Handbook of Software Engineering and Knowledge Engineering
Vol. 0, No. 0 (2000) 000-000

CONCEPTUAL MODELLING IN SOFTWARE ENGINEERING AND KNOWLEDGE ENGINEERING: CONCEPTS, TECHNIQUES AND TRENDS

OSCAR DIESTE[1], NATALIA JURISTO[1], ANA M. MORENO[1], JUAN PAZOS[1], ALMUDENA SIERRA[2]

[1] *Facultad de Informática*
Universidad Politécnica de Madrid
Campus de Montegancedo, 28660-Boadilla del Monte, Madrid (Spain)
{odieste, natalia, ammoreno, jpazos}@fi.upm.es

[2] *Escuela Superior de Ciencias Experimentales y Tecnología*
Universidad Rey Juan Carlos
C/ Tulipán s/n, 28933-Mostoles, Madrid (Spain)
asierra@escet.urjc.es

Conceptual modelling is a crucial software development activity for both Software Engineering and Knowledge Engineering. Each discipline, however, has developed its own techniques for conceptual modelling, and there is no agreement about a common set of techniques that can be used in both disciplines. This chapter will describe such techniques, paying special attention to the more recent and innovative ones, as well as to the concepts shared by the techniques used in the two disciplines.

The chapter will, therefore, outline the field of conceptual modelling within these two disciplines. Although the situation in the field is satisfactory, as can be inferred from the review conducted, there is still a lot of work to be done. Indeed, a series of shortcomings besetting the different techniques will be identified and an alternative perspective will be described, which points to a way of quieting such objections.

Keywords: Conceptual modelling, software engineering, knowledge engineering

1. Introduction

The software development process is a kind of problem-solving process. Problem, here, means a context, environment or situation where a software system shall be developed and operated. For a successful software system development, Software Engineers (and Knowledge Engineers) shall understand all problem components, relations, rules, constraints, etc. Such an understanding is a hard and time-consuming process, which requires specialised tools for being

performed. These tools, which allow the Software/Knowledge Engineer to understand the problem to solve, are known as conceptual models (CMs).

The process of creating CMs in software development is generally referred to as conceptual modelling, although it may be given other pet names depending on the actual discipline in which it is performed; for example, **problem analysis** in Software Engineering (SE) [1] or **conceptualisation** in Knowledge Engineering (KE) [2] (there are specific materials about Knowledge Engineering in "*The Knowledge Modelling Paradigm in Knowledge Engineering*", by E. Motta, in this same volume).

The design of CMs is a crucial activity in traditional and intelligent software development. CMs are essential for:

- Making real-world concepts and relationships tangible [3].
- Recording parts of reality that are important for performing the task in question, and downgrading other elements that are insignificant [4].
- Supporting communication among the various "stakeholders" (customers, users, developers, testers, etc.) [5].
- Detecting missing information, and errors or misinterpretations, before going ahead with system construction [6].
- Providing an orientation on how the software should meet a need [7].
- Providing a specification of the behaviour of the system under construction [8].

Taking into account the above citations, it can be said that CMs are critical in the problem identification and solution proposal activities of any development project. As software systems become more complex and the problem domain moves further away from knowledge familiar to developers, conceptual modelling is gaining in importance. The reason is that modelling acts as the starting point for understanding and, thus, being able to solve customer and/or user problems. This is clear from the work of several researchers in the disciplines of both SE and KE, who claim that proper conceptual modelling is crucial for the future development of software. So, for example, the papers by McGregor [9], Bonffati [10] or Høydalsvik [11] concerning SE stress how important conceptual modelling is in ensuring that CMs faithfully represent the problem to be solved in the user domain. Similarly, researchers in the field of KE, like Schreiber [6], Hoppenbrouwers [12] and Adelman [13], underscore the fact that the quality of the resulting expert system is critically dependent on the CM produced, because the CM contains the knowledge to be implemented in the future software system.

SE and KE have developed their own conceptual modelling techniques. Nevertheless, techniques in each discipline have been developed in isolation, with little or no relationship among them. Although interactions between SE and

KE are becoming stronger [14] [15], practitioners in each discipline do not know which techniques are available to solve problems in the other discipline.

Moreover, this lack of knowledge makes difficult an interchange of experience between SE and KE, and much more a possible integration of techniques from both disciplines into a common toolkit, which could be used for developing traditional and intelligent systems [16]. As will be discussed in this chapter, some common ideas and principles, which have gone beyond the boundaries of their discipline and influenced the CMs used in the other, can be identified.

In this chapter, the general state of conceptual modelling in SE and KE will be reviewed, with the following objectives:

- Systematically examine the state-of-the-practice of conceptual modelling in SE and KE.
- Show the relationship between the ideas and principles used by different modelling techniques in both disciplines.
- Identify common principles and concepts in both SE and KE disciplines.
- Discuss how well adapted conceptual modelling and the existing techniques are to the functions that they should perform and the goals they should achieve in the development process.

For this purpose, this chapter will be structured as follows. Section 2 will discuss the concept of **model** within software development and will go on to stress the usefulness of the different sorts of models, typifying a special type of models called **conceptual models**, which will be specifically addressed in section 3. Sections 4 and 5, introduce the major models in SE and KE, respectively. Once having explained such models, section 6 discusses the interchange of some principles and ideas between the models of the two disciplines. Section 7 then analyses the models described from the viewpoint of how good they are as CMs, according to the definition given in section 3. It will be concluded that the existing models generally fail to attain the established goals, and these shortcomings will be described. Finally, an approach to the conceptual modelling process will be proposed in section 8, which could possibly help to solve the problems discussed in the preceding section.

2. Types of Models in Software Development

The term *model* is extremely polysemic in colloquial speech. As a representative sample of the diversity of meanings of the term model, take look at the definitions given by Webster's dictionary [17]:

1. A small but exact copy of something (for instance, a ship model).

2. A pattern or figure of something to be made (for instance, clay models for a statue).
3. A description or analogy used to help visualize something that cannot be directly observed (for instance, a model of the atom).
4. A system of postulates, data and inferences used to describe mathematically an object or state of affairs.
5. A theoretical projection of a possible or imaginary system.

Each of the above definitions, and probably any other possible suggestion, refers to a particular purpose of models, that is, what they are useful for, what results their application will yield, etc., answering the question "what do we want to build a model for?" Now, let's take a look at the circumstances surrounding software development and the use to which models are put during development.

In the case of software development, there are two, clearly distinct reasons for using models. Firstly, models are used *to describe the problem raised by the user* and to be solved by the software system. Secondly, models are used *to describe what the software system* that solves this problem *will be like*. This means that the term *model* is used twice during software construction, for two different purposes [18]:

- **Descriptive or conceptual model**. The CM describes an existing part of the world, and is the output of the conceptual modelling activity. The CM matches Webster's definitions (3) and (4); that is, a model is derived from reality for the purpose of gaining an understanding of such reality.
- **Prescriptive or computational model** (CpM). The CpM is the definition of a software product, and is the output of requirements specification and design activities. CpM matches Webster definitions (1) and (2); that is, a reality, the software system, is built from the model.

Blum [19] gives a very clear view of this distinction, indicating that software development can be seen as two successive and separate moments. Figure 1 shows Blum's approach (the box in the figure symbolise the whole development process, and arrows the flow of activities):

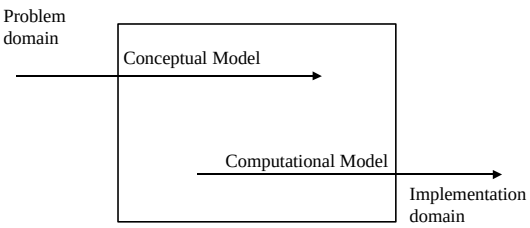

Figure 1. Software development process from the viewpoint of the types of models used

- A first problem-oriented phase. This is the phase where CMs are used to record the important concepts of the problem domain.
- A second system-oriented phase. This is the phase where CpMs are used to univocally determine the structure and functionality of the software system for implementation.

This chapter focuses on the first group of models, that is, on CMs. In the following section it will be described how this term emerged and how it is gaining in importance within the development process.

3. Conceptual Models in Software Development

The term CM originally emerged in the field of DataBases. CMs were used to represent data and relations, which were to be managed by an information system, irrespective of any implementation feature [20]. The scope of the term CM has gradually broadened since the approval of the ISO conceptual modelling standard [21], where its goal was to represent the *domain of discourse*. The domain of discourse is the set of data involved in the problem to be solved and the operations that affect the above data. In this context, only the operations that represented domain rules or, in other words, specific integrity constraints of the problem to be solved were represented in the CM.

CMs are used in the field of SE for more than is acknowledged by the above definition. The following citations show how CMs are considered in SE:

- Describe the universe of discourse in the language and in the way of thinking of the domain experts and users [22].
- Formally define aspects of the physical and social world around us for the purposes of understanding and communication [23].
- Help requirements engineers understand the domain [24].

Generally, the meaning of CM in SE is representation of the problem domain performed for the purpose of understanding and communication between developers and users.

