td><img src="image17" alt="Diagram" /></td>
<td><img src="image18" alt="Diagram" /></td>
<td><img src="image19" alt="Diagram" /></td>
<td><img src="image20" alt="Diagram" /></td>
</tr>
<tr>
<td>Application displays the Passenger Address Prompt.</td>
<td><img src="image21" alt="Diagram" /></td>
<td><img src="image22" alt="Diagram" /></td>
<td><img src="image23" alt="Diagram" /></td>
<td><img src="image24" alt="Diagram" /></td>
</tr>
<tr>
<td>Passenger inputs the Passenger Address.</td>
<td><img src="image25" alt="Diagram" /></td>
<td><img src="image26" alt="Diagram" /></td>
<td><img src="image27" alt="Diagram" /></td>
<td><img src="image28" alt="Diagram" /></td>
</tr>
<tr>
<td>Application records the Passenger Address.</td>
<td><img src="image29" alt="Diagram" /></td>
<td><img src="image30" alt="Diagram" /></td>
<td><img src="image31" alt="Diagram" /></td>
<td><img src="image32" alt="Diagram" /></td>
</tr>
<tr>
<td>{P2} Application sends the Passenger Address to the Cabin File Server.</td>
<td><img src="image33" alt="Diagram" /></td>
<td><img src="image34" alt="Diagram" /></td>
<td><img src="image35" alt="Diagram" /></td>
<td><img src="image36" alt="Diagram" /></td>
</tr>
<tr>
<td>Application forms a Prize Draw Card Transaction.</td>
<td><img src="image37" alt="Diagram" /></td>
<td><img src="image38" alt="Diagram" /></td>
<td><img src="image39" alt="Diagram" /></td>
<td><img src="image40" alt="Diagram" /></td>
</tr>
<tr>
<td>{C3} Application registers a Credit Card Transaction with the Credit Card Organisation.</td>
<td><img src="image41" alt="Diagram" /></td>
<td><img src="image42" alt="Diagram" /></td>
<td><img src="image43" alt="Diagram" /></td>
<td><img src="image44" alt="Diagram" /></td>
</tr>
</tbody>
</table>

#### Alternate track 1.

<table>
<thead>
<tr>
<th>Prize Draw</th>
<th>Prize Draw Options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Passenger selects the Exit Option.</td>
<td><img src="image45" alt="Diagram" /></td>
</tr>
</tbody>
</table>

#### Diagram P2 - Application sends the Passenger Address to the Cabin File Server.

<table>
<thead>
<tr>
<th>Main track</th>
<th>Passenger Address</th>
<th>Ticket Purchase Request</th>
<th>Timer</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application forms a Ticket Purchase Request.</td>
<td><img src="image46" alt="Diagram" /></td>
<td><img src="image47" alt="Diagram" /></td>
<td><img src="image48" alt="Diagram" /></td>
<td><img src="image49" alt="Diagram" /></td>
</tr>
<tr>
<td>Application sends the Ticket Purchase Request to the Cabin File Server.</td>
<td><img src="image50" alt="Diagram" /></td>
<td><img src="image51" alt="Diagram" /></td>
<td><img src="image52" alt="Diagram" /></td>
<td><img src="image53" alt="Diagram" /></td>
</tr>
<tr>
<td>{Start Timer.}</td>
<td><img src="image54" alt="Diagram" /></td>
<td><img src="image55" alt="Diagram" /></td>
<td><img src="image56" alt="Diagram" /></td>
<td><img src="image57" alt="Diagram" /></td>
</tr>
<tr>
<td>Application receives a Passenger’s Draw Number.</td>
<td><img src="image58" alt="Diagram" /></td>
<td><img src="image59" alt="Diagram" /></td>
<td><img src="image60" alt="Diagram" /></td>
<td><img src="image61" alt="Diagram" /></td>
</tr>
<tr>
<td>{Stop Timer.}</td>
<td><img src="image62" alt="Diagram" /></td>
<td><img src="image63" alt="Diagram" /></td>
<td><img src="image64" alt="Diagram" /></td>
<td><img src="image65" alt="Diagram" /></td>
</tr>
</tbody>
</table>

Page 40
Application displays the Passenger’s Draw Number.

Alternate track 1.

Ticket Purchase Request Timer Messager

Application receives a Long Time-Out Signal.
Application displays a Prize Draw Time-Out Message.

Diagram N1 - Passenger executes News.

Main track.

Application presents a Subject List.
Application displays News Options.

Passenger selects a Subject.

Application receives a Headline List from the Cabin File Server.

Application presents the Headline List.
Passenger selects a Headline.

Application receives a Headline Story from the Cabin File Server.

Application displays the Headline Story.
Passenger selects the Accept Option.

<Repeat>

Alternate track 1.

Passenger selects the Exit Option.

<Exit>

The Collaboration Graph (deliverable A2-1) for the system is shown below. For sake of clarity, collaborations concerning Timer, Credit Card Reader, Credit Card Transaction and Messager were shown in full only for the Faxgram and Shopping Applications’ objects. The diagram contains 45 objects.
The complexity of the diagram on the previous page can be compared to the complexity of PCB layout sheets when designing digital circuits with the use of integrated circuits. Each object with its responsibilities may thus denote a “custom made” circuit. For complex systems certain layering and modularity techniques can be proposed but this is left to future development of Discovery.

