Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

The Journal of Systems and Software 68 (2003) 183–198

The Journal of Systems and Software

www.elsevier.com/locate/jss

A conceptual model completely independent of the implementation paradigm

Oscar Dieste [a,*], Marcela Genero [b,1], Natalia Juristo [c,2], José L. Maté [c,2], Ana M. Moreno [c,2]

[a] *Departamento de Electrónica y Sistemas, Escuela Politécnica Superior, Universidad Alfonso X el Sabio, 28691-Villanueva de la Cañada, Madrid, Spain*

[b] *Departamento de Informatica, Escuela Superior de Informatica, Universidad de Castilla-La Mancha, Paseo de la Universidad, 4, 13071 Ciudad Real, Spain*

[c] *Departamento de Lenguajes y Sistemas, Informáticos e Ingeniería del Software, Facultad de Informática, Universidad Politécnica de Madrid, 28660-Boadilla del Monte, Madrid, Spain*

Received 23 December 2002; accepted 27 December 2002

Abstract

Several authors have pointed out that current conceptual models have two main shortcomings. First, they are clearly oriented to a specific development paradigm (structured, objects, etc.). Second, once the conceptual models have been obtained, it is really difficult to switch to another development paradigm, because the model orientation to a specific development approach. This fact induces problems during development, since practitioners are encouraged to think in terms of a solution before the problem at hand is well understood, thus anticipating perhaps bad design decisions.

An appropriate analysis task requires models that are independent of any implementation issues. In concrete, models should support developers to understand the problem and its constraints before any solution is identified. This paper proposes such an alternative approach to conceptual modelling, called "problem-oriented analysis method".

Keywords: Conceptual modelling; Generic conceptual model; Development orientation

1. Introduction

Requirements engineering (RE) activity is composed by four iterative tasks: elicitation, analysis, documentation and validation (SWEBOK, 2000). Of these tasks, analysis is one of the most critical, due to the huge importance of its goals: (1) understand the problem to be solved; (2) develop conceptual models (CMs), which represent the problem understanding; and (3) define the features of an implementation-independent solution to the problem in question, that is, identify the requirements to be satisfied by the future software system. CMs play a central role during analysis, since they make possible to

- make real-world concepts and relationships tangible (Motschnig-Pitrik, 1993);
- record parts of reality that are important for performing the task in question and downgrade other elements that are insignificant (Borgida, 1991);
- support communication among the various "stakeholders" (customers, users, developers, testers, etc.) (Mylopoulos et al., 1997);
- detect missing information, errors or misinterpretations, before going ahead with system construction (Schreiber et al., 1999).

Conceptual modelling is gaining in importance as software systems become more complex and the problem

[*] Corresponding author. Present address: Departamento de Sistemas Informaticos y Programacion, Facultad de Informatica, Universidad Complutense de Madrid, Ciudad Universitaria S/N, 28040 Madrid, Spain. Tel.: +34-91-394-75-46.

E-mail addresses: odiestet@fdi.ucm.es (O. Dieste), marcela.genero@uclm.es (M. Genero), natalia@fi.upm.es (N. Juristo), jlmate@fi.upm.es (J.L. Maté), ammoreno@fi.upm.es (A.M. Moreno).

[1] Tel.: +34-926-29-54-85x3740.

[2] Tel.: +34-91-336-6922/6921/6929.

domain moves further away from knowledge familiar to developers. In complex domains, to understand the user need becomes more difficult and, therefore, conceptual modelling grows to be crucial. Several researchers claim that proper conceptual modelling is crucial since it helps to represent the problem to be solved in the user domain (McGregor and Korson, 1990; Bonfatti and Monari, 1994; Høydalsvik and Sindre, 1993).

Nevertheless, several authors have argued that the CMs used nowadays are oriented to specific software development approaches. This orientation has two repercussions: (1) CMs have computational constraints, that is, CMs developers represent specific implementation characteristics in the domain models (Bonfatti and Monari, 1994; Høydalsvik and Sindre, 1993; McGinnes, 1992); (2) CMs prescribe the subsequent development process, in which the CMs are more or less directly transformed into design models (Henderson-Sellers and Edwards, 1990; Davis, 1993; Jalote, 1997; Northrop, 1997; Juristo and Moreno, 2000), and their transformation to a design model related to another development paradigm is exceedingly complicated and sometimes impossible.

This means that the CMs used nowadays are not appropriate for analysis. In this paper, we propose an alternative approach that aims to remove the above constraints. The paper is structured as follows: Section 2 discusses the problems with using CMs identified by several researchers, and establish the requirements for an appropriate CM. Sections 3 and 4 describe our approach for a conceptual modelling process independent of any development paradigm. Finally, the preliminary results of applying our approach are discussed in Section 5.

2. The computational orientation of conceptual models

The term CM originally emerged in the database field. CMs were used to represent the data and relations that were to be managed by an information system, irrespective of any implementation feature. Nevertheless, CMs are used for more than is acknowledged in databases. CMs are used in RE to

- encourage the analyst to think and document in terms of the problem, as opposed to the solution (Davis, 1993);
- describe the universe of discourse in the language and in the way of thinking of the domain experts and users (Beringer, 1994);
- formally define aspects of the physical and social world around us for the purposes of understanding and communication (Loucopoulos and Karakostas, 1995);
- help requirements engineers understand the domain (Kaindl, 1999).

Taking into account the above definitions, the main characteristics of any CM can be said to be *description* and *understanding*. That is, CMs should be used by developers to

- understand the user needs;
- reach agreement with users on the scope of the system and how it is to be built;
- use the information represented in the model as a basis for building a software system to meet user needs.

Several authors have pointed out that current CMs sometimes fail to do their jobs of description and understanding during analysis. Criticisms can be divided into two major groups:

- The orientation of the conceptualisation methods, stressing the fact that most CMs are oriented to getting a computational solution to the problem or need raised and not to easing the understanding of the user need. For instance, regarding to object orientation:
  - It is argued that object-oriented methods are a 'natural' representation of the world. Nevertheless, this idea is a dangerous over-simplification (McGinnes, 1992).
  - Object-oriented analysis has several shortcomings, most importantly in being target oriented rather than problem oriented (Høydalsvik and Sindre, 1993).
  - Object-oriented analysis techniques are strongly affected by implementative issues (Bonfatti and Monari, 1994).

  Thus, for example, data flow diagrams (DFD) are clearly guided by functions, the key components of structured software, and, likewise, the models used in object-oriented analysis lead directly to software developed by means of classes, objects, messages, polymorphism, etc., the basic concepts of object-oriented software.
- The association between CMs and specific approaches to software development. Here, the use of a given CM during early phases of the development limits the number of possible implementation alternatives and means that only the options that are compatible with the CM used originally are feasible. If computational characteristics are included in CMs, these are linked to a particular implementation approach, that is, once a given conceptualisation method has been selected to describe the problem domain, it is practically impossible to change the above method a posteriori without having to reanalyse the problem. This has also been stressed by several researchers:

- ◦ Because of a poorly understood overlap among different requirements languages, it is difficult to change languages mid-project (Davis et al., 1997).
- ◦ The use of a CM during analysis defines nearly univocally how the design shall be done (Henderson-Sellers and Edwards, 1990).
- ◦ Perhaps the most difficult aspect of problem analysis is avoiding software design (Davis, 1993).
- ◦ It is sometimes mistakenly believed that the structures produced during analysis will and should be carried through in design (Jalote, 1997).
- ◦ The boundaries between analysis and design activities in the object-oriented model are fuzzy (Northrop, 1997).
- ◦ The CM used preconditions the software system development approach (Juristo and Moreno, 2000).