CMs are also used in KE, where their goal is to model the expert knowledge without referring to any implementation mechanism. Newell [25] termed the level of abstraction at which CMs are located as *knowledge level*. Clancey [26] said that the knowledge level “accounts for behaviour in terms of interaction between agents and their environment”. Importance of CMs in KE is crucial, because they facilitate the understanding of the problem, and represent the knowledge needed by the software system to solve this problem [6] [27].

Despite their slightly different meanings in each discipline, the main characteristics of any CM are *description* and *understanding*; that is, the CM can be used by developers to:

1. Understand the needs raised by users.
2. Reach agreement with users on the scope of the system and how it is to be built.
3. Use the information represented in the model as a basis for building a traditional or intelligent software system to meet this need.

4. Conceptual Models in Software Engineering

Conceptual modelling takes a pre-eminent place in the discipline of SE. This has heightened in recent years, as the importance attached to the software development requirements specification phase has grown [28].

A wide variety of markedly different CMs has been defined in the discipline of SE. Firstly, the underlying *ontology* of all these CMs is very diverse. Ontology means the type of problem domain concepts that each CM is capable of representing [29]. In this respect, the CMs in SE range from models like the state transition diagram [1], which can represent only changes of state over time, to models like KAOS [30], which can represent a huge amount of both static (entities, relations, etc.) and dynamic (goals, processes, etc.) aspects of the problem domain.

Secondly, the intermediate representations, also known as *builders*, used by the different CMs, that is, the set of notations and symbols used by each CM to describe a given domain, is also very diverse. There are graphic-type models, such as are used by the object diagrams [31][32][33]; languages, such as are used by TELOS [34]; or models that combine graphic and informal representations, like the DFD [35], for which process specifications have to be created, normally using natural language text.

Taking into account the diversity of the approaches to conceptual modelling in SE, this section will be divided into two parts to assure a clearer discussion of the different models.

The first part will refer to what can be termed “classical” CMs. The oldest and most widely used models in SE, like SADT [36], DFD [35], object diagrams [31][32][33] or state transition diagrams [1], were termed classical models. The main characteristic of this type of models is how limited their ontologies are, that is, the fact that they can be used to represent a relatively small number of concepts about the problem domain. Individual classical CMs will not be discussed, as they are very well known. Instead, a description of groups of models will be given, classified according to the similarity of their respective ontologies.

The second part of this section will be devoted to a set of models developed during the 90s, which, unlike the classical CMs are characterised by having powerful ontologies that can be used to describe novel aspects, like goals, agents, etc. This set of models, called "advanced" CMs, include TELOS [34], KAOS [30], EM [37] and ***i**** [38]. They will be discussed according to the strategy they have used to extend their capability of representation beyond the classical CMs.

There are no good reviews of the CMs which will be discussed. Therefore, readers are referred to the original references for more details. However, the papers by Wieringa [39], which investigates the structured, that is, functional, and object-oriented methods, and by Davis [1], which explains at length the classical approaches to conceptual modelling in SE and offers several illustrative examples, are recommended reading.

4.1. Classical conceptual models in SE

As discussed earlier, a wide variety of CMs have been used in the discipline of SE, most of which belong to the set termed classical CMs. The best way of examining the models belonging to this set is to group them according to their orientation, as Davis [1] did, or using the terminology of this chapter, depending on the similarity of their ontologies. The groups defined thus are as follows:

- *Functional* CMs, whose paradigm is the DFD [35]. This kind of models describes the transformations of the data used in the problem domain. The transformations are described by means of the *process* concept, which receives an input data-set and generates an output data-set.
- *Object-oriented* CMs, whose utmost representatives are object diagrams [31][32][33] and use cases [40]. However, use cases should, strictly speaking, be considered as functional CMs. Object-oriented CMs describe the objects or object classes and the interrelations between them in the problem domain.
- *State-oriented* CMs, including state transition diagrams [1]. These CMs describe the configuration of objects, facts, phenomena, etc. on the problem domain, and the changes produced in this configuration along time.

Classical CMs have several advantages and serious drawbacks. This kind of models allows the modeller to express many things about the problem to solve, but they allow expressing only a partial view of the problem domain, that is, each kind of CMs is specialised in representing a set of facts about the problem domain. For example, let's consider the (canonical) problem of organising a conference (this example will be used in further sections). We can model this problem using the utmost representatives of each above-mentioned group, that

is, DFD, object diagrams and state-transition diagrams, as it is shown in Figures 2, 3 and 4, respectively.

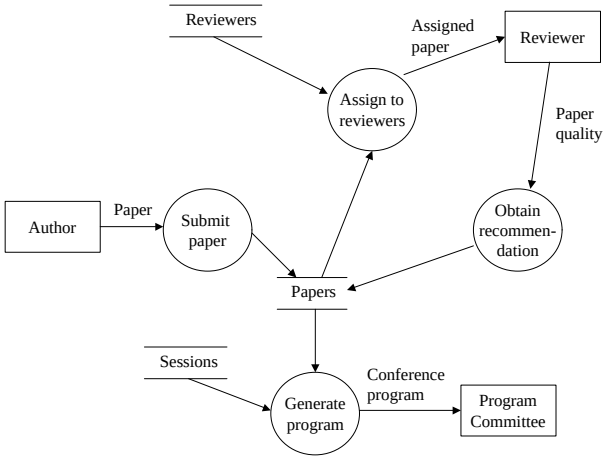

Figure 2. DFD for the conference problem.

Functional models (in the case of Figure 2, DFD) show the processes that transform input data into output data (for example, the generation of the "Conference program" using the "Papers" and "Sessions" data"). Nevertheless, this kind of models cannot express any other information, as the data structure, or the ordered sequence of events that occur in the conference organisation.

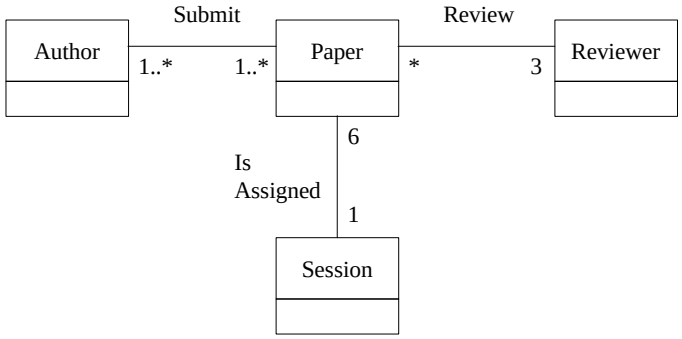

Figure 3. Object diagram for the conference problem.

For representing data, entity-relationship [41], object-role [42] or object-oriented CMs can be used (although it is not very strict, several authors, as Davis [1], consider that entity-relationship and object-role models belong to the object-oriented CMs group). Figure 3 shows an object-oriented CM built using

the UML notation. It is clear that, using object diagrams, it is impossible to express something like a data transformation (for doing this, it is needed to include *methods* in the object diagram, which is a controversial topic due to such a introduction makes the model harder to understand). Object-oriented cannot express time-ordered sequences of events, as DFD cannot either.

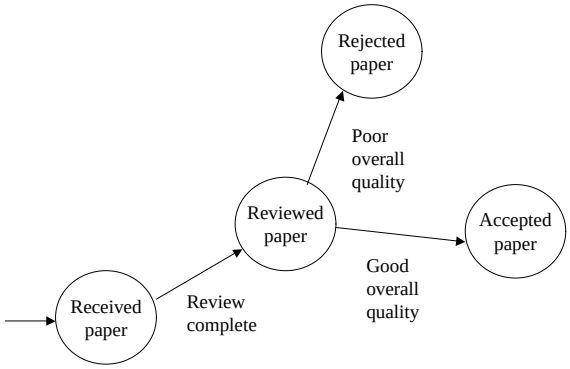

Figure 4. State-transition diagram for the conference problem.

The sequence of events in time can be expressed by means of a state transition diagram, as show in figure 4. Nevertheless, this model provides a partial view of the problem again, because data structure or data transformations are hidden behind the states and transitions of the diagram. In short, classical CM, as it was shown in the examples, possess enough power to express situations they are intended to, but it is impossible to improve their representation capabilities far from their frame of validity. For obtaining better results in the modelling activity, it was needed to define new CMs, which are introduced in next section.

4.2. Advanced Conceptual Models in Software Engineering

As specified earlier, the characteristic of a classical CM is that it can describe a given set of concepts in the client and/or user domain. For example, a DFD can be used to describe only the transformations that take place in the domain, whereas an object diagram can be used to describe the types of objects, as well as their interrelations.

Such a process of conceptual modelling has two drawbacks. Firstly, only a small set of domain concepts can be described, where the limit is established by each individual CMs capability of representation. Taking all the CMs used in SE as a whole, for example, the most important concepts that they can be used to describe are: inputs, outputs, processes, agents, data, objects, relations, states and transitions.

Secondly, as a result of the above, more than one CM, each of which describes partial aspects of the domain, has to be used at the same time to faithfully describe the problem domain. For example, structured approaches are characterised by using three CMs together [43]: a process model, built using DFD; a data model, usually described by means of a entity-relationship model, and a control model, based on the state transition diagram. Although the structured approach is a paradigmatic example, the joint use of different CMs is also a common feature of other approaches, like object-oriented methods, for example. Thus, object diagrams, DFD and state transition diagrams are all used together in OMT [31], whereas UML employs use case diagrams, object diagrams and other type of representations like interaction diagrams [44]. Nevertheless, CMs can be used together when it is not possible joining them in just one CM. For example, light is described in two ways in physics: as particles or as waves. Both representations are incompatible, that is, one of them excludes the other. Therefore, it is not possible to join them in just one single model, but they both are needed for understanding the essence and behaviour of light.