Each object from the Collaboration Graph has a corresponding CRC card in the CRC Forms deliverable (A2-2). Only the cards relevant to the applications presented in detail in the Interaction Model were shown. Some of the “...Options” cards were also deleted. All the descriptions of responsibilities denoted by ‘(vocabulary)” can be found in the Verb Phrase Vocabulary.

<table>
<thead>
<tr>
<th>Name: Credit Card Reader</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>1. Read Credit Card - (vocabulary)</td>
<td>1. Timer</td>
</tr>
<tr>
<td>2. Swipe Credit Card - (vocabulary)</td>
<td>2. User Interface</td>
</tr>
<tr>
<td>3. Validate Credit Card Details - (vocabulary)</td>
<td>3. CCA Request</td>
</tr>
<tr>
<td>4. Record Credit Card Details - (vocabulary)</td>
<td>4. Credit Card Transaction</td>
</tr>
<tr>
<td>5. Receive First (Second)Time-Out - (vocabulary)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Messenger</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>1. Display Prompts</td>
<td>-none-</td>
</tr>
<tr>
<td>2. Display Messages</td>
<td></td>
</tr>
<tr>
<td>3. Sound Bell - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>4. Display Total Order Value - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>5. Display Reservation Details - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>6. Display Booking Reference Number - (vocabulary)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Timer</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>1. Start Timer - start to count time-out</td>
<td>1. Credit Card</td>
</tr>
<tr>
<td>2. Stop Timer - end to count time-out</td>
<td>2. CCA Request</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: CCA Request Handler</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>1. Form CCA Request - (vocabulary)</td>
<td>1. User Interface</td>
</tr>
<tr>
<td>2. Send CCA Request - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>3. Receive CCA Reply - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>4. Receive CCA Reply Signal - receive a signal from the Cabin File Server that CCA Reply has arrived</td>
<td></td>
</tr>
<tr>
<td>4. Receive Long Time-Out - (vocabulary)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Credit Card Transaction</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>1. Record Authorisation Code - (vocabulary)</td>
<td>1. Timer</td>
</tr>
<tr>
<td>2. Record (Get) Credit Limit - (vocabulary)</td>
<td>2. User Interface</td>
</tr>
<tr>
<td>3. Record Registration Code - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>4. Form CCT Request - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>5. Receive CCT Reply Signal - receive a signal from the Cabin File Server that CCT Reply has arrived</td>
<td></td>
</tr>
<tr>
<td>6. Receive Long Time-Out Signal - (vocabulary)</td>
<td></td>
</tr>
<tr>
<td>7. Form Credit Card Transaction - (vocabulary)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Shopping</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>Responsibilities</td>
<td></td>
</tr>
<tr>
<td>1. Do Shopping - execute the Shopping Application and make a shopping order</td>
<td>1. Shopping Options</td>
</tr>
<tr>
<td>2. Present Shopping Instructions - (vocabulary)</td>
<td>2. Shopping Item List</td>
</tr>
<tr>
<td></td>
<td>3. Credit Card</td>
</tr>
<tr>
<td></td>
<td>4. Order.</td>
</tr>
<tr>
<td></td>
<td>5. Credit Card Transaction</td>
</tr>
</tbody>
</table>
### Name: Shopping Options

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Display Shopping Options - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>2. Select Start Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>3. Select Exit Option - (vocab.)</td>
<td>none</td>
</tr>
</tbody>
</table>

### Name: Shopping Item List

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Prepare Order - (vocab.)</td>
<td>1. Order Options</td>
</tr>
<tr>
<td>2. Display Shopping Item List - (vocab.)</td>
<td>2. Item Selection Options</td>
</tr>
<tr>
<td>3. Select Item - (vocab.)</td>
<td>3. Order Cancel Options</td>
</tr>
<tr>
<td></td>
<td>4. Order</td>
</tr>
</tbody>
</table>

### Name: Order

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Confirm Order - (vocab.)</td>
<td>1. Order Revision Options</td>
</tr>
<tr>
<td>2. Place Order - (vocab.)</td>
<td>2. Item Delete Options</td>
</tr>
<tr>
<td>3. Add Item - (vocab.)</td>
<td>3. Delivery Address</td>
</tr>
<tr>
<td>4. Revise Order - (vocab.)</td>
<td>4. SFH Request</td>
</tr>
<tr>
<td>5. Display Order - (vocab.)</td>
<td>5. User Interface</td>
</tr>
<tr>
<td>7. Calculate Total Order Value - (vocab.)</td>
<td>7. Credit Card Transaction</td>
</tr>
<tr>
<td></td>
<td>8. Timer</td>
</tr>
</tbody>
</table>

### Name: Order Options

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Display Order Options - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>2. Select Add Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>3. Select Cancel Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>4. Select Accept Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>5. Select Revise Option - (vocab.)</td>
<td>none</td>
</tr>
</tbody>
</table>

### Name: Item Delete Options

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Display Item Delete Options - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>2. Select Confirm Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>3. Select Abort Option - (vocab.)</td>
<td>none</td>
</tr>
</tbody>
</table>

### Name: Delivery Address

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Prompt for Delivery Address - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>2. Input Delivery Address - (vocab.)</td>
<td>none</td>
</tr>
</tbody>
</table>

### Name: SFH Request

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Record Delivery Address - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>2. Form SFH Request - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>3. Send SFH Request - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>4. Receive SFH Reply - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>5. Receive Long Time-Out Signal - (vocab.)</td>
<td>none</td>
</tr>
</tbody>
</table>