Owing to this limitation, if data flow diagrams have been used to model the problem domain, for example, it will almost certainly be necessary to use the structured method in later development phases; whereas a method of object-oriented development will have to be used following upon an object-oriented analysis. Therefore, if we intended to switch development paradigms, that is, for example, pass from a data flow diagram to an object-oriented design, this transformation would lead to an information gap that is very difficult to fill. This occurs because each CM acts like a pair of glasses used by the developer to observe the domain and user reality. These glasses highlight certain features, tone down others and hide others. Once the real world has been filtered through the CM, it is difficult to retrieve anything that has been lost or condensed, even if the later development process requires this information. The only way of recovering the features lost in the CM filter is to reanalyse reality using a different pair of glasses; that is, to repeat the operation using another CM. Authors like Coleman et al. (1994), Champeaux et al. (1993) or Wieringa (1991) have already discussed this situation, addressing the incompatibility between the CMs used in the structured approach and object-oriented CMs, owing to the conceptual difference between the elements used in both approaches.

In short, the software system development approach can be said to be preconditioned from the very start, as soon as the CMs are built. The problem with including computational considerations within the CM is that developers are forced to make a solution-oriented decision during the early development phases, when the problem to be solved is still not well enough understood. This means making design decisions when not all the information relevant to the problem is known. Developers thus run the risk of making the wrong decision, because they are not in possession of all the information. Excepting trivial problems, this precondition implies that the development approach is chosen before the user need has been understood, which is the job of conceptual modelling. Even worse, very often the CMs selected are models with which developers are familiar, models called for by individual standards or even, as specified by Mylopoulos et al. (1999), the models that are "in fashion". So, in the era when the structured approach was in vogue, techniques such as DFDs were used for conceptual modelling, whereas, today, with the rise of object-oriented programming and design, techniques like object diagrams, interaction diagrams, etc., are employed for problem analysis.

In order to avoid the commented problems of current CMs, they should include all the information required about the problem for developers to later address the software system that is to solve the user problem. Indeed, it is needed that conceptualisation methods meet the following:

- Understanding the need raised by the user before considering an approach for developing a software system that meets this need.
- The understanding of the need must be independent of the chosen problem-solving approach, that is, it must not precondition the use of any development approach.
- Having criteria for deciding which is the best development approach once the user need has been understood.

These criteria can only be met by redefining the conceptual modelling process as it is now carried out in the RE analysis task.

3. An implementation-paradigm independent conceptual model

The proposed approach, called "problem-oriented analysis method" (POAM), tries to meet the above-mentioned criteria, and is characterised by (1) using representation diagrams, which we call generic conceptual models (GCMs), that do not presuppose any implementation paradigm; (2) defining a detailed analysis process; and (3) deriving, from the GCM, the best-suited CM (that is, a CM now used in RE, like DFD, use cases, etc.) to continue with development according to the methods used nowadays. The following sections present the main components of the proposed approach, that is, the GCM, which is described in Section 3.1, and the POAM process, which is discussed in Section 3.2.

3.1. Generic conceptual model

The CMs currently used in software engineering have to be used exclusively, that is, mostly rule out the use of a complementary CM. In some cases, when using a DFD, for example, the use of supplementary notations, such as process specifications or even the entity/relationship model, is permitted, although these are subordinated to the process diagram set out in the DFD.

The GCM proposed in this approach is based on complementariness. Instead of using a representation schema that dominates the modelling process, three complementary representation schemas are used. Complementary means: (1) each schema supports the others, satisfactorily recording information they do not represent and (2) the information in one schema can migrate, that is, move from one schema to another without the GCM losing information.

Complementariness is important because the way the information is expressed benefits or impairs its understanding (Vessey, 1991). The different components of the GCM can represent the same information, expressing it either as a graph, table or text. This means that each analysis process participant can select and use the best-suited expression, either on the basis of previous experience or in accordance with current needs.

The proposed GCM components are as follows:

- *Element maps:* Information representation structures belonging to a given knowledge domain or problem. Element maps are variations on the conceptual maps, derived from the work of Ausubel on Learning Theory and Psychology, later formalised by Novak and Gowin (1984). We use the term ‘element maps’ instead of ‘concepts maps’, because ‘concept’ is a overloaded word in SE. For example, ‘concept’ is often used to mean ‘data’. When we use ‘element’, we are talking about ‘static concepts’, like data, rules or facts, but also about ‘dynamic concepts’, like processes, events, and so on.

  Conceptual maps (as employed in psychology) can be used to express and graphically represent concepts and associations between concepts as a hierarchical structure. Element maps differ from conceptual maps used in Psychology on three essential points: (1) they are generally structured as a complex graph and not necessarily hierarchically; (2) both the concepts (elements in our approach) and the associations, which represent established knowledge in conceptual maps, are likely to evolve over time as the analysis progress and (3) some special concepts (elements in our approach) and associations have been defined to restrict the spectrum of possible readings of the elements map for the purpose of raising the efficiency of POAM application.
- *Dictionaries:* Tabulated information representation schemas. Dictionaries have a set of predefined fields that define what sort of information they record. There are two main types of dictionaries:
  - *Identificative dictionary (or glossary):* This dictionary merely records the information required to recognise a element or association appearing while investigating the problem and to distinguish one element or association from another.
  - *Descriptive dictionary:* Its goal is to record negotiated information about elements and associations, that is, information that all the participants in the analysis process agree to be true. This information is, additionally, practically complete, that is, all the important aspects of the problem and its solution will have been identified and recorded if this dictionary has been correctly built.
- *Narrative description:* Natural language text that describes the information recorded in the elements map and the dictionaries. The narrative description can be automatically derived from the elements map and dictionaries (although the result is not a literary masterpiece), which has some clear benefits for model validation. The text is very understandable for end users and, as there is a bijective relationship between the narrative description and the other representation schemas, the comments and corrections made by the users can be fed back into those schemas.

The three above-mentioned representation schemas are used during the POAM process activities and steps. The POAM process is described below.

3.2. Generic conceptual model development process

There are two points of inflection during analysis, each determined by its goals, that is: (1) move from ignorance to an understanding of the problem to be solved, which should be reflected in the creation of CMs, and (2) go from an understanding of the problem to a solution characterization, which moves from a very abstract level in the early stages of analysis (some restrictions, characteristics, etc., of the future software system) to a more concrete formulation as the developer’s knowledge about the problem increases (a list of the desired software system features). Therefore, the proposed process is composed of two activities, as shown in Fig. 1(a). The two activities differentiate two states in analysis: a problem-oriented state and a solution-oriented state.

The goal of the first activity, called *problem-oriented analysis*, is to understand the problem to be solved and ends when the GCM, which represents the acquired knowledge, has been developed. This GCM is the input for the second activity, called *software-oriented analysis*, whose goal is to identify which typically used CM is best suited for representing the problem, as well as to transform the GCM into the above-mentioned CM.

This first level of decomposition is too general to guide a developer as to how to perform analysis. Therefore, both activities are divided into two steps, as shown in Fig. 1(b), which are further broken down into