The advanced CMs emerged as a means of surmounting the above-mentioned drawbacks. For this purpose, these models include richer and more powerful mechanisms of representation. Advanced CMs employ two different strategies to improve their capability of representation:

- Enriching the CM ontology, that is, increasing the number of concepts covered by the above models. This means that they can represent novel concepts, which the classical CMs could not represent, like goals, dependencies, constraints, rules, etc.
- Defining extendible ontologies, that is, allowing the user of the CM to define the type of concepts to be represented. This means that the CMs can be extended and adapted depending on the domain and problems in question.

These two improvements do not operate simultaneously, that is, advanced CMs use only one of the two strategies. Indeed, the possibility of extending and adapting concepts that a CM can represent makes it possible to simulate any other model type, irrespective of the number of concepts covered. The two strategies and the advanced CMs that implement them are described below.

4.2.1. Enriching the ontology

There are two possible mechanisms for enriching the ontology underlying a CM. The first is to broaden the concepts that can be represented by a CM, defining special-purpose builders that represent the above concepts. The second is to define a *meta-model*, that is, a generic structure on the basis of which to build

individual CMs. The following two sections describe each of the above mechanisms.

4.2.1.1. Definition of special-purpose builders

The first of the above-mentioned mechanisms has been the most commonly used for defining new CMs. This mechanism entails increasing the number of concepts that can be described by a model by adding new builders to the model. Each new builder represents a given concept. Also, semantics, that is, an interpretation, must be defined for each new builder by means of which its meaning can be unambiguously understood.

By way of an example, suppose that we want to represent processes in an entity-relationship diagram. The easiest, albeit not a very strict, solution would be as follows.

1. Define a new builder, called “process”.
2. Define a graphic notation for the above builder. For example, a pentagon could symbolise the “process” builder.
3. Define semantics for the builder. The semantics could be similar to the process semantics in a DFD. The semantics would involve the “process” transforming the data. The necessary input data are obtained from the entities or model attributes. The output data are also entities or attributes.

Figure 5 shows an example of the entity-relationship model, modified as described above, which shows how the process “Assign session to paper” would be described in a conference or workshop domain. In this figure, the static perspective has been described by means of the classical symbols of entity-relationship models. Using this diagrammatic convention, we can express that each session includes several papers, and each paper is written by several authors and has several reviewers. The new builder, symbolised by a pentagon, lets us to express a dynamic perspective, that is, the process of assigning a paper to a session once we assure it has the required quality.

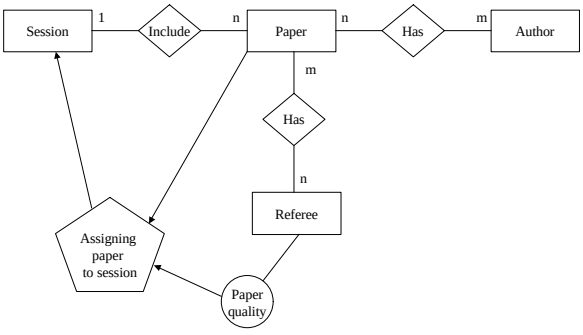

Figure 5. Entity-relationship model modified to represent processes.

Introduction of new concepts and builders has been the mechanism used in the ***i**** advanced CM. ***i**** (which stands for Distributed Intentionally) was originally defined in [38]. This CM can be used to describe how different actors in a domain depend on others to achieve their objectives, as well as to specify the internal motivations by which they are guided. The actors are obviously not people, they are entities that operate autonomously and have their own internal dynamics. ***i**** is composed of two different and complementary sub-models: the Strategic Dependency Model (SDM) and the Strategic Rationale Model (SRM).

SDM can be used to describe a set of *dependencies* among actors. When there is a dependency between two actors, one depends on the other to achieve a goal, perform a task or get a resource. The actor that provides the means for satisfying the dependency is termed *depender*. The actor that is benefited when the dependency is satisfied (or harmed, otherwise) is termed *dependee*.

i* makes a distinction between four types of dependencies: *goal dependency*, where one actor depends on another to reach a given state or assure that a given condition is met [45]; *task dependency*, where one actor depends on another to be able to perform a given activity; *resource dependency*, where one actor depends on another to get any physical utility, like a tool, or logical utility, like information of some kind; and, finally, *soft-goal dependency*, which is similar to a goal dependency, except that there is no "a priori" procedure for determining whether or not the goal is attained.

For example, soft-goal dependency arises when a conference programme committee asks referees to select the best papers. "Select the best papers" is a goal that can be attained to a greater or lesser extent rather than in absolute terms. Figure 6 shows how the above example would be represented in ***i****, using its associated graphic notation.

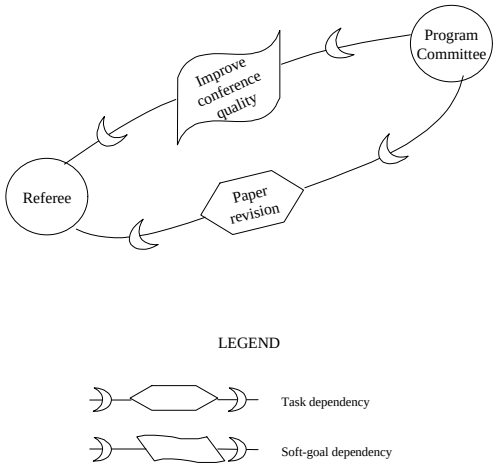

Figure 6. Example of the use of soft goals in ***i****.

SRM can be used to describe the reasoning or internal motivation of each actor. The above description is carried out in relation to the dependencies of each actor on other actors, albeit from the private viewpoint. This means that the description is made within the internal province of each actor, which is called *actor boundary* and is not shared with the other actors. The SDM and SRM are, therefore, strongly interrelated: SDM describes the goals shared by a set of actors, whereas SRM describes which mechanisms are brought into play by each actor to achieve its own goals.

4.2.1.2. Definition of meta-models

The second mechanism for enriching the ontology of a CM involves explicitly defining a meta-model. A meta-model is the description of the concepts that can be represented by a CM. Figure 7, for example, presents a simple meta-model for the DFD.

As shown in Figure 7, the meta-model is (usually) described similarly to the entity-relationship diagram. This sort of description can be used to specify what the model components (diagram entities), and the relations between these builders (relations between diagram entities) are. The meta-model shown in Figure 7 briefly indicates that data flow diagrams are composed of “processes”, “data stores”, “external entities” and “data flows”. Additionally, it also indicates, using the relations “input” and “output”, that the processes, stores and external entities generate (output relation) and receive (input relation) data flows. Nevertheless, Figure 7 does not include all the aspects for describing DFDs, such as, for example, the permitted connections (an external entity cannot be directly linked with a data store) or the decomposition of processes into subprocesses.

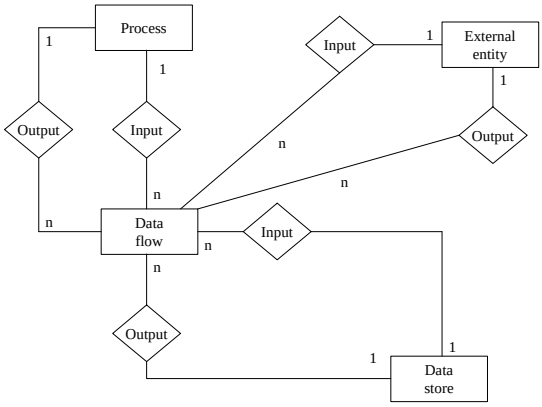

Figure 7. DFD meta-model.

If a meta-model is defined, it is possible to distinguish between the levels at which modelling takes place, which are used unconsciously in practice. These are the *meta-level*, which defines the concepts that a given CM is capable of recording; the *domain-level*, at which the concepts of the meta-level are instantiated using specific information about a domain; and, finally, the *instance-level*, at which a distinction is made between each of the possible domain-level occurrences [46].

These three levels are shown graphically in Figure 8, using a DFD. As above, this DFD represents the organisation of a hypothetical conference. In the upper side of the figure, the META-level symbols and rules appear. These symbols and rules are the highest level of model abstraction, and they state what facts, at the domain level, the model can express. The DOMAIN-level, in the middle of figure 8, is what we usually know as a "model", that is, a meaningful combination of symbols with meaning, which follows the rules defined at the META-level. The INSTANCE-level, in the bottom side, is the lowest level of abstraction and it is never considered in practical modelling.

Two advanced CMs that explicitly use the meta-model concept are: KAOS (Knowledge Acquisition in autOmated Specification), originally defined by Lamsweerde [30] and described at length by Dardenne [47], and EM (Enterprise Modelling), described by Kirikova [37]. Figure 9 presents a simplification of the KAOS meta-model, which is described with more or less evident adaptations of entity-relationship diagrams. Although it introduces different concepts, the EM meta-model is also defined as a complex network of interrelations similar to the KAOS meta-model.