### Name: Order Confirmation Options

<table>
<thead>
<tr>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Display Order Confirmation Options - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>2. Select Confirm Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>3. Select Cancel Option - (vocab.)</td>
<td>none</td>
</tr>
<tr>
<td>Name: Prize Draw</td>
<td>Responsibilities</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>1. Enter Prize Draw - execute the prize draw application and participate in a prize draw</td>
<td>2. Present Prize Draw Instructions - (vocab.)</td>
</tr>
<tr>
<td>2. Present Prize Draw Instructions - (vocab.)</td>
<td>3. Passenger Address</td>
</tr>
<tr>
<td>3. Passenger Address</td>
<td>4. Credit Card Transaction</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Prize Draw Options</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Display Prize Draw Options - (vocab.)</td>
<td>2. Select Start Option - (vocab.)</td>
<td>-none-</td>
</tr>
<tr>
<td>2. Select Start Option - (vocab.)</td>
<td>3. Select Exit Option - (vocab.)</td>
<td>-none-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Passenger Address</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Input Passenger Address - (vocab.)</td>
<td>2. Record Passenger Address - (vocab.)</td>
<td>1. Ticket Purchase Request</td>
</tr>
<tr>
<td>2. Record Passenger Address - (vocab.)</td>
<td>3. Send Passenger Address - (vocab.)</td>
<td>2. Timer</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Ticket Purchase Request</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Form Ticket Purchase Request - (vocab.)</td>
<td>2. Send Ticket Purchase Request - (vocab.)</td>
<td>-none-</td>
</tr>
<tr>
<td>2. Send Ticket Purchase Request - (vocab.)</td>
<td>3. Receive Ticket Purchase Request - (vocab.)</td>
<td>-none-</td>
</tr>
<tr>
<td>3. Receive Ticket Purchase Request - (vocab.)</td>
<td>4. Receive Ticket Purchase Request Signal - receive a signal from the Cabin File Server informing that the Ticket Purch. Req. has arrived</td>
<td></td>
</tr>
<tr>
<td>4. Receive Ticket Purchase Request Signal - receive a signal from the Cabin File Server informing that the Ticket Purch. Req. has arrived</td>
<td>5. Receive Long Time-Out Signal - (vocab.)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: News</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Read News - execute news application and read headline news</td>
<td>2. Present Subject List - (vocab.)</td>
<td>1. News Options</td>
</tr>
<tr>
<td>2. Present Subject List - (vocab.)</td>
<td>3. Select Subject - (vocab.)</td>
<td>2. Headline List</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: Headline List</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>2. Present Headline List - (vocab.)</td>
<td>3. Select Headline - (vocab.)</td>
<td>-none-</td>
</tr>
<tr>
<td>3. Select Headline - (vocab.)</td>
<td>4. Receive Headline Story - (vocab.)</td>
<td>-none-</td>
</tr>
<tr>
<td>4. Receive Headline Story - (vocab.)</td>
<td>5. Display Headline Story - (vocab.)</td>
<td>-none-</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: News Options</th>
<th>Responsibilities</th>
<th>Collaborators</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Display News Options - (vocab.)</td>
<td>2. Select Accept Option - (vocab.)</td>
<td>-none-</td>
</tr>
<tr>
<td>2. Select Accept Option - (vocab.)</td>
<td>3. Select Exit Option - (vocab.)</td>
<td>-none-</td>
</tr>
</tbody>
</table>
6. Design Phase.

I. Notation and Activities.

In the literature on object-oriented technology there seems to exist an uncertainty on when to finish ‘analysing’ the system, and when to start ‘designing’ it (see e.g. [Jacobson, 1994], p. 145). This is caused by a rather vague boundary between the two stages of software development for object-oriented technology. Especially the first generation methods do not offer clear guidelines. For methods that concentrate on building static ‘object models’ we can thus hardly distinguish between analysis and design (e.g. Coad/Yourdon’s method). In Discovery, however, we have a clear distinction and clear definition of the boundary. This is because the models of the Design Stage are different than the models of the Analysis Stage. This is in accordance with the overall philosophy of Discovery, which says that the transitions between phases should not necessarily be ‘seamless’, but instead there should be a clear definition of the transition process.

In the first of the two Phases of the design stage we will concentrate on re-organising, and modularising the system. This includes building inheritance and containment hierarchies and joining these hierarchies with the existing ‘software components libraries’. As we can see, we leave the detailed specification of objects/classes to the last Implementation Stage, which is in fact detailed design. With such an approach, we can perform reuse of components easier than if we would have already made more detailed specification of classes. With only the responsibilities of objects/classes defined at this stage we are more flexible in modifying the analysis models with the purpose of reusing existing components.

6.1. Object/Class Model.

The Design Phase is based on the Object/Class Model, which is derived directly from the Collaboration Model of the Analysis Phase. Only in this model we start to develop the important two hierarchies of inheritance and containment. This model is also a basis for the equally important reuse step. In this section we will propose a notation for the Object/Class model, and then we will give guidelines for constructing it from the Collaboration Model.

The main problem with defining the notation for the Object/Class Model is that most of the existing methods have different ways of presenting it. Some notations distinguish between a class and its objects, some do not. There is a broad spectrum of icons representing a class. For instance Booch proposes cloud shaped icons, whereas Coad and Yourdon use rectangles with rounded corners, and Jacobson uses circles. Even more exotic notation is used in the Moses method, where classes are denoted by tablets, which resemble those given by God, inscribed with the ten commandments\(^2\). However, most methods denote classes with differently ‘decorated’ rectangles. For Discovery we will not try to propose a separate notation. In this work we will use notation taken from Moses with slight modifications. The main reason is its clarity and good visual impact. However, the basic class icon will be changed from the tablet-like into an ordinary rectangle, like in Fig. 5.