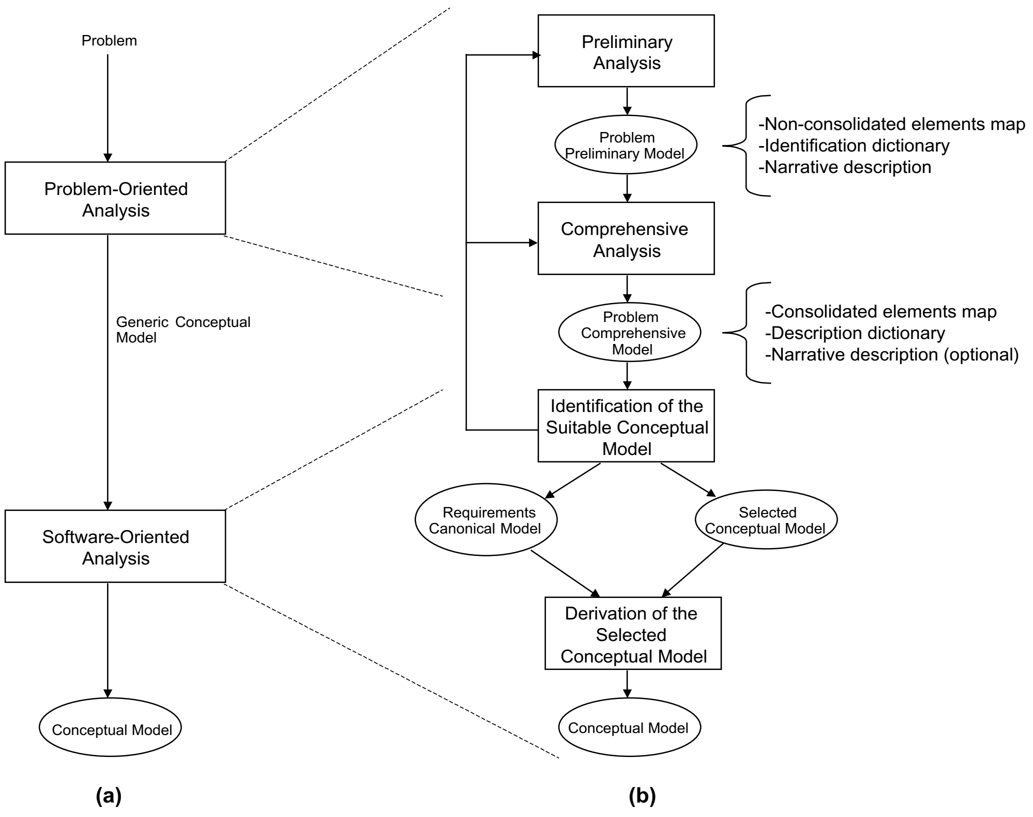

Problem
Problem-Oriented Analysis
Generic Conceptual Model
Software-Oriented Analysis
Conceptual Model
(a)
Preliminary Analysis
Problem Preliminary Model
-Non-consolidated elements map
-Identification dictionary
-Narrative description
Comprehensive Analysis
Problem Comprehensive Model
-Consolidated elements map
-Description dictionary
-Narrative description (optional)
Identification of the Suitable Conceptual Model
Requirements Canonical Model
Selected Conceptual Model
Derivation of the Selected Conceptual Model
Conceptual Model
(b)

Fig. 1. Proposed process.

detailed tasks. Thus, *problem-oriented analysis* is decomposed into the following steps:

- *Preliminary analysis:* In this step, the problem is examined superficially with the aim of defining a *preliminary model*. The goals of this step are to (1) identify the most important elements of the problem domain; (2) describe these elements; and (3) organise all the elements of the problem domain into a structure, by means of which to define the associations there are among these elements.
- *Comprehensive analysis:* In this step, the problem is studied in as much detail as required to develop the *comprehensive model*, that is, the complete GCM. The goals of this step are to (1) check that the important problem elements have been identified; (2) describe the above elements exhaustively and (3) clearly determine the associations among elements.

In the above paragraphs, we introduced the concepts of *preliminary* and *comprehensive models*. The preliminary model is a simplified version of the GCM, obtained after the *preliminary analysis*, which is composed of (1) a elements map—usually hierarchical and not generally a graph, (2) identificative dictionary and (3), narrative descriptions. The comprehensive model, that is, the complete GCM output at the end of the *comprehensive analysis*, differs from the preliminary model in that (1) the elements map is more detailed and is generally a graph, (2) the descriptive dictionary is used instead of the identificative dictionary and (3) the narrative description is optional and is usually excluded.

Having completed the *problem-oriented analysis*, we will get an exhaustive description of all the important problem elements and of the spectrum of associations between these elements. This information, contained in the GCM, is of intrinsic value, as it helps developers and other participants in analysis to understand the problem, which is one of the key objectives of analysis.

Using the proposed approach, however, we can go even further to derive, from the information contained in the GCM, a CM by means of which to continue software system development using any of the development approaches now available, such as structured, object-oriented or real-time approaches. That CM is derived in the *software-oriented analysis* activity. This activity is decomposed into two steps.

- *Identification of the suitable conceptual model:* In this step, we identify the suitable conceptual model (SCM). The SCM is the target CM that can represent all the information gathered in the GCM for a given problem more fully.

An interpretation procedure has to be applied to the GCM to identify the SCM. The interpretation procedure can be used to rewrite the GCM from a computational viewpoint, that is, to assign builders used by the classical CMs to the constituent elements of the GCM, which in turn form the building blocks of computer systems.

We have used a requirements representation formalism proposed by Davis et al. (1997) for rewriting purposes, although it has been profoundly modified for use with the GCM. This formalism, termed "canonical model", in accordance with its author's intent, provides a set of building blocks that can be used to represent the information contained in a range of CMs. This means that it can be used as a *lingua franca*, which averts, as explained below, having to deal with each CM separately.

The interpretation procedure, therefore, involves assigning a computational interpretation to each of the building blocks of the GCM or, in other words, assigning each GCM element to one of the canonical model elements.

This assignation will be totally formalised and engineer independent, unless any ambiguities arise in the assignation. Ambiguity is the possibility of assigning two or more elements of the canonical model to any given GCM element. In this case, it is the engineer who has to decide, depending on the semantics of the GCM and the canonical model, which particular interpretation is the best suited.

After interpretation, the GCM is called the requirements canonical model (RCM), as the GCM can now be read in computational terms, as a description of what should be future software system operation. After outputting the RCM, we can determine the SCM.

The SCM will be the CM that is capable of representing most RCM propositions. We have defined a measure, called fitness, to give a quantitative value of suitability. Fitness is defined as the ratio between the propositions a given CM can represent and the total number of RCM propositions. Accordingly, the SCM is the CM with the highest fitness value.

Additionally, this measure provides supplementary information, namely, the extent to which the SCM is suitable. For example, a CM may be suitable (that is, be the best of all the models) and still very partially represent the information gathered about the problem domain (in this case, low fitness values would be obtained). Additionally, it can even establish what difference, in terms of representation capability, there is between two particular CMs (which would be the difference between the respective fitness values).

- *Derivation of the selected conceptual model:* In this step, the RCM is translated into the target CM.

We use a derivation procedure to generate the target CM. The derivation procedure basically involves using a set of derivation tables and rules. There are as many tables as there are possible target CMs. Each derivation table contains all the possible combinations of canonical model elements that can be expressed in a given target CM, along with the expression of this combination in the particular format used by the CM in question (graphs, text, tables, etc.).

As each GCM element has been labelled in the RCM and we have calculated the fitness of the different CMs, we can now refer to the appropriate derivation table and use it to directly generate fragments of the target CM. These fragments can later be assembled, unambiguously, to finally output the correct target CM.

The derivation rules modulate the use of each derivation table, altering the RCM in a controlled manner, so that the target CM finally obtained resembles as closely as possible a target CM developed independently for the same problem.

The target CM obtained in the above step can be refined by entering more information. However, this refinement is neither direct, nor can it be formalised, owing to the fact that the GCM cannot be interpreted directly in computational terms. Therefore, the developer will have to select what knowledge to record in the target CM and what to discard. Once complete, the target CM will have the same drawbacks as CMs developed directly, that is, some knowledge about the problem will have been lost and the target CM will be linked to a given development approach. Nevertheless, there is a big difference between filtering problem elements using the current and the proposed conceptual modelling processes. With the development processes now in use, developers do not take into account the problem elements that are not compatible with the CM used (DFD, use cases, etc.) *before the problem is understood*. Using the proposed approach, developers are encouraged to study and record all the possible problem perspectives in the GCM. Therefore, the loss of knowledge occurs *when the problem has been understood*, thus avoiding early decisions on how to solve the problem at hand.

4. Conceptual modelling using our proposal

An example showing the steps of the proposed process, as well as the use of the components of the GCM is given in the next section. This example will illustrate all the theory explained above. Suppose we have the following problem, set out in natural language:

Hospital 123 has two patient admission procedures. The first is the admission of patients on a waiting list. The second is the admission of patients who visit the emergency department.

When a patient on a waiting list is admitted, the patient is assigned a room on a ward depending on the com-

plaint that is to be treated. For example, if a patient is to undergo coronary by-pass surgery, the patient would be assigned to a room on the cardiology ward.

The patients admitted from the waiting list are assigned a reference physician. This physician can be any doctor belonging to the speciality related to the complaint that is to be treated.

On the other hand, patients who are admitted from the emergency department are immediately treated prior to any administrative procedure. Once treated, they are assigned a room no later than three hours after admission, according to the same rules as patients admitted from the waiting list. The only difference is that their reference physician will be the doctor who treated them in the emergency department rather than a physician of any particular speciality.