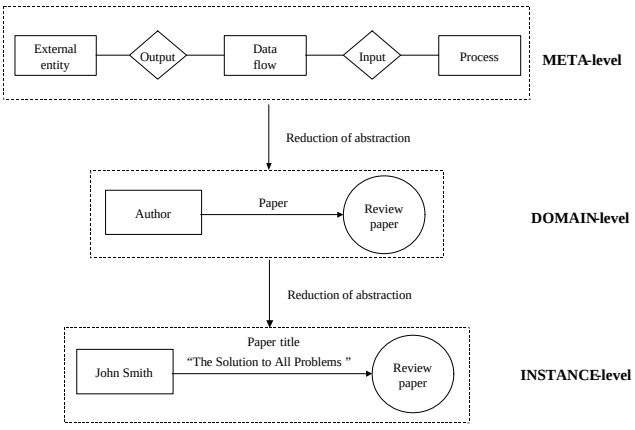

Figure 8. Meta, domain and instance levels for the conference problem using DFD.

KAOS and EM can be used to represent both static and dynamic aspects of a domain. This is achieved by explicitly including builders for entities and

relations in the meta-model, for example, apart from builders for processes or activities. Additionally, both models introduce novel concepts, like goal, agent or constraint, which did not exist in the classical models and are very useful in the CM. This becomes perfectly clear from comparing the meta-model shown in Figure 9 with the meta-model for the data flow diagram presented in Figure 7.

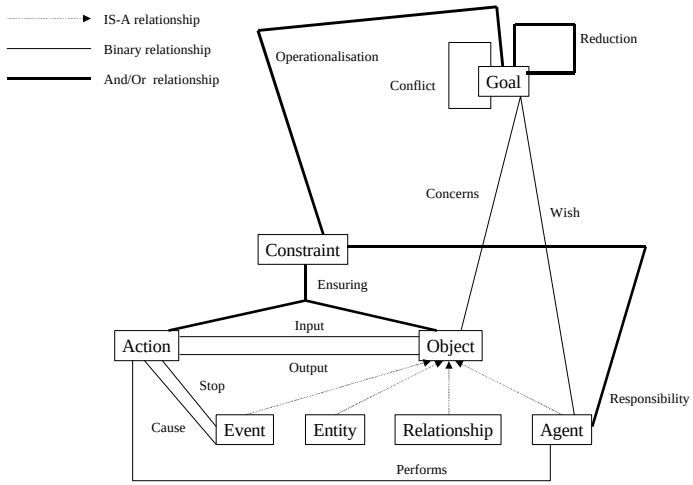

Figure 9. KAOS meta-model.

The notation used by the two CMs is complex, and a complex algebraic definition can even be provided for KAOS. Therefore, it is a better idea to illustrate the use of these CMs with an example rather than giving the notation. The example again models a conference using KAOS. This model is illustrated in Figure 10.

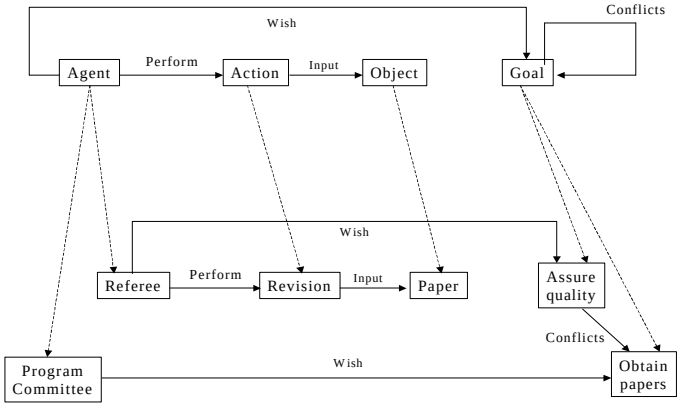

Figure 10. Partial model of a hypothetical conference in KAOS.

For the sake of clarity, Figure 10 shows the meta-level, plus the domain-level, making it easier to interpret. A more extensive example using KAOS is given in [48], where it is also shown how this CM can co-exist with i^*.

4.2.2. Extendible ontologies

The use of fixed ontologies, even especially rich ones, always has the drawback of them not being able to encompass the wealth of concepts and shades of meaning that actually exist in the real world. For example, suppose that we need to model the control of an industrial furnace. Suppose, also, that control involves keeping the furnace within a temperature range, above which there is a danger of explosion. Again using KAOS, the set of concepts discussed could be modelled as indicated in Figure 11.

There is no doubt that an important concept in the above problem definition is *risk*. This is due to the fact that an explosion of the furnace, for example, could lead to financial losses or even to the loss of human lives. However, as KAOS does not explicitly account for the concept of risk, there are only two possible actions during modelling:

1. Obviate the above concept, as we did in Figure 11.
2. Ascribe the concept of risk to a pre-existing model builder. It would be ascribed on the basis of some relationship of similarity between the concept to be represented and the meaning of the model builder.

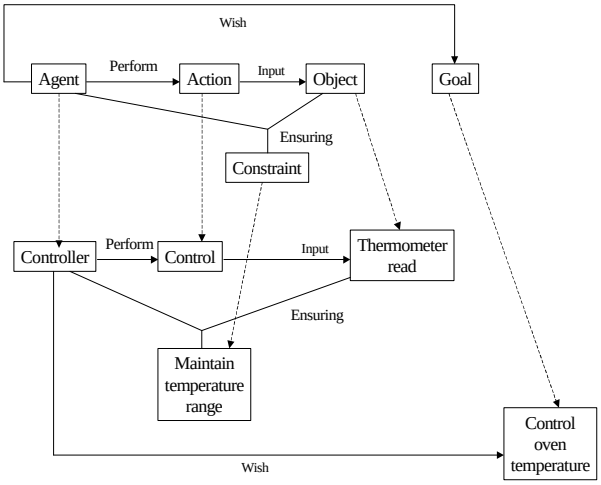

Figure 11. Model of an industrial furnace in KAOS.

Generally, most classical or advanced models that have fixed ontologies take action (2). Ascribing domain concepts to model-specific builders involves

creating an injection rather than a bijection from the real world to the CM. Some shades of the real-world meaning are inevitably lost in this way, as shown in Figure 12.

This loss of knowledge is not due to an abstraction, as one might think at first glance. Abstraction means that two (or more) different objects, facts, phenomena, etc. of real world are considered in the model as similar, due to the important properties to the modeller are perceived as similar. Nevertheless, when there exists objects, facts, phenomena, etc. in the real world that the modeller cannot assign univocally to a concept in the model, but these objects, are important to the problem in question, modeller should decide how register such information. This decision is a trade-off among several alternatives, any of them losing some information (perhaps important) about the real world. The trade-off usually ends assigning the object, fact, phenomena, etc. to the more similar concept available in the model.

One mechanism for solving the above-mentioned problem is to make provision for defining the type of builders to be used for each individual problem. This means that the CM is "reinvented" in each modelling process and adapted to the problem in question.

TELOS (from the Greek τελος, which means purpose, end) [34] falls within this group. TELOS is a model whose main builder is the class. As in object orientation, a class is an abstraction by means of which to group a set of real-world elements under one name.

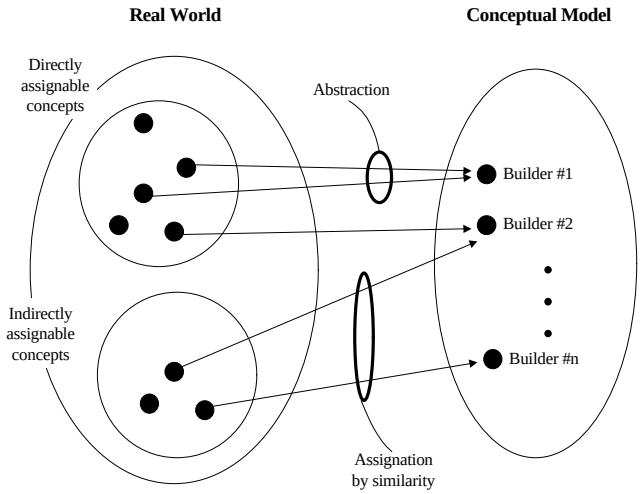

Figure 12. Modelling using a CM having a fixed ontology.

TELOS permits the classical object-oriented operations on classes: aggregation, classification and inheritance. Additionally, rules and constraints,

which represent invariants, preconditions, postconditions or deductive rules that are applied to the instances of different classes, can also be specified in predicate logic.

Extendibility is determined by the possibility of building a specific model, using predefined classes of TELOS, including the predefined class `MetaClass`. The mechanism used is similar to the creation of a specialisation hierarchy. The root of the hierarchy is the class `MetaClass,` whereas the terminal classes provide the concepts for modelling. An example of the above hierarchy is illustrated in Figure 13.

Unlike a specialisation hierarchy, however, the child classes are instances rather than a refinement of the parent classes. This is an important difference, as the child classes can be defined by entering the constraints on parent class attributes. Defined thus, classes can be used as first-order objects to model individual problems.

Any sort of CM can be defined by means of a mechanism for creating extendible ontologies. A definition of SADT [36], for example, is given in [34], and TELOS was used in practice to build some models within the ESPRIT-II NATURE project [49].