Generally, the Object/Class Model is presented in a form of a graph similar to that seen already in the Collaboration Model. However, this time we have to use precise terminology of the object-oriented paradigm and ‘objects’ of the Collaboration Graph will have to be transformed into ‘classes’ of the Object/Class Model. The former name had to be used throughout the Analysis Phase in order not to confuse the Domain Expert (i.e. the user) and not to introduce an abstract notion of a class.

\(^2\) Even the CASE tool developed for Moses (i.e. Abraham) has a name taken from the Bible’s Book of Exodus.
As it was already mentioned, a single class will be denoted by a rectangle with its name written inside it (see Fig. 5). In the inheritance hierarchy, and in the libraries there also occur abstract (or deferred) classes. In order not to go into much implementation details, on the design level we may assume that such classes have abstract behaviour which is specified separately for different child classes, and that such classes may not be instantiated in the system. To follow the Moses notation, an abstract class will be denoted by a dashed rectangle. Similarly, classes that are taken from the library are denoted by rectangles with thick borders. This should represent a “dosed” nature of a library class which prevents from direct modifications, but leaves possibility to modify through inheritance. Abstract library classes have additionally dashed border. Figure 5 summarises the proposed notation for classes.

![Figure 5. Classes - notation.](image)

To complete our Object/Class Model on the design level we need to denote relationships between classes. There are three types of relationships that the O/C Model will contain: inheritance, association and aggregation. Inheritance is denoted by a thick arrow leading from the child class to its parent. Multiple inheritance is represented by two or more arrows coming from the child to its ancestors. The summary of notation for inheritance relationships is presented on Fig. 6.

![Figure 6. Inheritance relationships.](image)

An association relationship is denoted by a normal arrow with an open circle placed in the middle. Aggregation, which is a special case of association is in turn denoted by an arrow with a filled circle. It has to be noted that the association relationship can be bi-directional, but aggregation for
obvious reasons can be only unidirectional. Both above relationships can have cardinality/existence constraints specified. The notation is a pair of numbers representing the minimum and maximum number of participating classes at either end of connection. This notation allows to denote also optional relationships, for which the lower bound is equal to zero. Summary of notation for association and aggregation is given on Fig. 7.

![Diagram of associations and aggregations](image)

**Figure 7. Association and aggregation.**

When we established the notation for the Object/Class Model we have to specify the way in which to derive it from the Collaboration Graph. The main techniques used here are restructuring and simplification of communication patterns. To perform this we should first identify subsystems within the Collaboration Model. The general criterion is that the subsystems should have maximum cohesion included in them, and minimal coupling between them [Simons, 1994]. To assure maximum cohesion we should use binding capacities shown in the Collaboration Graph, and isolate (encapsulate) those groups of classes which have relatively large number of mutual bonds and where the sum of binding capacities is high. When we identify such groups we should define a “subsystem class” which would allow to reorganise and simplify mutual bindings. This approach was suggested in [Wirfs-Brock, 1990] and by slightly modifying notation we can represent it in our current model (see Fig. 8). As we can see from Fig. 8, the bonds were changed into associations. The semantics of association is similar to the semantics of a bond between classes in the Collaboration Model. Association denotes a client-server relationship between two classes. It does not distinguish however between static references and dynamic passing of them during program execution. This distinction has to be established in the Visibility Model of the Implementation Phase. Thus, in the example on Fig. 8 we have decided to brake mutual bonds between subsystem’s classes which results in the fact that e.g. Purchaser does not know anything about the Vendor.

Additionally to defining subsystems with mutual communication between objects, we should also define aggregation subsystems and inheritance hierarchies. Aggregation can be also discovered by looking at the Collaboration Graph. A uni-directional bond between two objects can certainly be transformed directly into a uni-directional association. For that association to become an aggregation certain criteria have to be met. First criterion can be that of common-sense semantics of a relationship.
between objects. It is obvious that an association between a car, its engine and wheels can be treated as an aggregation where a car aggregates an engine and wheels. Some less obvious aggregation relationships can be discovered by examining the structure of the Collaboration Graph, and determining which objects may have their lifetime bound and equal to the lifetime of some other object requesting services from it. This approach needs some dose of experience, as these links are normally not directly apparent.

![Diagram](image)

**Figure 8. Simplification of communication.**

Inheritance relationships can be determined by looking at common responsibilities of different objects. This can be done by looking at the CRC card level of the Collaboration Model. Venn diagrams, as proposed in [Wirfs-Brock, 1990] can be used to factor out sets of common responsibilities. The inheritance relationships do not usually have equivalent in the Collaboration Graph. They have to be discovered when constructing the Object/Class Model. The inheritance relationships can be included in the overall Object/Class diagram. However, for larger systems it is much more convenient to produce a separate inheritance diagram. This diagram would use only the notation as in Fig. 6, and possible lower levels of inheritance can (and should) be included.

### 6.2. Reuse of Software Components.

An important element of the Design Stage is the issue of reuse. In fact it is perhaps one of the key issues in the whole software development cycle. Some authors stress the importance of so called "software components" as a possible solution to the "software crisis" (see e.g. [McIlroy, 1976]). Constant research is done in this direction but we are yet far from satisfactory solutions.