At first glance, this problem could apparently be modelled in several different ways. For example, given the problem characteristics (objects present, transformation processes that seem to exist, etc.), a data flow diagram would appear to be a suitable representation, as would an entity/relationship or a class diagram. However, the use of POAM makes it unnecessary to hypothesize, in this moment, which is the best-suited diagram type. During analysis, the problem is modelled using the GCM and, only later, before passing on to design, will we decide which is the best-suited CM and, depending on this decision, which development approach will be most effective for building the future software product.

The first step of POAM is preliminary analysis. As this is not a real case, but a test case where (1) the information is not acquired incrementally, as happens during elicitation, (2) there are no ambiguities and (3) complexity is controlled at minimum levels, the modelling output after preliminary analysis would be approximately as shown in Fig. 2.

Fig. 2 shows the preliminary element map. This map shows the key elements present in the problem description (*patients, doctors, rooms, wards*, etc.), as well as the key associations (a patient is *admitted* from the waiting list or the emergency doctor is the *reference physician* of an emergency patient). The preliminary element map is easily confused, during preliminary analysis, with semantic data models or class diagrams. However, this is only a seeming similarity, as, in this intermediate step of POAM, we have mainly described the structural aspects of the problem, which are, precisely, the aspects on which the above-mentioned conceptual models focus.

The preliminary elements map can be likewise expressed by means of the identificative dictionary, or glossary, shown in Table 1 or by means of narrative text, as shown in Table 2.

Note that each representation is similar to, while, at the same time, slightly different from, the others. This is due to the fact that each GCM representation mechanism focuses on different aspects of the information acquired. The element map highlights, primarily, the associations between the different elements, whereas the

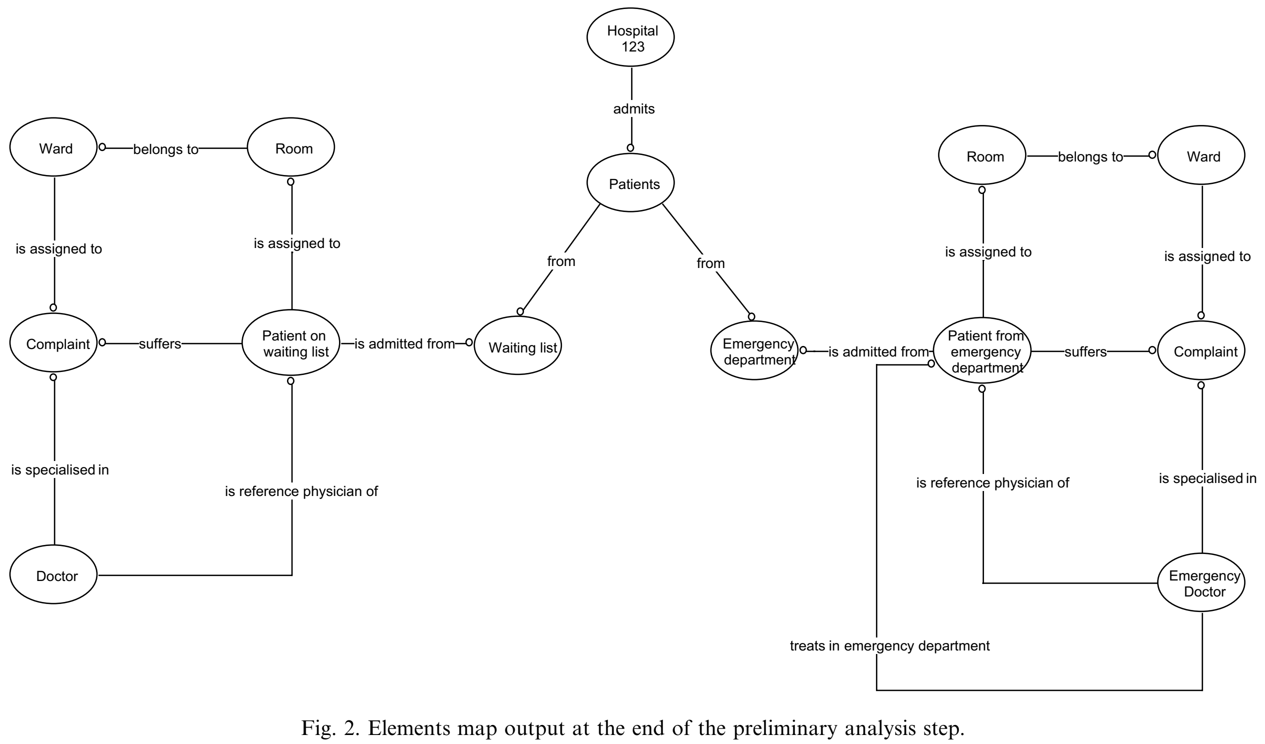


Fig. 2. Elements map output at the end of the preliminary analysis step.

Table 1
Identificative dictionary

Element	Description
Hospital 123	Admits patients
Patients	From waiting list
	From emergency department
Waiting list patient	Is assigned a room
	Suffers complaint
	Is admitted from waiting list
Emergency department patient	Is assigned a room
	Suffers complaint
	Is admitted from emergency department
Doctor	Is reference physician of waiting list patient
	Is specialised in complaint
Emergency doctor	Is reference physician of emergency department patient
	Is specialised in complaint
	Treats patient in emergency department
Room	Belongs to ward
Room	Belongs to ward
Ward	Is assigned to a complaint
Ward	Is assigned to a complaint
Complaint	
Complaint	

The apparent duplication in recorded information (two different entries for *room*, *ward* and *complaint*, are due to a GCM technicism: two different elements in the elements map are considered different even if they bear the same name. This rule prevents information loss, primarily in the early stages of analysis, as information that appears to be similar may turn out to be very different later.

Table 2
Narrative text

Hospital 123 admits patients from the waiting list or from the emergency department
Waiting list patients are assigned a room and suffer a complaint and are admitted from the waiting list
Emergency department patients are assigned a room and suffer a complaint and are admitted from the emergency department
Doctor is the reference physician of the waiting list patient and is specialised in the complaint
Emergency doctor is the reference physician of the emergency department patient and is specialised in the complaint and treats the emergency patient in the emergency department
Room belongs to ward
Ward is assigned to complaint

The text has been slightly (but not arbitrarily) modified to improve readability.

identificative dictionary emphasises the meaning of each element. Finally, the narrative text locates elements and associations at the same level, easing communication with the less technically competent participants in the analysis task.

The comprehensive analysis step outputs a higher level of refinement in the information acquired about the problem. This refinement is achieved by refining the elements and associations in the elements map. Of these refinements, the distinction between sets and individuals, the identification of the properties of the different elements and the addition of dynamic or time details that may have been overlooked during preliminary analysis are of special importance. The element map output is shown in Fig. 3.

In any case, note that it is difficult to read the comprehensive element map in terms of any particular implementation. This map takes the form of "spaghetti", where structural, logical and dynamic aspects are mixed coherently, which means that it can be expressed in a variety of conceptual models at the same time. Therefore, the comprehensive element map does not predetermine any particular implementation, rather it can be used to rewrite the problem in a variety of different ways, each suited to a particular development approach.

The interpretation procedure has to be applied to identify the best-suited development approach. For this purpose, it is preferable for the information recorded in the comprehensive elements map to be transcribed to the descriptive dictionary shown in Table 3. The descriptive dictionary is obtained directly from the comprehensive elements map after applying a set of formal transcription rules (Dieste, 2003).

The descriptive dictionary provides a compact description of the information recorded in the comprehensive elements map, which makes it easier to apply the interpretation procedure. This procedure is composed, as mentioned above, of several steps, which are not discussed for reasons of space. The final result of applying the interpretation procedure, that is, the RCM, would be as shown in Table 4.

Several propositions have been interpreted by the analyst in the RCM shown in Table 4, because several links can be used between two elements of type Entity, as shown in Table 5, which states all possible combinations among elements and links. These propositions have been interpreted by selecting the link closest to the meaning of the association in each particular case.