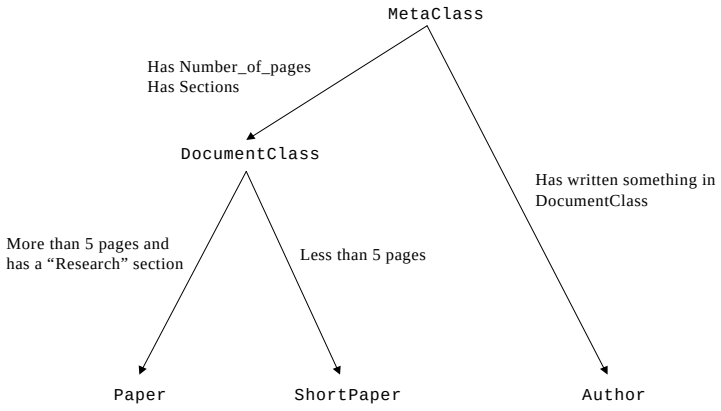

Figure 13. Definition of a new ontology in TELOS.

5. Conceptual Models in Knowledge Engineering

As discussed above, conceptual modelling plays just as an important role in KE as it does in SE, since it is the activity by means of which to define expert problem-solving behaviour and is the essential starting point for entering the above behaviour into a knowledge-based system (KBS).

There are several approaches to modelling knowledge: Problem-Solving Methods (PSM), ontologies and Knowledge-Based Systems (KBS) development

methodologies. In this chapter, all we address are methodologies, since PSMs cannot be considered as techniques for building CM, the main goal of this chapter. Additionally, the CMs for ontologies can be regarded as part of the CMs for KBS, whose construction is covered by the development methodologies.

KBS development methodologies explicitly address the definition of CMs [50] [51] [52]. Unlike SE, where, as discussed in the preceding section, the most commonly used CMs are relatively simple, the CMs in KE are fairly rich and complex, recording a huge number of concepts about the problem domain. All the methodologies described divide the CM into three different representation levels, aimed at proposing an order in which concepts should be acquired. These CM representations, or knowledge levels, are referred to slightly differently depending on the methodology in question. We will use the following terminology in this chapter:

- **Strategy models**. Strategy models identify and describe the task performed by the expert in order to carry out a given job. Additionally, they identify the (sub)tasks resulting from the decomposition of each main task, as well as the order in which the above (sub)tasks have to be performed, when they have to be executed and under what conditions. Although, owing to the slight differences between the methodologies, it is risky to generalise, the goal of this sort of models could be said to be to describe the task hierarchy, which, at a given level of decomposition, is simple enough to be implemented by a PSM or an *ad hoc* method, built especially for the case in question. This strategic level is called *task level* in other papers.
- **Reasoning models**. Reasoning models are used to represent the reasoning that the system is to carry out to perform the (sub)tasks represented in the task model, specifying which sorts of domain knowledge are required for each reasoning step and what knowledge is gained as a result of the above reasoning. This level is also sometimes termed *inference level.*
- **Domain models**. Domain models represent the domain structure, which has to be known to make inferences and execute tasks. Domain models are the static part of the CM, whereas the strategy and reasoning levels make up the dynamic part of the model.

In the following, we will analyse the CMs used by the KE methodologies, focusing attention on how each knowledge level of the expertise model is described. We will actually describe the following modelling languages: CML [53] [54], linked to CommonKADS [50], KARL [55] [56], linked to MIKE [52], and MODEL [57], linked to PROTÉGÉ-II [51]. These languages have special builders for each knowledge level.

5.1. Domain level

All the above methodologies describe the structural aspects of the problem domain at the domain level. These methodologies structure this knowledge level around the concepts of classes of objects and relations, like SE object-oriented models do. However, the syntax and nomenclature varies from KE to SE and even from one KBS development methodology to another. Thus, for example, classes are called *concepts* in CML [53] [54], whereas they are termed *frames* in MODEL [57] and are referred to as *classes* only in MIKE [55] [56]. Each of these languages also presents other builders for modelling other aspects of reality. Although the aspects they aim to model are basically the same, there are, as for classes, some differences in the nomenclature and representation methods.

Apart from concepts and relations, CML has the following builders [6].

- Attributes that define concepts.
- Structures that are concepts that the modeller does not want to describe in detail at any given time.
- Expressions that can be used to define domain constraints and axioms. These expressions can be used to define set membership, set inclusion, relationships of order or equality between attribute values, etc. For example, one cannot be one's own supervisor.

Apart from classes and relations, MIKE also includes the following for defining the domain level [55].

- Attributes to define concepts.
- Formulas by means of which to define any constraints there are in the domain, like expressions in CML.

MODEL can be used to represent the domain concepts by means of a builder called frame [57]. A series of slots can be defined for each frame, which describe the attributes that describe the concept. Constraints on the values of the attributes can also be defined by means of a builder, called facet. The relations in MODEL must be defined as frame-type attributes; that is, a definition closer to the symbolic than the conceptual level.

Both CML and KARL [54] [55] have a graphic notation that is very similar to the one used by object languages, such as shown in Figure 14. Rectangles symbolise classes of object, and lines symbolise relations (inheritance relation is allowed, and it is symbolised by a triangle, as show in Figure 14).

Note that the three languages represent classes and their attributes, relations between them and constraints, although each has its particularities. Only CML has an additional element, structure, by means of which to indicate when a class has not yet been fully defined.

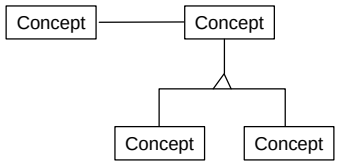

Figure 14. Graphic notation used by CML and KARL

5.2. Reasoning level

The static domain concepts are not enough to define how an expert behaves when solving problems, as they do not include the reasoning required to yield new facts, conclusions, etc., on the basis of the existing knowledge represented at the domain level. Therefore, the goal of the reasoning level is to explicitly define the reasoning to be pursued to solve the KBS target problem, that is, the inferences can be made using the domain knowledge and the knowledge roles that model inference input and output.

CML has two builders for modelling the reasoning level [6], which are as follows.

- The basic inferences, known as sources of knowledge. CML sets apart a group of special basic inferences: transfer functions. These inferences represent the operations of communication with the outside of the system.
- The metaclasses or roles that participate in the inferences as inference input or output.

MIKE has the same builders, but makes a distinction between several types of roles [56]:

- Views: supply an inference with domain knowledge
- Stores: model the data flow dependencies among inferences
- Terminators: are used to give the final results.

An example of the diagram of the two languages is given in Figure 15, where roles or metaclasses are symbolised as rectangles, and inferences or knowledge sources as ovals. The rationale behind this figure is to show that in CommonKADS and MIKE roles (note that terminology is different in both approaches) are the inputs required and outputs generated by the inference process.

The problem-solving method is composed of the inferences and roles, as well as the data flow dependencies between inferences, in both CommonKADS and MIKE [52]. At the conceptual level, these two languages define which

inferences and which knowledge roles are required to solve the problem. However, they do not specify how this reasoning is done. This means that they specify what PSM to use. However, a PSM is not detailed at the conceptual level, it is a design or computational model, as mentioned above.

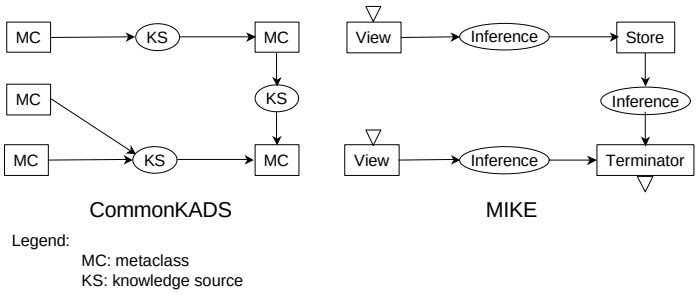

Figure 15. Structure of inferences in CML and KARL.

In PROTEGE-II, the reasoning level has a slightly different structure that in CommonKADS or MIKE. In these two methodologies, the reasoning level contains the allowed inferences. These inferences are related to domain knowledge, in one side, and to a PSM, in the other. In PROTEGE-II, there not exists the "inference" concept. The relationship between domain knowledge and PSM is implemented using the "mapping relations" concept [58], instead. A mapping relation is a translation that makes possible the PSM to use the knowledge of the domain level.

5.3. Strategy level

The basic inferences represented at the reasoning level along with their input and output knowledge are not enough to solve the KBS target problem. The basic inferences indicate the reasoning to be carried out. However, they make no reference to the control of this reasoning, that is, how the above basic inferences are to be sequenced to solve the problem. There are several courses of action and several inferences applicable at any one time in a somewhat complex domain, which means that they have to be controlled to arrive at the right solution.

The goal of the strategy level, therefore, is to identify and define the sequence of valid actions in the problem domain, controlling the number and type of inferences that are to be made.

The structure of the strategy level of the three languages is illustrated similarly using a decomposition tree [6] [58] [59]. The tasks that appear at the bottom of the tree are basic inferences in CML and KARL and mechanisms in MODEL, as shown in Figure 16. They are described at the reasoning level. This is all as far as task decomposition is concerned.