An approach of Discovery towards reuse of classes (or software components) is to maintain plasticity of the models before the component reuse step. This plasticity allows us to change our designs easily without invalidating the whole development process, and changing the system architecture dramatically. This plasticity is achieved, as the reader may have already noticed, by basing our initial analysis models around the behaviour of objects rather than the data they contain.

Until the design stage, each of the objects (or rather classes) was basically described by its name and a list of responsibilities which denote the behaviour of this object. This description is relatively general and does not assume any specific implementation. Thus it is not too difficult to substitute a class from our current Object/Class Model with a class from the “components library”. To systematise this approach we will propose a set of steps which can be applied during the reuse part of the Design Phase. All the steps include certain operations on a library which we will call the Components Library. Before defining these steps we will describe assumptions about the library.
6.2.1. Components Library.

We propose the following assumptions about the Components Library:

1. We assume that the library contains a number of class inheritance hierarchies as described in section 6.1. These hierarchies are usually built gradually and amended during each consecutive project developed in a software house.

2. Names are carefully assigned to components. The names should be consistent with the overall company strategy, and consistent with a “central vocabulary”. This vocabulary should be specified and maintained for all the problem domains that a company operates in. A format for such a vocabulary was proposed in section 4.3. Each name may also have several aliases assigned to it in order to reduce effort when seeking for components.

3. Each class in the Components Library is fully specified in accordance with the major development method used in the company. This work proposes such specification and bases it in the Discovery’s Implementation Phase.

4. Each class should additionally have a list of general responsibilities which should be named and described according to the format given in sections 5.1 and 5.2.

5. The company may apply a general strategy of dividing the Components Library into two separate libraries - a Generalised Components (GC) Library and Components Proposed for Generalisation (CPG) Library. This approach was proposed in [Henderson-Sellers, 1993], and will be described in detail in the next section.

6.2.2. Reuse Steps.

1. **Find a similar component.** In order to find a component that is similar to the currently considered class we can seek in the library for a name which is close to that of our class. We can also look for similarly described responsibilities. Thus the company's database containing the Components Library should be able to handle these two types of queries. An alternative query should seek for aliases similar to our class's name.

2. **Use the component directly.** If the component we have found has a superset of responsibilities of our class, we can reuse the component directly. Sometimes, only some of the responsibilities overlap. In such case we can try to rephrase or modify the responsibilities of the class. If this is possible and the resulting class has a subset of responsibilities of the found component we can reuse this component directly.

3. **Inherit from the component.** If we have found a component (or several components) that include only a subset of the behaviour proposed for our new class, then we should try to inherit from it (them). Only the responsibilities not fulfilled by the parent component(s) should be specified in the Implementation Phase. Specifications for other responsibilities can be reused.

4. **Generalise an inherited class.** After we have specified a class we can try to write it more generally and include in the Components Library. If a class was inherited from a component, it will eventually form the next level of inheritance in the library hierarchy.

5. **Move the class to the Components Proposed for Generalisation Library.** If we think that our class has a potential of becoming a component, we should move it to the CPG Library (see: [Henderson-Sellers, 1993]). We do not yet generalise this class. Thus we do not add to the effort taken during the development of the current system. Generalisation is deferred to the next project as described in the next step.

6. **Generalise a class from the CPG Library.** If we did not find any classes in the main Generalised Components Library, we can try and seek for them in the CPG Library. If we find a suitable object, this means that an object proposed to be potentially general in a previous project indeed has this feature. Thus, now we may try to specify this component generally and use it as if it was found in step 2. Such an approach spreads the costs of additional generalisation effort into several projects. In fact this solution can be treated as more fair, as the client pays only for generalisation that is needed for his project.

6.3.1. Preparation of the Object/Class Model (Activity D1).

Techniques
1. Simplification of communication patterns.
2. Generalisation.
3. Aggregation.

Deliverable
1. Object/Class Model (D1-1) - a graph specifying classes and relationships between them.

Notation
1. Classes denoted by rectangles as on Fig. 5. Inheritance relationships denoted by thick arrows as on Fig. 6. Association relationships (including aggregations) denoted by thin arrows as on Fig. 7.

6.3.1. Reuse of Software Components (Activity D2).

Techniques
1. Selection of software components.
2. Direct reuse or inheritance from software components.

Deliverable
Contributes to Object/Class Model (D1-1).

II. Case Study.

We will illustrate the notation used in part I on a fragment of the IFCS system. To show the construction of inheritance hierarchy we will use an example of two similar classes - Shopping class and Prize Draw class. If we isolate only the parts of the system concerning these two classes we will result with a diagram shown below.
We will now assume that we have two components present in the Components Library. The first element is called Application, and the second - OnScreenOptions. Both of these elements will serve us as parents for our existing classes.

The Application class is responsible for executing itself, and for displaying on-screen instructions. In our subsystem shown above we have two classes with similar behaviour. The Shopping class is responsible for ‘doing shopping’, and the Prize Draw class is responsible for “entering prize draw”. Both of these responsibilities can be treated as specialisation’s of “execute application”. Similarly, both classes should display relevant instructions on the screen. Moreover, the two classes use the Credit Card and Credit Card Transaction in the same manner to pay for using appropriate applications. Thus, in our inheritance hierarchy we will introduce a new class - PayableApplication. This class will be additionally responsible for handling credit card operations.