Having output the RCM, the SCM is identified and derived automatically. To identify the SCM, we have to calculate how many RCM propositions can be expressed in each of the classical conceptual models. They are calculated using the identification tables. The identification tables are complementary to Table 5, as they identify what CM(s) can express each element–link–element combination in Table 5. By way of an example, Table 6 shows the identification table for the class diagram.

Using the identification tables, the number of propositions supported by each CM can be calculated directly. This will compute the fitness of each model and, finally, identify the SCM. Considering only the most popular CMs, such as the data flow diagram (DFD), the entity–relationship diagram (ER), the class diagram (CD), the state transition diagram (STD), the statechart (SCT) and use cases (UC), the fitness calculation would be as shown in Table 7.

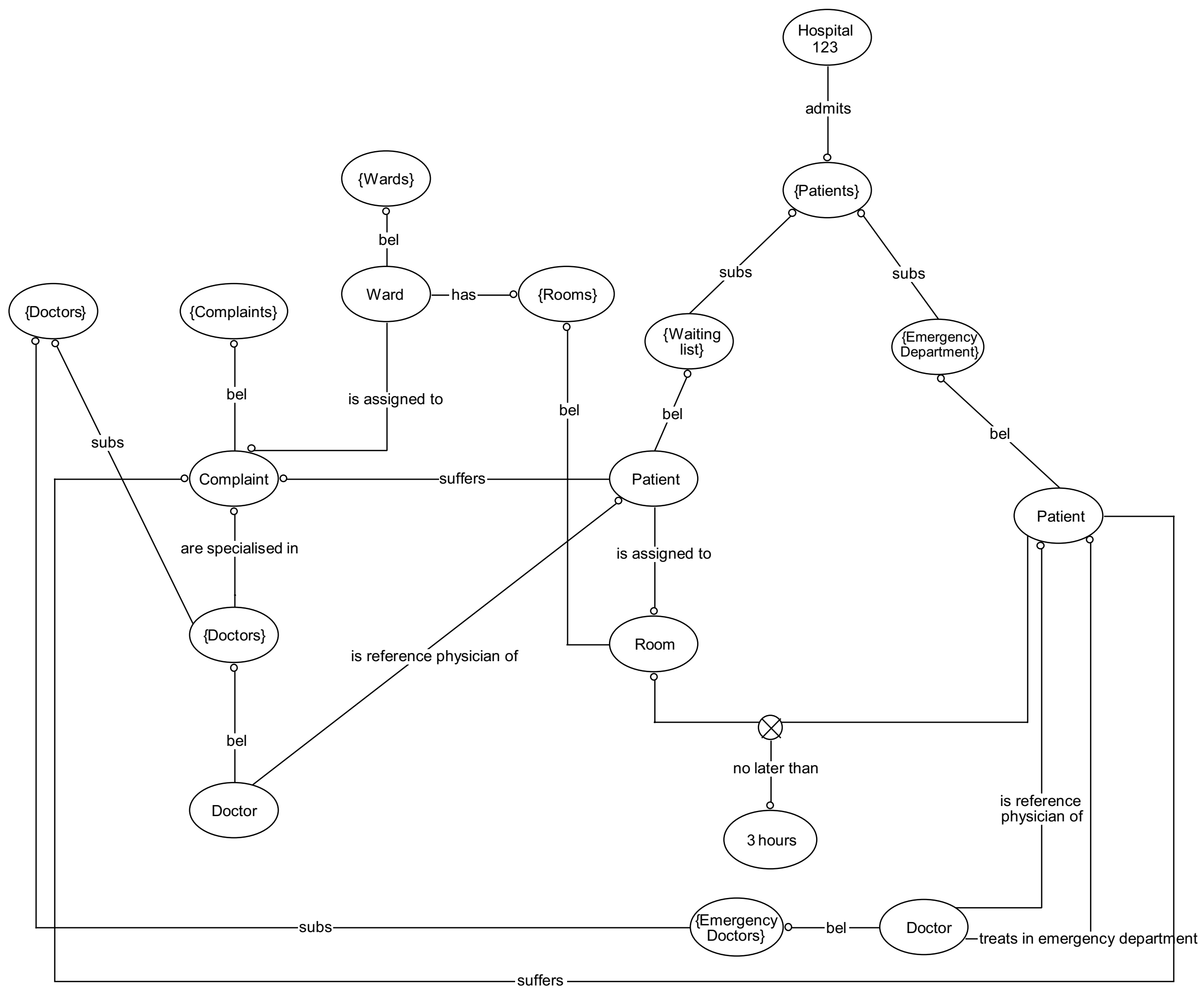


Fig. 3. Comprehensive elements map.

Table 7 shows that the SCM is the class diagram, as its fitness is greater than all the CMs (at least, of all the ones that have been considered in the SCM calculation) and it can express 71% of all the RCM propositions. If the class diagram is the SCM, the best-suited development approaches are all the ones that use a class diagram, such as the different object-oriented approaches.

Having identified the SCM, all we have to do now is use the derivation tables and rules to get the target CM. There are as many derivation tables and rules as there are CMs (Dieste, 2003). These tables and rules can be used to get fragments of the target CM from the propositions it expresses. For example, Table 8 shows the derivation table for the class diagram (SCM for the example introduced in this section).

For example, from the proposition:

Entity[notrepl] : Hospital 123 Rel : admits Entity[repl] : Patients

We can get the fragment shown in Table 9, as the derivation table contains an entry "Entity[repl] Rel Entity[repl]".

The different fragments can then be assembled, unambiguously, to get the final version of the diagram. In the case discussed in the example, the diagram output is shown in Fig. 4.

The diagram shown in Fig. 4 is not the best possible class diagram, it is the class diagram that can be derived from the GCM. This model can be later modified to improve or complete diagram aspects, in order to make the resultant class diagram clearer and simpler.

5. Validation

The proposed approach has been validated and refined after running several case studies using POAM and using the CMs prescribed by other development

Table 3
Descriptive dictionary

Statements			
Sets	Rooms		
	Wards		
	Complaints		
	Patients		
	Doctors_1		
	Doctors_2		
	Emergency doctors		
Subsets	Waiting list	subs	Patient
	Emergency dept	subs	Patients
	Doctors_2	subs	Doctors_1
	Emergency doctors	subs	Doctors_1
Individuals	Complaint	bel	Complaints
	Room	bel	Rooms
	Patient_1	bel	Waiting list
	Patient_2	bel	Emergency Dept
	Ward	bel	Wards
	Doctor_1	bel	Doctors_2
	Doctor_2	bel	Emergency Doctors
Definitions Index	Definition		
Propositions Index	Association	Concept-1	Concept-2
1	Hospital 123	admits	Patients
2	Doctors_2	is specialised in	Complaint
3	Patient_1	suffers	Complaint
4	Patient_1	is assigned	Room
5	Ward	has	Rooms
6	Ward	is assigned to	Complaint
7	Doctor_1	is reference physician of	Patient_1
8	Doctor_2	treats in emergency dept.	Patient_2
9	Doctor_2	is reference physician of	Patient_2
10	Patient_2	is assigned	Room
11	Patient_2	suffers	Complaint
12	p10	no later than	3 h

approaches (structured, object-oriented and real-time approaches).

The set of test cases used was representative of the problems and situations typically encountered during software systems development. Because the problem classification schemas existing in the literature are not very formal (Glass and Vessey, 1995), the design of the test cases was guided by more formal CM classification schemas (Webster, 1988; Zave, 1990; Davis, 1993; Blum, 1996). We used Davis' classification schema (Davis, 1993), as it is the simplest and, probably, the most well known, to the point that it is practically isomorphorous with the three major types of development approach (structured, object-oriented and real-time approaches).

Davis (1993) divides the models into three groups: function-oriented, object-oriented and state-oriented (hereinafter they are referred to as F, O and S models, respectively). Each model type is suited for the problems whose structure more or less matches the building blocks of these models (that is, what can be termed F, O and S problems, respectively). This can be used to reasonably build test cases that are representative of these problem types. Additionally, test cases can be built for mixed problems (FO, FS, OS, FOS).