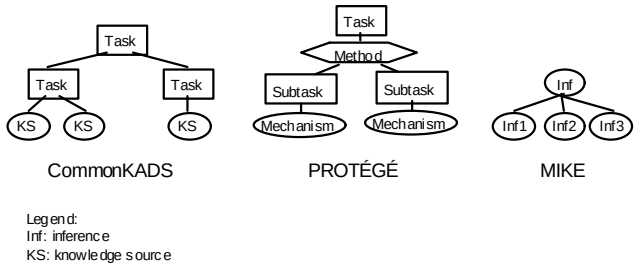

Figure 16. Decomposition of the task in each language.

As far as the sequence in which these basic inferences or mechanisms are executed is concerned, that is, task control, this is defined in all three languages using similar builders to 3rd generation programming languages (selections, repetitions, sequences and assignations), although each language has its own syntax.

5.4. Relationship between the knowledge levels

The languages described may appear to represent different facets of reality separately, but this is not the case. The elements or, better still, knowledge that is represented at the domain level is used to define the inferences. The roles of the reasoning level are covered by domain level concepts or classes. The strategy level is also related to the inference level, as the subtasks are no longer divided, they are basic inferences and are described at the reasoning level. This relationship is illustrated in Figure 17.

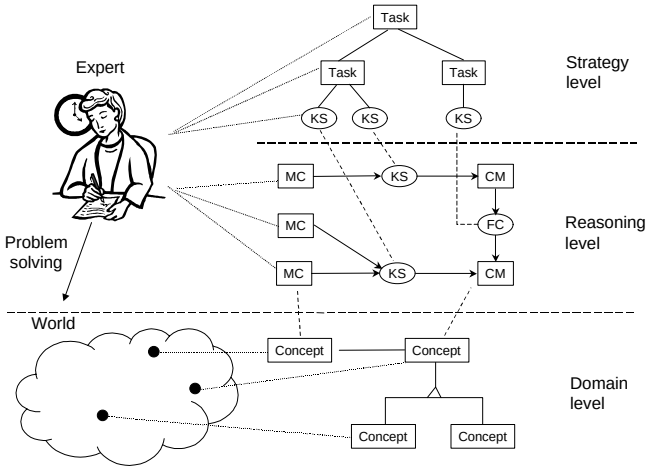

Figure 17. Interrelationship of the CM knowledge levels.

The above sections discussed the CMs used in the disciplines of SE and KE. In the following, we will discuss several issues that generally affect all the above-mentioned CMs. The common features of all the methods of conceptualisation will be discussed in section 6, and a series of problems pointed out by several researchers concerning the above methods of conceptualisation will be specified in section 7. This chapter will conclude by discussing an approach to conceptual modelling that can solve the above-mentioned problems in section 8.

6. Similarities between Software Engineering and Knowledge Engineering Conceptual Models

The classical and advanced models used in the disciplines of SE and KE were described above. Taken as a whole, the general impression we get is that the CMs used by the two disciplines have things in common. Indeed, as we said earlier, the disciplines of SE and KE have influenced each other, and some of the concepts created in one discipline have been absorbed to a greater or lesser extent by the other.

Generally, all the approaches described, except for the oldest methods used in SE, that is, function models and state-oriented models, can be said to have two common characteristics:

- As a general rule, the ontology underlying the set of formalisms and languages is based on the concepts of classes and relations. This is only logical considering that they have to represent the structure of the real world.
- Concepts like agent, goal, rule or constraint are used extensively. These concepts can be used to model many aspects of the problem domain.

These concepts are much more patent in some models than in others. For example, the object-oriented models explicitly include objects and relations as builders, as do TELOS and all the KE languages described. However, the representation formalism used by each individual model can make the above concepts more or less apparent. It is very easy to identify the objects and relations in TELOS, for example, and a bit more difficult in MODEL. As far as the concepts of goal, agent, rule or constraint are concerned, their use is confined to the KE languages and advanced SE models, and it is more difficult to identify the above concepts in the classical SE models. There are exceptions, however: the concept of agent or “actor” is used in use cases, whereas some constraints can be specified by means of pre- and post-conditions in some object-oriented approaches, as is advocated by Eiffel [33].

There are several reasons for the existence of the above-mentioned characteristics. The first is the need to represent the problem domain more objectively and in more detail to ease its understanding and make for a more efficient development process. The above need is felt in both SE and KE and is causing the CMs to evolve, gradually becoming more powerful and having greater capability of representation.

The second reason is the realisation that the CMs used in each discipline are insufficient and that it can be beneficial to include concepts from other disciplines. This is especially patent in SE, which has borrowed concepts from other disciplines for its CMs. Some have been added directly and without adaptation, as was the case of data models. Others, like the concepts of goal, agent or constraint, were adopted only partially.

The fact that there are influences and common characteristics indicates that all the disciplines are converging, as the use of common concepts implies that there is some uniformity between the methods of conceptual modelling. Nonetheless, it is clear that we still have not reached the point of developing CMs that can be shared by disciplines and we are even further away from single formalisms, which could be used simultaneously in SE and KE.

Despite the mutual influence of one method on another, the dearth of such "interchangeable" models indicates that any sort of total unification of conceptual modelling concepts between the disciplines of SE and KE is still a long way off. Indeed, each discipline is like an island, whose only means of communication are glass bottles thrown into the water.

Nonetheless, the reasons for this isolation are not to be found in the individual characteristics of each discipline, the particular problems they solve or the methods and tools they use. The reason underlying the diversity of conceptualisation methods is due to each one being linked to a given software development approach. This approach or computational paradigm restricts the possibility of using the conceptual modelling method in other settings apart from the one for which it was specifically designed. The following section addresses this idea in more detail.

7. Limitations of the Current Approach to Conceptual Modelling

The CMs now used in SE and KE are conditioned by computational constraints proper to given development approaches, that is, they are more like prescriptive or computational models (CpM) than CM, which, as mentioned above, are characterised by being user oriented.

The criticisms of the conceptualisation methods go in two directions. The first refers to 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. The second point refers to the association between CMs and specific approaches to software development. Here, the use of a given CM during the early phases of the development process limits the number of possible implementation alternatives and means that only the options that are compatible with the CM used originally are feasible. The above-mentioned problems are discussed in more detail below.

7.1. Computational orientation of the methods of conceptual modelling

Many of the CMs are oriented to providing a computational solution to the problem raised in the user domain. This orientation to the solution is necessitated by the fact that the representations of the different CMs include computer-related concepts. Several researchers have pointed out that conceptualisation methods suffer from this limitation, mainly in the object-oriented community, as the following:

- It is argued that object-oriented methods are a 'natural' representation of the world. Nevertheless, this idea is a dangerous over-simplification [60].
- Object-oriented analysis has several shortcomings, most important in being target-oriented rather than problem oriented [11].
- Object-oriented analysis techniques are strongly affected by implementative issues [10].

There are authors that extend the critics to other models, too. For example, M. Jackson argues that:

- DFD's are vague pictures suggesting what someone thinks might be the shape of a system to solve a problem, but they do not say what the problem is [61].
- There exists no theory of how a model relates to the real world [62].

Thus, for example, the data flow diagrams are clearly guided by functions, key components of structured software and, likewise, the models used in object-oriented analysis lead directly towards software developed by means of classes, objects, messages, polymorphism, etc., basic concepts of object-oriented software. On the other hand, the CMs used in KE are oriented to heuristic problem solving, proper to experts. The problem with including computational considerations within the CM is that the software or knowledge engineer is 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 variables relevant to the problem are known. Engineers thus run the risk of making the wrong decision

because they are not in possession of all the information. This has quality-related implications for the final software product [63].

7.2. Association between conceptual models and development approaches

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:

- The use of a CM during analysis defines nearly univocally how the design shall be done [64].
- Perhaps the most difficult aspect of problem analysis is avoiding software design [1].
- It is sometimes mistakenly believed that the structures produced during analysis will and should be carried through in design [65].
- The boundaries between analysis and design activities in the object-oriented model are fuzzy [66].
- The software system development approach is preconditioned by the CM used [67].

Owing to this limitation, for example, if data flow diagrams have been used to model the problem domain, it will almost certainly be necessary to use the structured method in later development phases; a method of object-oriented development will have to be used following upon an object-oriented analysis. Similarly, if the problem has been conceptualised in KE, it will be practically impossible to use a SE development method if it is discovered that the knowledge level does not call for a knowledge-based system.

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 very difficult to fill. This gap between CMs and CpMs is caused by the fact that each CM acts like a pair of glasses used by the engineer 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 this information is required by the CpM. 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. This situation has already been discussed by authors like Coleman [68], Champeaux [69] or Wieringa [70], who address the incompatibility between the CMs used in the structured approach and object-oriented CMs,

owing to the conceptual difference between the elements used in the two approaches.

Despite this limitation, some attempts have been made to derive object-oriented models from structured models [71] [72] [73]. However, the guidelines proposed are confined to mere heuristics, which are not applicable in all cases. Moreover, Henderson-Sellers [64] claims that the reverse transformation is impossible and even that any transformation of structured models to object-oriented models would be partial and indirect.

Additionally, these switches from a CM of one approach to a CpM of another paradigm, even if they were possible, are not usually feasible in practice in view of the time and cost constraints and size of projects nowadays. Mostly, the CMs used are those with which developers are familiar, called for by individual standards or even, as specified by Mylopoulos [74], 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.