The OnScreenOptions class is capable of displaying several on-screen buttons, and determining if the buttons are pressed. For our system it will be useful to create a new child class - ApplicationOptions. This class would be responsible for displaying two options - Start and Exit. It would also display a prompt to press one of the buttons, and determine which of the buttons were pressed. Then, the two ”xxOptions” classes existing in the graph on the previous page would inherit from this class and specialise their prompts. In result we will have the following structure of our fragment of the system.
Chapter 7. Implementation Phase.

1. Notation and Activities.

In the final Implementation Phase we complete the Object/Class Model by specifying several supporting models which are detailed in this section. The Implementation Phase's models maintain the overall philosophy of Discovery which is to keep the level of complexity revealed to a developer in a single model in reasonable boundaries. Thus, the models presented in this section reveal the lower levels of abstraction where again the number of elements revealed makes them still comprehensible to the developer.

Implementation Phase of Discovery can be treated as a detailed design phase. However, the hierarchy of abstractions can finally lead us to code specification of individual classes.

7.1. Feature Layer.

In the Feature Layer we specify all the features of the classes contained in the Object/Class Model. The features (this term is taken from Eiffel [Meyer, 1988]) can be divided into properties and operations. Properties describe the current state of an object (instance of a class). Operations define the behaviour of the class, and allow to change its state or to offer services. Thus, generally the term “properties” denotes all the data attributes of a class, as well as all the functions that return the current state of a class’s instance. Operations denote all the services offered by the class. This includes most of the methods (functions and procedures) offered by the class (except for those functions that describe properties).

This division into properties and operations may seem a little artificial. Why not simply divide the features into attributes and methods? However, at the Feature Level we do not want to specify issues that are to be determined in the Method Layer (see next section). It is only there where we decide whether a particular property should be implemented as an attribute or a function (method).

In the Feature Layer we will also distinguish between hidden and exported features. This means that we will show here which of the encapsulated features are to be visible to other classes (public) and which are to be hidden (private).

The notation used in the Feature Layer is consistent with that of the general Object/Class Model. Again it is adapted from [Henderson-Sellers, 1994], i.e. from the Moses method. A basic class icon will be amended with a second (‘feature’) part which will contain a list of all the class’s features. All the exported (public) features will be placed in a box that protrudes from the icon. The hidden (or private) features will be placed wholly inside the icon. This notation is depicted on Fig. 9.

The lists included in the feature part of an icon include properties which are denoted by their name followed by a colon and type of value returned (or stored). Operations should have also their name listed, and a list of parameters included in a pair of brackets. A simplified view may include just the names of the listed features without revealing their types and parameters. An example of notation is given on Fig. 10. However, we should look here at the target language and preferably use its notation.

The Method Layer is a further expansion of the Feature Layer. Each of the methods from this higher layer is specified here in detail. However, before this specification can be made we have to decide which of the properties are to be implemented as methods. After this decision is made, each of the methods should be specified formally.

Formal specification of a method is based on a notion of “contracts”. This notion was introduced by Meyer in [Meyer, 1988]. The idea is very interesting as it introduces formal methods to object-oriented technology in a useful way. A contract is a metaphor which denotes mutual obligations between a client and a server of some service (method). When a method is invoked, these two objects fulfil a certain agreement which is specified in the method specification.
To denote a contract, two formal conditions are specified - the precondition and the postcondition. A precondition defines the client’s obligation for the contract. It sets constraints on the parameters supplied by the client. If the parameters do not fulfill the precondition - it means that the client has broken the contract. In such case the server is not obliged to do nothing and may return an information about contract violation (an exception). The postcondition denotes the server’s obligation. It sets constraints on the results of method invocation. The constraints are mainly set on the resulting state of the server object. Thus the method contracts have to be defined in accordance with the class properties defined in the Feature Layer. If the postcondition is violated - the server has broken the contract. It is now its duty to repair the damage and return the object’s state to proper condition.

In the Method Layer we will thus define contracts for each of the operations from the Feature Layer and for all those properties that are to be implemented as functions. Each of these methods will have a method sheet, which will contain the elements of the contract. An example of a method sheet is given on Fig. 11. The top part of the sheet is reserved for the method’s full signature. The signature contains the method’s name, its parameters, and/or the type of function’s result. It can be written in the target language as it was suggested in the previous section. The second part contains the precondition. Usually this precondition should be written as a sequence of conditions set on the values of parameters at the time of invocation. Similarly, the third part of the sheet contains the postcondition, which should generally be a sequence of conditions set on the resulting state of the object, or the returned value.

<table>
<thead>
<tr>
<th>Account</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance: POUND</td>
</tr>
<tr>
<td>OverdraftLimit: POUND</td>
</tr>
<tr>
<td>Withdraw(amount: POUND)</td>
</tr>
<tr>
<td>Deposit(amount: POUND)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>invariant</th>
</tr>
</thead>
<tbody>
<tr>
<td>Balance ≥ OverdraftLimit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Withdraw(amount: POUND)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>precondition</strong></td>
</tr>
<tr>
<td>amount &gt; 0</td>
</tr>
<tr>
<td>Balance - amount ≥ OverdraftLimit</td>
</tr>
<tr>
<td><strong>postcondition</strong></td>
</tr>
<tr>
<td>Balance = old Balance - amount</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Deposit(amount: POUND)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>precondition</strong></td>
</tr>
<tr>
<td>amount &gt; 0</td>
</tr>
<tr>
<td><strong>postcondition</strong></td>
</tr>
<tr>
<td>Balance = old Balance + amount</td>
</tr>
</tbody>
</table>

**Figure 11. Method sheets, invariants - notation.**

Additionally to defining pre- and postconditions of all the class’s methods, we can also define a general constraint set on the state of all the class’s instances (objects). This constraint is called an invariant. An invariant can be treated as an additional part added to all the class’s postconditions, and should be placed at the bottom of the class icon (see Fig. 11).