A total of seven test cases were designed (F, O, S, FO, FS, OS and FOS types). These test cases were solved by a series of subjects with differing software development experience levels, which were divided into four different sized groups, owing to availability problems:

- Four professors (control group—CG). The CG members solved the test cases using the models of the three development approaches (structured, object-oriented and real-time approaches).
- Five developers experienced in at least one development approach (structured, object-oriented or real-time approaches). The developers formed test group 1 (TG1). The members of TG1 were allowed to use the models they considered to be best suited to solve the test cases.
- Eighteen graduate and post-graduate students (test group 2—TG2). The members of TG2 could, likewise, use the classical approach that they felt was better suited. This second group also had to fill in a questionnaire containing several questions regarding how well acquainted they were with the conceptual model they selected, so that we could check whether the selection was based on the problem features or on the subjective preferences of each experimental subject.
- Three graduate and postgraduate students (test group 3—TG3), who solved the test cases using POAM.

The aim of the CG was to verify that the design of the test cases was not biased, that is, that each case really corresponded with a F, O, S, FO, FS, OS or FOS problem. The goal of TG1 and TG2 was to compare POAM effectiveness against the usual way of selecting CMs in software development by both experienced practitioners and recently trained or nearly trained professionals. Finally, the goal of TG3 was to study CM selection using the proposed approach (POAM).

For every test case, we compared the CMs output using the four different approaches (POAM, structured, object-oriented and real-time), and we evaluated:

- That each test group belonged to the problem group for which it was designed based on CG's judgement and comparing the models generated using the structured, object-oriented and real-time approaches.

Table 4
Requirements canonical model

Statements			
Sets	Entity[repl]: Rooms		
	Entity[repl]: Wards		
	Entity[repl]: Complaints		
	Entity[repl]: Patients		
	Entity[repl]: Doctors_1		
	Entity[repl]: Doctors_2		
	Entity[repl]: Emergency doctors		
Subsets	Entity[repl]: Waiting list	Subs: subs	Entity[repl]: Patient
	Entity[repl]: Emergency dept.	Subs: subs	Entity[repl]: Patients
	Entity[repl]: Doctors_2	Subs: subs	Entity[repl]: Doctors_1
	Entity[repl]: Emergency doctors	Subs: subs	Entity[repl]: Doctors_1
Individuals	Entity[notrepl]: Complaint	Bel: bel	Entity[repl]: Complaints
	Entity[notrepl]: Room	Bel: bel	Entity[repl]: Rooms
	Entity[notrepl]: Patient_1	Bel: bel	Entity[repl]: Waiting list
	Entity[notrepl]: Patient_2	Bel: bel	Entity[repl]: Emergency dept.
	Entity[notrepl]: Ward	Bel: bel	Entity[repl]: Wards
	Entity[notrepl]: Doctor_1	Bel: bel	Entity[repl]: Doctors_2
	Entity[notrepl]: Doctor_2	Bel: bel	Entity[repl]: Emergency doctors
Definitions			
Index	Definition		
Propositions			
Index	Association	Concept-1	Concept-2
1	Entity[notrepl]: Hospital 123	Rel: admits	Entity[repl]: Patients
2	Entity[repl]: Doctors_2	Rel: are specialised in	Entity[notrepl]: Complaint
3	Entity[notrepl]: Patient_1	Rel: suffers	Entity[notrepl]: Complaint
4	Entity[notrepl]: Patient_1	Rel: is assigned	Entity[notrepl]: Room
5	Entity[notrepl]: Ward	-Pof: has	Entity[repl]: Room
6	Entity[notrepl]: Ward	Rel: is assigned to	Entity[notrepl]: Complaint
7	Entity[notrepl]: Doctor_1	Rel: is reference physician of	Entity[notrepl]: Patient_1
8	Entity[notrepl]: Doctor_2	Rel: treats in emergency dept.	Entity[notrepl]: Patient_2
9	Entity[notrepl]: Doctor_2	Rel: is reference physician of	Entity[notrepl]: Patient_2
10	Entity[notrepl]: Patient_2	Rel: is assigned	Entity[notrepl]: Room
11	Entity[notrepl]: Patient_2	Rel: suffers	Entity[notrepl]: Complaint
12	p10	Constraint: no later than	Value: 3 h

The numbers after the names of some elements (doctors, patient, etc.) are used to make a distinction between elements that have the same name in the comprehensive elements map.

- How effective TG1, TG2 and TG3 were in predicting the best-suited conceptual model.
- The correctness and validity of the conceptual models output by TG1, TG2 and TG3.
- The time it took TG1, TG2 and TG3 to develop the conceptual models.

The results obtained so far appear to support the following findings:

- The test cases were designed properly. The opinion of the CG in this respect was unanimous, as it selected the model for which the test case was designed (F, O, S, etc.) in each case. Additionally, the selected model was clearly the best suited when compared against the other models (F, O, S, etc.) for the same test case.
- Several points should be considered with respect to effectiveness:
  - The effectiveness of TG3 was 100%, where the best-suited model was selected in all seven test cases.
  - The effectiveness of TG1 and TG2 was clearly lower than TG3. With respect to TG1, only one subject identified the best model for all the test cases, whereas two members selected the correct model in six cases and the other two in four of the seven cases. With respect to TG2, only 33% of the subjects identified the best model in all seven test cases. Another 33% correctly identified six of the seven test cases, and the other 33% came up with the correct model in five or less cases. In mean terms, the effectiveness of TG1 was 77% correctly solved cases, whereas the effectiveness of TG2 was 83%.
  - To check the claims made in this paper about software developer tendentiousness and preferences for a given development approach, we studied

Table 5
Possible combinations among elements and links when applying the interpretation procedure

	Entity[repl]	Entity[notrepl]	Process	Predicate	Transition	Message	Constraint	Value	Statespace
Entity[repl]	spec subs pof rel activate								
Entity[notrepl]	spec pof rel bel activate	spec pof rel activate							
Process	pof sends receives -activate	pof sends receives -activate	spec pof activate						
Predicate	operand	operand	activate	operand					
Transition			stimulus response	stimulus	stimulus response				
Message	-sends -receives	-sends -receives	-sends -receives	-operand	-stimulus -response	pof			
Constraint	-operand	operand	pof	-operand	-operand	-operand	operand		
Value	-sends -receives	-sends -receives	-sends -receives pof	-operand	-stimulus -response	-operand	-operand	pof	
Statespace	pof -sends -receives	pof -sends -receives	-sends -receives pof	-operand	-stimulus -response	-operand	-operand	hval	pof spec

Separate tables should be used when proportions are considered. The table is symmetric and, therefore, the top right-hand side is not shown.

Table 6
Identification table for the class diagram

	Entity[repl]	Entity[notrepl]	Process	Predicate	Transition	Message	Constraint	Value	Statespace
Entity[repl]	spec subs pof rel								
Entity[notrepl]	spec pof rel	spec pof rel							
Process	pof	pof							
Predicate									
Transition									
Message									
Constraint									
Value	pof	pof							
Statespace	pof	pof							

The table is symmetric and, therefore, the top right-hand side is not shown. The blank boxes indicate element–link–element combinations that are not permitted in the class diagram.