Finally, the software system development approach can be said to be preconditioned from the very start, when the CMs are built. Moreover, excepting trivial problems, this precondition means that the development approach is chosen before the user need has been understood, which is the job of conceptual modelling.

7.3. An alternative solution

Due to all the above-mentioned conceptual modelling problems, it can be said that, on the one hand, CMs have to be brought closer to users, using a language that they can understand, which will improve validation. On the other, they need to include all the information required about the problem for developers to later address the software system that is to solve the problem. Indeed, it is necessary to define conceptualisation methods that meet the following formal criteria of adjustment:

1. Understanding the need raised by the user before considering an approach for developing a software system that meets this need.
2. 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.
3. 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 now carried out in SE and KE. The only way of redefining this

process is by taking into account that, as specified earlier, the software development process is related to both the user need space and the machine space of the software that is to meet the need raised by the above user, as shown in Figure 18.

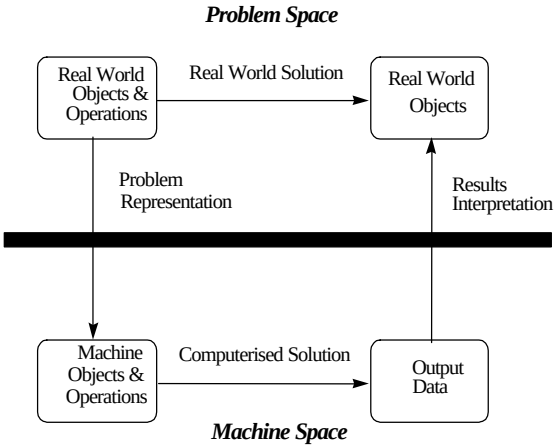

Figure 18. Problem Space versus Machine Space.

This double relationship means first that it is not feasible to build any sort of CM without always taking into account the fact that the end purpose of the development is to build a software product. Second, and as mentioned earlier, it is not advisable to bring design-related issues into the conceptual modelling stage.

Therefore, the only solution is to redefine what are considered to be two different moments in conceptual modelling. The first moment is oriented to the problem, where the attention focuses on customer and/or user needs. The second moment is oriented to the system, where implementation alternatives must start to be considered. These two moments define the conceptual modelling approach presented in the following section.

8. Conceptual models as separate from computational models

Traditionally, as mentioned above, conceptualisation methods have been conditioned by computational constraints. The proposed solution seeks to bring the conceptual modelling process closer to the user domain. Figure 19 shows this approach, where conceptual modelling is composed of two activities: need-oriented modelling and software system-oriented modelling. The need-oriented modelling process is directly linked to the user need and is independent of the chosen development approach. On the other hand, the software system-oriented

modelling is dependent on the development approach and is, therefore, further removed from the user need.

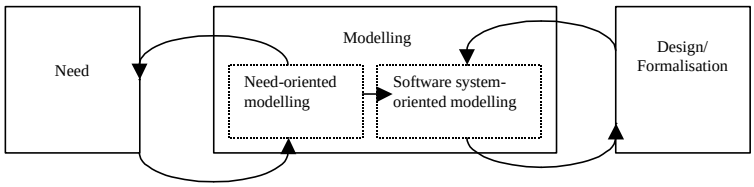

Figure 19. Tasks of a Development Process with Generic CMs.

The goal of need-oriented modelling is to give the software or knowledge engineer an understanding of the procedure used by an expert to solve a problem, and/or, the needs of the users. The software or knowledge engineer, therefore, needs to know what the need is and how it is satisfied in the user's world. So, this need will have to be outlined in a CM, which does not precondition the use of any development approach. This model will be termed generic CM to distinguish them it from the CMs used today. Generic CMs would benefit the development process as follows [10]:

- Independence between conceptual modelling and subsequent development phases, that is, the possibility of carrying out analysis before choosing a given development approach
- Independence from the computer system, that is, the possibility of using conceptual modelling results for developing software using any of the possible development approaches
- Independence from evolving technology, that is, the possibility of the same modelling approach being valid even if the form of software development is modified.

During the software system-oriented modelling process, the software engineer has to select the best current development approach (structured, object-oriented, knowledge-based, database, etc.), or any other approach that could be discovered in the future, for building the software that is to meet the above need, depending on the features of the generic model. The software engineer may opt to adopt different approaches for different parts of the system. Therefore, a set of heuristics have to be defined to determine what would be the best approach for computationally solving a specific need.

Having selected the best suited development approach, the traditional models used by each approach (data flow diagrams, object models, rules, etc.) have to be derived, models which, as mentioned above, are characterised by

their computational bonds. In order to promote this process of transformation, a set of rules will be required for deriving each of the most commonly used CpMs, like object-oriented or structured development models, from the generic CM. Figure 20 describes the possible products to be outputted during problem analysis.

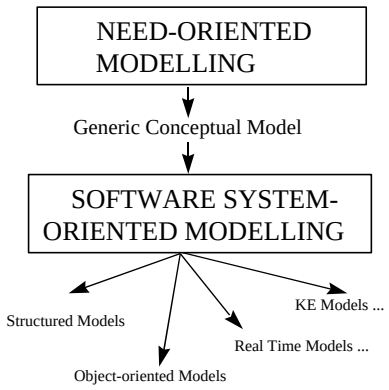

Figure 20. Proposed conceptual modelling process.

9. Conclusions

In this chapter, we have outlined the state of the art in the field of conceptual modelling in SE and KE. Although there are considerable differences in the number and type of CMs in each discipline, the underlying concepts they use all clearly converge. These are: first, object-oriented concepts from the discipline SE; and second, goal, belief or intention, from the discipline of KE. Obviously, the above-mentioned convergence is not due to any effort at standardisation, but to the permeability of the different disciplines, which have managed to take the best from the others.

However, although the current state of the field could be rated as satisfactory, owing to the number and wealth of formalisms used, quite a few researchers who point out a series of shortcomings in all the CMs used. These shortcomings can be divided into two types. Firstly, the modelling formalisms and languages still include too many computational, that is, implementation-related, considerations concerning the concepts they handle, artificially limiting how the problem domain can be described. Secondly, the use of a given representation formalism obliges software or knowledge engineers to adopt a

given development approach, as ensues precisely from the inclusion of the above-mentioned implementation-related considerations.

Therefore, apart from outlining the state of the field of conceptual modelling, an approach has been proposed in this chapter, whose goal is to overcome the shortcomings of the conceptual modelling methods. The main characteristic of this approach is the division of modelling (called analysis in SE or conceptualisation in KE) into two different and separate phases. The first phase is problem oriented and is characterised by the use of a generic CM. The second phase is solution oriented, and the formalisms to be used are prescriptive, including the computational considerations proper to a given development approach.

10. References

1. A.M. Davis, *Software requirements: Objects, functions and states*. Prentice-Hall International, 1993.
2. B.J. Wielinga, A. Th. Schreiber, J.A. Breuker, *KADS: A modelling approach to knowledge engineering*. Knowledge Acquisition, vol. 4, no. 1, 1992.
3. Motschnig-Pitrik R., *The semantics of parts versus aggregates in data/knowledge modelling*. Proceedings of the CAiSE'93, number 685 in Lecture Notes in Computer Science, Springer Verlag, Paris, France, June 1993.
4. A. Borgida, *Knowledge representation, semantic modeling: Similarities and differences*. In Entity-Relatioship Approach: The Core of Conceptual Modeling, H. Kangasalo (ed.), Elsevier Science Publishers B.V., 1991.
5. J. Mylopoulos, A. Borgida, E. Yu, *Representing software engineering knowledge*. Automated Software Engineering, no. 4, 1997.
6. G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde y B. Wielinga, *Knowledge engineering and management: the CommonKADS methodology*. MIT Press, 1999.
7. J. Ares, J. Pazos, *Conceptual modelling: an essential pillar for software quality development*. Knowledge-based systems, , no. 11, 1988.
8. M. Boman, J.A. Bubenko, P. Johannesson, B. Wangler, *Conceptual Modelling.* Prentice Hall series in computer science, 1997.
9. J.D. McGregor, T. Korson, *Object oriented design*. Communications of the ACM, vol. 33, no. 9, 1990.
10. F. Bonfatti, P.D. Monari, *Towards a general purpose approach to object-oriented analysis*. In: Proceedings of the International Symposium of Object Oriented Methodologies and Systems, Palermo, Italy, 1994.
11. G.M. Høydalsvik, G. Sindre, *On the purpose of object oriented analysis*. in: Proceedings of the Conference on Object Oriented Programming, Systems, Languages and Applications, New York, USA, 1993.
12. J. Hoppenbrouwers, B. van der Vos, S. Hoppenbrouwers, *NL structures and conceptual modelling: Grammalizing for KISS*. Data & Knowledge Engineering, vol. 23, no. 1, 1997.
13. L. Adelman, S.L. Riedel, *Handbook for evaluating knowledge-based systems*. Conceptual Framework and Compendium of Methods, Kluwer Academics Publisher, Boston, 1997.