At the method layer we may also choose to define the algorithmic functionality of a method. However, we will not give any guidelines here and we will leave it to future development of Discovery.
### 7.3. Behaviour Model.

With the Behaviour Model we try to establish the sequence of method invocations in the system. A Method Trace Diagram can be created for each system operation. This resembles what we did in the Interaction Model, but here it is performed on a lower level of abstraction. However, a link between Interaction Diagrams and Method Trace Diagrams should be maintained. This link can be expressed by a similar notation for method traces, however there can be also given an alternative notation which is presented on Fig. 12. In both of these notations the main purpose is to show the time sequence of method invocations for a single operation.

![Diagram]

**First version of notation (adapted from [Henderson-Sellers, 1994])**

1. LastPeriodInterest: POUND
2. Balance: POUND
3. DayOfLastInterest: DAY

**Second version of notation**

1. Balance: POUND
2. InterestRate(currentBalance: POUND): INTEREST_RATE
3. DayOfLastInterest: DAY

*Figure 12. Method Trace Diagram - notations.*

### 7.4. Visibility Model.

The Visibility Model shows us the nature of relationships between objects in terms of their dependency in time. Two objects may be constantly bound and a relationship may be of a permanent nature. However, in many cases references are maintained only for a short period in the lifetime of two objects (dynamic reference). Such discrimination is denoted by the Visibility Model.
Chapter 7. Implementation Phase.

The idea of visibility analysis comes from Fusion ([Coleman, 1994]). It will be treated here as an extension of the overall Object/Class Model. The basic discrimination between permanent and dynamic references can be denoted in the O/C Model by dashing those relationships that are of dynamic nature. However, often it is beneficial to know also the timing sequence of dynamic object referencing. It can be proposed to incorporate this timing sequence into the Method Trace Diagrams. However, this will be left to future development of Discovery.


7.5.1. Definition of the Feature Layer (Activity I1).

Techniques
1. Elicitation of properties and operations.
Deliverable
1. Feature extension of the Object/Class Model (I1-1) - addition of features to the class icons.
Notation
1. Two lists of features each placed in a box. The public features’ box protrudes the class icon. The private features’ box is contained wholly in the class icon. Properties separated from operations by a dashed line. See Fig. 9 and 10.

7.5.2. Definition of the Method Layer (Activity I2).

Techniques
1. Formal description of client-server contracts.
Deliverable
1. List of Method Sheets (I2-1) - a method sheet for each method elicited in the Feature Layer.
Notation
1. Method sheet - a form containing the method signature, a precondition and a postcondition. See Fig. 11.

7.5.3. Construction of the Behaviour Model (Activity I3).

Techniques
1. Analysis of method invocations.
Deliverables
1. Behaviour Model (I3-1)- a set of Method Trace Diagrams that describe time sequences of method invocations.
Notation
1. Method Trace Diagram - a diagram with arrows leading between objects in a time sequence. Each arrow is denoted by a consecutive number and a method signature or name. Two versions of notation are presented on Fig. 12.

7.5.4. Construction of the Visibility Model (Activity I4).

Techniques
1. Analysis of referencing dynamics.
Deliverables
Contributes to Object/Class Model (D1-1) and Behaviour Model (I3-1).
II. Case Study.

Here we will continue with the fragment of the system presented in part II of the previous chapter. Below we will present two hierarchies including Application with PayableApplication, and OnScreenOptions with ApplicationOptions. The diagrams on this page show the Feature Layer for this part of the system. Deferred (or abstract) methods are placed in square brackets. Note also that only features added or changed in a child class are shown.

The classes that inherit from PaymentApplication (Shopping and PrizeDraw in our example) should define their versions of ‘ActionBeforeSwipe’, ‘ActionsAfterSwipe’ and ‘SetInstructionPages’. Classes inheriting from Application Options (i.e. ShoppingOptions and PrizeDrawOptions) should define ‘DisplayPrompt’ and ‘SetButtons’.

The diagram above shows us just the static structure of each class. To clarify the semantics of the listed features we will now draw a part of the Behaviour Model concerning the classes shown.
The diagrams below constitute Method Trace Diagrams for invocation of the ‘Create’ and ‘Execute’ methods of one of the PayableApplication’s children. The second diagram can be compared with diagrams S1 and P1 of the Interaction Model.

<table>
<thead>
<tr>
<th>Create</th>
<th>PayableApplication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td></td>
</tr>
<tr>
<td>SetInstructionPages</td>
<td></td>
</tr>
<tr>
<td>SetTransactionSum</td>
<td></td>
</tr>
<tr>
<td>(0: POUND)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Execute</th>
<th>PayableApplication</th>
<th>Application Options</th>
<th>Credit Card Reader</th>
<th>Credit Card Transaction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execute</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisplayInstructions</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisplayAndChoose:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BUTTON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisplayPrompt</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DisplayButtons</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ButtonPressed: BUTTON</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HideButtons</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destroy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActionsBeforeSwipe:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECISION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ReadCreditCardDetails:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC_RESULT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifyCreditCard:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CC_RESULT</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ActionsAfterSwipe:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECISION</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Create</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SetSum(TransactionSum:</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>POUND)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Register</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destroy</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

It is assumed that CreditCardReader and CreditCardTransaction have appropriate methods defined as used in the diagrams.
8. Summary and Conclusions.