Table 7
Determination of conceptual model fitness

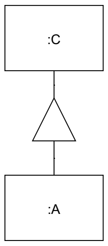
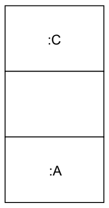
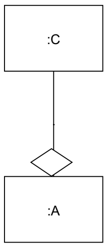
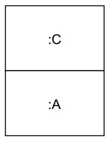
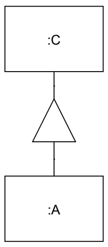
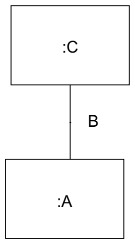
Models				DFD	ER	DC	DTE	STC	CU
	Entity[repl]: Rooms			√	√	√			√
	Entity[repl]: Wards			√	√	√			√
	Entity[repl]: Complaints			√	√	√			√
	Entity[repl]: Patients			√	√	√			√
	Entity[repl]: Doctors_1			√	√	√			√
	Entity[repl]: Doctors_2			√	√	√			√
	Entity[repl]: Doctors			√	√	√			√
	Entity[repl]: Waiting list	Subs: subs	Entity[repl]: Patient			√			
	Entity[repl]: Emergeny dept.	Subs: subs	Entity[repl]: Patients			√			
	Entity[repl]: Doctors_2	Subs: subs	Entity[repl]: Doctors_1			√			
	Entity[repl]: Emergency doctors	Subs: subs	Entity[repl]: Doctors_1			√			
	Entity[notrepl]: Complaint	Bel: bel	Entity[repl]: Complaints						
	Entity[notrepl]: Room	Bel: bel	Entity[repl]: Rooms						
	Entity[notrepl]: Patient_1	Bel: bel	Entity[repl]: Waiting list						
	Entity[notrepl]: Patient_2	Bel: bel	Entity[repl]: Emergency dept.						
	Entity[notrepl]: Ward	Bel: bel	Entity[repl]: Wards						
	Entity[notrepl]: Doctor_1	Bel: bel	Entity[repl]: Doctors						
	Entity[notrepl]: Doctor_2	Bel: bel	Entity[repl]: Emergency doctors						
p1	Entity[notrepl]: Hospital 123	Rel: admits	Entity[repl]: Patients		√	√			
p2	Entity[repl]: Doctors_2	Rel: are specialised in	Entity[notrepl]: Complaint		√	√			
p3	Entity[notrepl]: Patient_1	Rel: suffers	Entity[notrepl]: Complaint		√	√			
p4	Entity[notrepl]: Patient_1	Rel: is assigned	Entity[notrepl]: Room		√	√			
p5	Entity[notrepl]: Ward	-Pof: has	Entity[repl]: Rooms			√			
p6	Entity[notrepl]: Ward	Rel: is assigned to	Entity[notrepl]: Complaint		√	√			
p7	Entity[notrepl]: Doctor_1	Rel: is reference physician of	Entity[notrepl]: Patient_1		√	√			
p8	Entity[notrepl]: Doctor_2	Rel: treats in emergency dept.	Entity[notrepl]: Patient_2		√	√			
p9	Entity[notrepl]: Doctor_2	Rel: is reference physician of	Entity[notrepl]: Patient_2		√	√			
P10	Entity[notrepl]: Patient_2	Rel: is assigned	Entity[notrepl]: Room		√	√			
P11	Entity[notrepl]: Patient_2	Rel: suffers	Entity[notrepl]: Complaint		√	√			
P12-1	Constraint: no later than	Operand:	P10						
P12-2	Constraint: no later than	Operand:	Value: 3 h						
Fitness				0.23	0.55	0.71	0.0	0.0	0.23

The division of proposition p12 into two propositions (p12-1 and p12-2) is called splitting and occurs when an association has to be interpreted in logical or dynamic terms. The shading has been used to highlight the propositions that cannot be expressed in any of the conceptual models.

the extent to which the experimental subjects always selected the same model during the experiment. After an analysis of the data output, we found that all (100%) of the subjects belonging to TG1 select the same model in at least four of the seven test cases (57% of cases). The models selected by the subjects belonging to TG2 are more dispersed. However, nine out of eighteen subjects (50%) selected the same model in at least four of the seven test cases (57%). On the other hand, there is a balance between the models identified as best suited by the members of TG3, and no particular model stands out.

- Finally, we studied how the conceptual tendentiousness of the analyst distorts conceptual model selection. Note that the modellers were given the information obtained from an impartial elicitation process in the seven cases studied. As mentioned above, however, the conceptual preferences of an analyst examining a domain act like glasses that give preference to some aspects and disregard others. To check this phenomenon, a special test case was designed that simulated the activity of a tendentious analyst eliciting information from a domain about a problem not suited to his/her preferences. As was to be expected, the tendentious analyst gets information from the domain that gives preference to the problem features that match his/her preferences in detriment to the features least suited to his/her view. This deformed

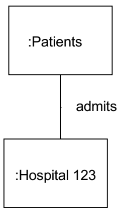
Table 8
Derivation table for the class diagram

Element (A)	Link (B)	Element (C)	Derives	Element (A)	Link (B)	Element (C)	Derives
Entity[repl]	Spec	Entity[repl]		Entity[repl]	Receives	Message	
Entity[repl]	Pof	Entity[repl]		Statespace	Pof	Entity[repl]	
Entity[repl]	Subs	Entity[repl]		Entity[repl]	Rel	Entity[repl]	

Rules:
1. If an element of type Entity[notrepl] also belongs (Bel) to an element of type Entity[repl], then replace it with latter
2. If an element of type Entity[repl] is a subset of (Subs) an element of type Entity[repl], then replace it with the latter
3. If an element Entity[notrepl] satisfies neither rule (2) or (3), consider whether it can be considered coherently as Entity[repl]

We consider only a fragment of the table, which has more entries than shown.

Table 9
Derivation of a fragment of the SCM from the proposition Entity[notrepl]: Hospital 123 Rel: admits Entity[repl]: Patients

Element (A)	Link (B)	Element (C)	Derives
Entity[notrepl]: Hospital 123	Rel: admits	Entity[repl]: Patients	

Note that the class diagram derivation rule (3) has been applied, considering Entity[notrepl]: Hospital 123 as Entity[repl]: Hospital 123.

information was delivered to all the test groups. CG, TG1 and TG2 selected the model preferred by the analyst (even though it was not suited to the problem), whereas TG3 (POAM) was not influenced by the analyst and selected the best-suited model. Note that the analyst's tendentiousness managed to deceive the expert modellers, as they were unable to stop their decision from being influenced by falsely upgraded aspects (to which more importance was attached, more space allotted, etc., in the elicited information) as opposed to falsely downgraded aspects (which, although they appeared in the elicited information, were checked by the elicitor's preferences).

- The validity of the CM obtained using POAM is greater than the CMs constructed using classical approaches. The reason appears to be that students gain a better understanding of the problem by performing the *preliminary analysis* and *comprehensive analysis* sub-activities. Having a better understanding, they are able to create not only a syntactically correct, but also a semantically coherent SCM. Using structured, object-oriented or real-time approaches, students tend to subordinate understanding to representation, outputting semantically ambiguous, albeit syntactically correct, conceptual models.