14. B. Biébow, S. Szulman, *Acquisition and validation of software requirements*. Knowledge Acquisition, no. 6, 1994.
15. A. Aadmodt, M. Nygård, *Different roles and mutual dependencies of data, information and knowledge – an AI perspective on their integration*. Data and Knowledge Engineering, no. 16, 1995.
16. N. Juristo, *Guest editor's view*. Knowledge-Based Systems, no. 11, 1998.
17. *Webster's new encyclopedic dictionary*. Könemann, Cologne, 1994.
18. R. Wieringa, *Requirements engineering: Frameworks for understanding*. Wiley, Chichester, 1995.
19. B. I. Blum, *Beyond programming. To a new era of design*. Oxford University Press, New York, 1996.
20. ANSI/X3/SPARC, *Study group on data base management systems*. Interim Report 75-02-08, 1975.
21. J.J. van Griethuysen, *ISO - concepts and terminology for the conceptual schema and the information base*. N695, ISO/TC9/SC5/WG3, 1982.
22. D. Beringer, *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, 1994.
23. P. Loucopoulos, V. Karakostas, *Systems requirements engineering*. McGraw-Hill, Berkshire, 1995.
24. H. Kaindl, *Difficulties in the transition from OO analysis to design*. IEEE Software, vol. 16, no. 5, 1999.
25. A. Newell, *The knowledge level*. Artificial Intelligence, vol. 18, no. 1, 1982.
26. W.J. Clancey, *The knowledge level reinterpreted: Modelling socio-technical environments*. International Journal of Intelligent Systems, vol 8, no. 1, 1993.
27. M. Kolp, A. Pirotte, *An aggregation model and its C++ implementation*. In: M.E. Orlowska and R. Zicari (eds.), Proceedings of International Conference on Object Oriented Information Systems (OOIS'97), Springer-Verlag, Australia, 1997.
28. J. Siddiqi, *Challenging universal truths of requirements engineering*. IEEE Software, vol. 11, no. 2, 1994.
29. Y. Wand, *A proposal for a formal model of objects*. Won Kim, Frederick H. Lochovsky (Eds.): Object-Oriented Concepts, Databases, and Applications, ACM Press and Addison-Wesley, 1989.
30. A. Lamsweerde, A. Dardenne, F. Dubisy, *The KAOS Project: Knowledge acquisition in automated specification of software*. Proceedings of the AAAI Spring Symposium Series, Stanford University, March 1991.
31. J. Rumbaugh, *Object-oriented modelling and dDesign*. Prentice-Hall, 1991.
32. P. Coad, E. Yourdon, *Object oriented analysis*. Yourdon Press, 1990.
33. B. Meyer, *Object oriented software construction*. Prentice-Hall, 1988.
34. J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, *TELOS: Representing knowledge about information systems*. ACM Transactions on Office Information Systems, vol. 8, no. 4, 1990.
35. T. DeMarco, *Structured analysis and system specification*. Prentice-Hall, 1979.
36. D.T. Ross, *Structured Analysis: A language for communicating ideas*. IEEE Transactions on Software Engineering, vol. 3, no. 1, 1977.
37. M. Kirikova, J.A. Bubenko, *Enterprise modelling: Improving the quality of requirements specification*. Information systems Research seminar In Scandinavia, IRIS-17, Oulu, Finland, 1994.
38. E. Yu, *Modelling strategic relationships for process reengineering*. PhD Dissertation, Dept. of Computer Science, Univ. of Toronto, 1995.

39. R. Wieringa, *A survey of structured and object-oriented software specification methods and techniques*. ACM Computing Surveys, vol. 30, no. 4, 1998.
40. I. Jacobson, *Object-oriented software engineering: A use case driven approach*. Addison-Wesley Object Technology Series, 1994.
41. P. Chen, *The Entity-relationship model: Towards a unified view of data*. ACM Transactions on Database Systems, vol. 1, no. 1, 1977.
42. J.R. Abrial, *Data semantics*. Data Base Management, North-Holland, Amsterdan, 1974.
43. E. Yourdon, *Structured analysis*. Yourdon Press, 1989.
44. G. Booch, I. Jacobson, J. Rumbaugh, *The unified modelling language*. Addison-Wesley, 1999.
45. G. Agha, *Actors: A model of concurrent computation in distributed systems*. MIT Press, Cambridge, MA, 1986.
46. H. Nissen, M. Jeusfeld, M. Jarke, G. Zemanek, H. Huber, *Managing multiple requirements perspectives with metamodels*. IEEE Software, vol. 13, no. 2, 1996.
47. A. Dardenne, A. Lamsweerde, S. Fickas, *Goal-directed requirements acquisition*. Science of Computer Programming, vol. 20, 1993.
48. E. Dubois, E. Yu, M. Petit, *From early to late requirements: A process-control case study*. In: Proceedings 9th Intl Workshop on Software Specification and Design, Isobe, Japan, 1998.
49. M. Jarke, J. Bubenko, C. Rolland, A. Sutcliffe, Y. Vassiliou, *Theories underlying requirements engineering: An overview of NATURE at genesis*. Technical Report NATURE-93-01, 1993 (ftp://ftp.informatik.rwth-aachen.de/pub/NATURE/).
50. A. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, W. Van de Velde, *CommonKADS; A comprehensive methodology for KBS development*. IEEE Expert, vol. 9, 1994.
51. Grosso W.E., Eriksson H., Fergerson R. W., Gennari J. H., Tu S. W. y Musen M. A., *Knowledge modeling at the millennium (The design and evolution of Protege-2000)*. SMI-1999-0801 Report, 1999.
52. J. Angele, D. Fensel, R. Studer, *Domain and task modelling in MIKE*. In: Domain Knowledge for interactive system design, Chapman & Hall, 1996.
53. A. Anjewierden, G. Schreiber, *CML2 syntax (2.2.1)*. http://www.swi.psy.uva.nl/projects/kads22/cml2doc.html.
54. B.J. Wielinga, A.T. Schreiber, *Conceptual modelling of large reusable knowledge bases*. In: K. von Luck & H. Marburger (Eds.), Management and Processing of Complex Data Structures, Springer-Verlag, Berlin, Germany, 1994.
55. D. Fensel, *The Knowledge acquisition and representation language KARL*. Kluwer Academic, 1995.
56. D. Fensel J. Angele, R. Studer, *The Knowledge acquisition and representation languaje KARL*. IEEE Transactions on Knowledge and Data Engineering, vol. 10, no. 4, 1998.
57. J. H. Gennari, *A brief guide to MAÎTRE and MODEL*, 1993 (http://smi-web.stanford.edu/projects/protege/nextstep/maitre/MaitreIntro/MaitreIntro.html).
58. J.H. Gennari, S.W. Tu, T.E. Rothenfluh, M.A. Musen, *Mapping domains to methods in support of reuse*. International Journal of Human-Computer Studies, vol. 41, no. 3, 1994.
59. J. Angele, D. Fensel, R. Studer, *Developing knowledge-based systems with MIKE*. Journal of Automated Software Engineering, vol. 5, no. 4, 1998.
60. S. McGinnes. *How objective is object-oriented analysis?* In: Proceedings of the CAISE'92 Advanced Information Systems Engineering, 1992.

61. M. Jackson, *Software requirements and specifications. A lexicon of practice*. Addsion-Wesley, 1995.
62. M. Jackson, *Will there ever be software engineering?* IEEE Software, vol. 15, no. 1, 1998.
63. A. Davis, K. Jordan, T. Nakajima, *Elements underlying the specification of requirements*. Annals of Software Engineering, Balzer Engineering Prublishers, 1997.
64. B. Henderson-Sellers, J. Edwards, *The object oriented systems life cycle*. Communications of the ACM, vol. 33, no. 9, 1990.
65. P. Jalote, *An integrated approach to software engineering*. Springer-Verlag, 1997.
66. L.M. Northrop, *Object-oriented development*. In: Software Engineering, IEEE Computer Soc. Press, Los Alamitos, USA, 1997.
67. N. Juristo, A.M. Moreno, *Introductory paper: Reflections on conceptual modelling*. Data and Knowledge Engineering, no. 33, 2000.
68. D. Coleman, P. Arnold, S. Bodoff, C. Dollin, *Object-oriented development: The fusion method*. Prentice-Hall, 1994.
69. D. Champeaux, D. Lea, P. Faure, *Object oriented system development*. Addison-Wesley, New York, 1993.
70. R. Wieringa, *Object-oriented analysis, structured analysis, and jackson system development*. In: F. van Assche, B. Moulin, C. Rolland (eds.) Proceedings of the IFIP WG8.1 Working Conference on the Object-Oriented Approach in Information Systems, North-Holland, 1991.
71. P. Ward, S. Mellor, *How to integrate object orientation with structured analysis and design*. IEEE Software, vol. 6, no. 2, 1989.
72. F.Y. Kuo, *A methodology for deriving an entity relationship model based on a data flow diagram*. Journal of Systems and Software, vol. 24, no. 2, 1994.
73. J. George, B. Carter, *A strategy for mapping from function-oriented software models to object oriented software models*. Software Engineering Notes, vol. 21, no. 2, 1996.
74. J. Mylopoulos, L. Chung, E. Yu, *From object-oriented to goal-oriented requirements analysis*. Communications of the ACM, vol. 42, no. 1, 1999.