At the present stage of Discovery’s development, the current work was to constitute the first step towards precise definition of the method. In fact, with this work we have performed a first full round of the specification process. It was not the goal of this work to present a final version of Discovery, as this was certainly out of scope of this thesis. However, certain important results were obtained and important new elements have been added to the method. The achievement of this work is that some of the solutions proposed here were already included in the first draft of the method (see: [Simons, 1996]).

By conducting the In-Flight Cabin Services case study we wanted to show applicability of the method in a real industry project. This task was certainly fulfilled successfully. The case study project resulted in a 50 page document which contains the final deliverable partially described in this work. The subsequent system contained 45 classes which allows to classify it as a medium scale one. Although this, the techniques used in Discovery allowed to reach the Design Phase with only one person involved in the development process. However, no detailed design was made, and this part of the method still waits for its testing in the industry.

The specification of the method presented in this work concentrates on its main path. Optional models and deliverables were not specified in such detail as other, obligatory parts. Also the Task Model needs a more precise definition. In fact it was noted throughout the case study that a task priority matrix could be a very useful way of establishing which of the system tasks should be developed first. The need to introduce such a technique was recognised during the development process of the IFCS system. With certain financial, timing or versioning constraints set on the project, it is of great benefit to prioritise system tasks before more detailed development is started. It can be noted that the time constraints of the present work were a good simulation of such real life constraints.

The method as defined in previous studies ([Simons, 1994], [Simons, 1996]) was composed of several techniques and have included a general propositions of notations. With this work we introduce certain new elements to the method and we propose a precise development path. The techniques used previously are still maintained, and a general ‘philosophy’ of the method is thus sustained. The development process is based on the process specified earlier, but precise boundaries of the intermediate models and deliverables are given.

A wholly new technique of vocabulary building has been introduced to the method. The technique, which was already incorporated to the first draft of Discovery, constitutes a link between the unstructured user requirements and precise narrative modelling of the system requirements. This link allows the user and the developer to communicate in the same language of the considered problem domain. With the introduction of the vocabulary, the narrative modelling was constrained to use only the phrases contained in it. This added a new level of precision to the requirements specificiation, and reduced the level of ambiguity.

With this work we also propose a notation for the task scenario structuring. However, the final version of notation still has to be determined. The previous version of script notation was based on the SVDPI grammar, which is still retained in this work. However, it was also proposed in [Simons, 1996] to introduce a TCA extension to this grammar, which allows for ‘if” conditions and “while” loops. This work proposes ‘alternate tracks’ notation, which can be argued as a better way of structuring scripts, which is closer to “story telling” with different endings rather than to structural programming.

Another important element of Discovery changed as a result of this work and its case study is the Collaboration Graph. It was determined that the objects on this graph should be linked with just a single arrow denoting all the collaboration between them, instead of introducing separate link for each of the responsibilities. This allowed for a better level of abstraction, especially important for the systems of size comparable to the IFCS system. It can be also suggested that certain techniques which would divide the developed system into modules or subsystems could be introduced to the method.
References.

Books and Articles.


Coad, P. and Yourdon, E. (1990a), OOA - Object-Oriented Analysis, 2d ed., Prentice-Hall.

Coad, P. and Yourdon, E. (1990b), OOD - Object-Oriented Design, Prentice Hall.


Owen, D. (1996), Software blamed for rocket failure, Financial Times, 6 June


Simons, A. (1994), Object Oriented Analysis and Design (lecture notes), Department of Computer Science, University of Sheffield.


Wirfs-Brock, R., Wilkerson, B. and Wiener, L. (1990), Designing Object-Oriented Software, Prentice Hall.

World Wide Web References.

Note: the WWW locations given below are not guaranteed to be valid in the future.

http://www.is.cs.utwente.nl:8080/dmrg/OODOC/oodoc/oo.html

Fusion Home Page (1996),
http://www.hpl.hp.com/fusion/

Open Project Home Page (1996),

Shlaer/Mellor Method Home Page (1996),
http://www.projtech.com/smmethod/smmethod.html

Simons, A. (1996), Discovery Home Page,
http://www.dcs.shef.ac.uk/~ajhs/discovery.html

Rational Software (Booch’s method) Home Page (1996),
http://www.rational.com/

UML (Unified Modelling Language) Home Page (1996),
http://www.rational.com/ot/uml.html
### Appendix. Project Diary.

<table>
<thead>
<tr>
<th>Date</th>
<th>Event</th>
</tr>
</thead>
<tbody>
<tr>
<td>June - 15 July 1996</td>
<td>Background reading, gathering of techniques and notations.</td>
</tr>
<tr>
<td>15 July 1996</td>
<td>Introductory meeting with preliminary concepts about Discovery.</td>
</tr>
<tr>
<td>7 August 1996</td>
<td>Final version of requirements received from EASAMS.</td>
</tr>
<tr>
<td>5 September 1996</td>
<td>Major meeting with discussion about Discovery techniques.</td>
</tr>
<tr>
<td>10 September 1996</td>
<td>Requirements and Analysis documents sent to EASAMS.</td>
</tr>
</tbody>
</table>