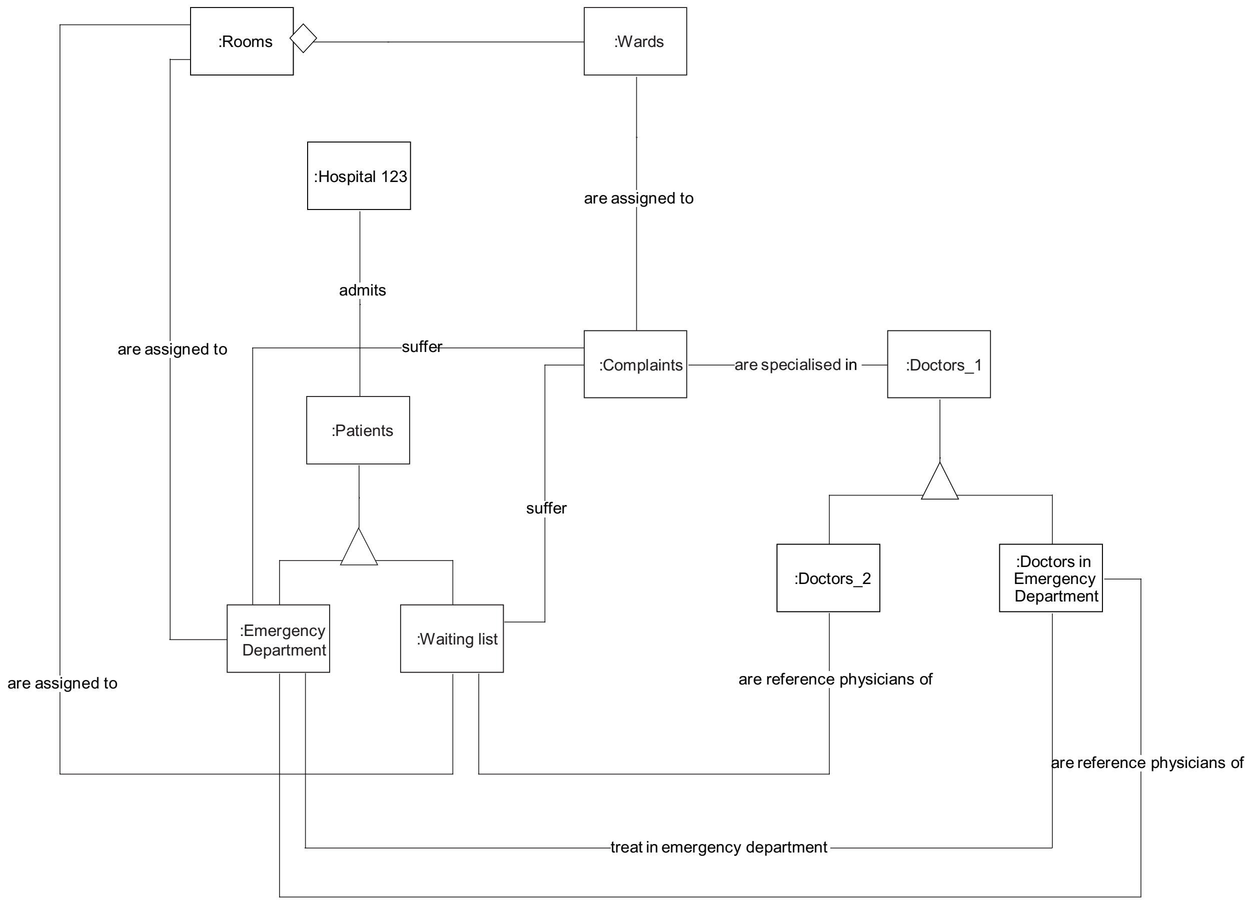


Fig. 4. Final SCM (class diagram).

- It takes longer to develop the conceptual model using POAM. This was a foreseeable point in view of the increased workload involved in subdividing analysis. However, many of the POAM steps can be automated, either fully or partially, which means that a suitable automated support could help to compensate for the additional workload caused by the proposed approach.

From the above results, we can state that POAM is a more effective approach to conceptual modelling than the classical development approaches (structured, object-oriented and real-time approaches). The only shortcomings of POAM appear to be related to the time it takes to build the generic conceptual model and identify and derive the classical conceptual models. However, this shortcoming refers to the efficiency and not to the effectiveness of the proposed approach. With the aim of improving POAM efficiency, we are now developing a software tool to support the proposed process, as many of the steps can, for the most part, be automated. This tool should improve POAM efficiency so that it can be used in real development projects, where efficiency considerations are often just as or more important than questions of effectiveness.

6. Conclusions

RE is now one of the most significant activities in software development. This importance derives from the fact that it is during requirements engineering that the problem to be solved is examined and the features that the future software system will implement are determined. The study of the problem to be solved depends, largely, on conceptual models. These models, such as data flow diagrams or class diagrams, can be used to represent information about the problem to be solved. However, these models have several shortcomings, such as computational bonds and the predetermination of a given design type.

With the objective of providing a solution to the above-mentioned shortcomings, a method called "problem-oriented analysis method" (POAM) has been proposed. This method uses a set of CMs, grouped in the "Generic Conceptual Model", which have a high representation capability and are suited for modelling a wide range of domains. Additionally, the computational bonds have been minimised or totally removed, which means that the modelling is free of any computational consideration that predetermines a given design type. Finally, from the GCM, it is possible to select the best development approach from all those now available and derive the conceptual models used by the above approach.

The validation carried out to check POAM effectiveness has shown that the proposed approach is effective and it even performs better than practitioners experienced in conceptual modelling. In particular, POAM avoids subjective selection based on the preferences and opinions of the conceptual model developers, making an objective selection of the best-suited conceptual model to represent a given problem.

References

Beringer, D., 1994. Limits of seamless in object oriented software development. In: Proceedings of the 13th International Conference on Technology of Object Oriented Languages and Systems (TOOLS), Versailles, France.

Bonfatti, F., Monari, P.D., 1994. Towards a general-purpose approach to object-oriented analysis. In: Proceedings of the International Symposium of Object Oriented Methodologies and Systems, Palermo, Italy.

Borgida, A., 1991. Knowledge representation, semantic modelling: similarities and differences. In: Kangasalo, H. (Ed.), Entity–Relationship Approach: The Core of Conceptual Modelling. Elsevier Science Publishers.

Blum, B.I., 1996. Beyond Programming to a New Era of Design. Oxford University Press.

Champeaux, D., Lea, D., Faure, P., 1993. Object Oriented System Development. Addison-Wesley, New York.

Coleman, D., Arnold, P., Bodoff, S., Dollin, C., 1994. Object-Oriented Development: The Fusion Method. Prentice-Hall.

Davis, A., Jordan, K., Nakajima, T., 1997. Elements Underlying the Specification of Requirements. In: Annals of Software Engineering. Balzer Engineering Publishers.

Davis, A.M., 1993. Software Requirements: Objects, Functions and States. Prentice-Hall International.

Dieste, O., 2003. Analysis method for multi-paradigm conceptual modelling. PhD dissertation. Universidad Politécnica de Madrid.

Glass, R.L., Vessey, I., 1995. Contemporary Application—Domain Taxonomies. IEEE Software 12 (4).

Henderson-Sellers, B., Edwards, J., 1990. The object oriented systems life cycle. Communications of the ACM 33 (9).

Høydalsvik, G.M., Sindre, G., 1993. On the purpose of object oriented analysis. In: Proceedings of the Conference on Object Oriented Programming, Systems, Languages and Applications, New York, USA.

Jalote, P., 1997. An Integrated Approach to Software Engineering. Springer-Verlag.

Juristo, N., Moreno, A.M., 2000. Introductory paper: reflections on conceptual modelling. Data and Knowledge Engineering 22 (2).

Kaindl, H., 1999. Difficulties in the transition from OO analysis to design. IEEE Software 16 (5).

Loucopoulos, P., Karakostas, V., 1995. Systems Requirements Engineering. McGraw-Hill, Berkshire.

McGinnes, S., 1992. How objective is object-oriented analysis? In: Proceedings of the CAISE'92 Advanced Information Systems Engineering.

McGregor, J.D., Korson, T., 1990. Object oriented design. Communications of the ACM 33 (9).

Motschnig-Pitrik, R., 1993. The semantics of parts versus aggregates in data/knowledge modelling. In: Proceedings of the CAiSE'93. In: Lecture Notes in Computer Science, vol. 685. Springer-Verlag, Paris, France.

Mylopoulos, J., Borgida, A., Yu, E., 1997. Representing software engineering knowledge. Automated Software Engineering 4 (3).

Mylopoulos, J., Chung, L., Yu, E., 1999. From object-oriented to goal-oriented requirements analysis. Communications of the ACM 42 (1).

Northrop, L.M., 1997. Object-oriented development. In: Software Engineering. IEEE Computer Society Press, Los Alamitos, USA.

Novak, D., Gowin, D.B., 1984. Learning to learn. Cambridge University Press.

Schreiber, G., Akkermans, H., Anjewierden, A., de Hoog, R., Shadbolt, N., Van de Velde, W., Wielinga, B., 1999. Knowledge Engineering and Management: The CommonKADS Methodology. MIT Press.

SWEBOK, 2000. Guide to the Software Engineering Body of Knowledge. http://www.swebok.org.

Vessey, I., 1991. A theory-based analysis of the graph versus tables literature. Decision Sciences 22.

Webster, D.E, 1988. Mapping the design information representation terrain. IEEE Computer 21 (12).

Wieringa, R., 1991. Object-oriented analysis, structured analysis, and Jackson system development. In: van Assche, F., Moulin, B., Rolland, C. (Eds.), Proceedings of the IFIP WG8.1 Working Conference on the Object-Oriented Approach in Information Systems. North-Holland.

Zave, P., 1990. A comparison of the major approaches to software specification and design. In: Thayer, R.H., Dorfman, M. (Eds.), System and Software Requirements Engineering. IEEE Computer Society Press.