The Knowledge Engineering Review, Vol. 24:1, 5–21. © 2009, Cambridge University Press
doi:10.1017/S0269888909000150 Printed in the United Kingdom

The concepts of model in information systems engineering: a proposal for an ontology of models

DIANA M. SÁNCHEZ, JOSÉ MARÍA CAVERO and ESPERANZA MARCOS

Kybele Research Group, Department of Computing Languages and Systems II, School of Computer Science Engineering, Rey Juan Carlos University, C/Tulipán s/n, Móstoles, 28933 Madrid, Spain;
e-mail: diana.sanchez@urjc.es, josemaria.cavero@urjc.es, esperanza.marcos@urjc.es

Abstract

Modelling is one of the most frequent tasks in the area of information systems (ISs), with models such as schemata, ontologies, patterns and architectures forming the bases for their creation. Very often, however, the difference between these types of models is not clear and causes confusion and erratic use of the terms. The aim of this paper is to clarify the concept of model through a study of some of the more common ones used in ISs. This proposal is presented through an ontology, where we show how we conceptualize models according to their role in the development of an IS: the model understood as the representation of a domain or as a reality serving as an example.

1 Introduction

Today, the concept of *model* is to be found in many different areas in the field of computing. In the development of many information systems (ISs), it may be used to mean a database schema, an eXtensible Markup Language (XML) schema, a diagram of classes, of collaboration or of Unified Modelling Language (UML) use cases, a data (meta)model such as a relational model, a meta-metamodel of the Meta-Object Facility (MOF) type, and so on.

It is obvious that, despite the similarities among the above concepts, there are also many differences and that the role played by each one of these *models* is different. Generally speaking, although each model has a specific function, the roles taken on by a model during the development of an IS fall into two groups. These roles have been studied by Marcos and Marcos (2001), and have been named *model as original* and *model as copy*. The former concentrates on models as realities in themselves serving as examples, whereas the latter concerns models with the aim of representing a domain. *Models as copies* represent a domain according to a specific point of view. This means that one domain may have multiple *models as copies*, depending on the point of view used to build it.

These two roles can be illustrated with two situations, the first of which is an artist painting a portrait. In this case, the person being painted plays the role of *model as original*, someone who exists but who is taken as a reference. The second situation is of a scientist designing a mathematical model to explain a phenomenon. This second model is not reality, but is the representation of a phenomenon, a *model as copy* of the event studied.

Figure 1 shows another example from the computing point of view which introduces a new use of the model concept that increase the confusion. On one hand, we have the original model (in this case a business selling products), which we want to represent in an IS; and on the other hand, its representation by means of a schema or diagram using some formalism. In this case, we use the Entity/Relationship (E/R) *model*, which allows us to create a *conceptual data model* that will

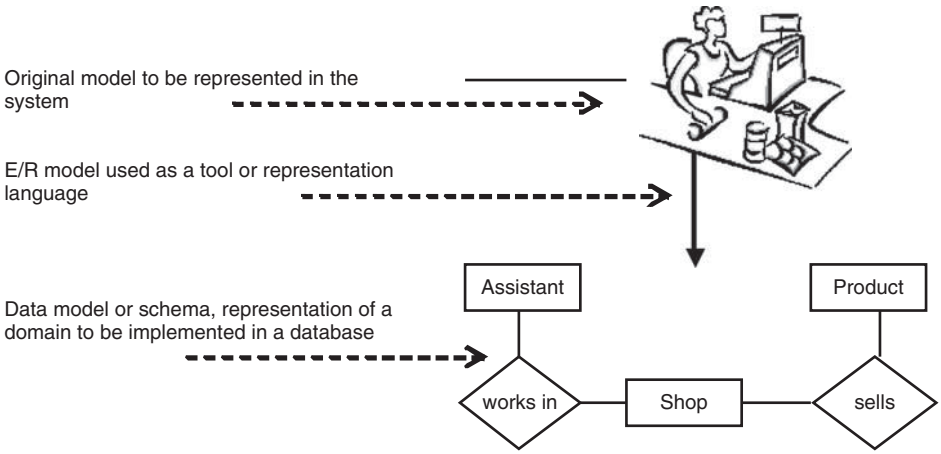

Figure 1 Different uses of the concept *model*

eventually be implemented in a database management system. As Date (2003) recognizes, it is quite normal in computing to use the name *data models* for both concepts, that is, for the tool or formalism (in this case, the E/R model) and the result of its application (the schema or E/R diagram).

According to Marcos and Marcos's classification, both reality and the E/R model would be classed as models as originals, whereas the schema would be a model as copy.

The situation above contrasts with the quest for standardization of concepts, which has brought about, among other things, ontologies in computing. Ontologies are a type of model concentrating on representing the concepts present in a domain, so ontologies are a type of model as copy. The aim of computing ontologies is to facilitate tasks of sharing and interoperability between different applications (Gómez-Pérez *et al*., 2003) by means of sharing common concepts, for which ontologies seek to represent our knowledge of a domain computationally. But for this to be possible, agreement must first be reached on the concepts involved, which fuels much debate.

The aim of this paper is to present an ontology of models, looking at the different meanings of the concept of *model* and allied ones to make it possible to assess the role of some of those used in the field of IS development. The construction of this ontology involved studying the most common uses of the term *model* and other similar ones in ISs, on the basis of which models have been classified according to Marcos and Marcos (2001).

This ontology has been expressed in UML and will be constructed step by step to eventually show our division of models by role and the classification of those most used in ISs. Moreover, the analysis of each model will reveal that they can play either role. Unlike the studies mentioned above, this one also tackles allied concepts such as schema, ontology, pattern, architecture and so on, from the discipline of ISs and following the lines of research laid down in earlier studies (Sánchez *et al*., 2005).

This article is organized as follows. Section 2 summarizes related work on the concept of model and its uses in ISs. Section 3 is a discussion on various types of model in ISs. Section 4 concerns the debate of the role taken on by each of the models presented in Section 3. Here an analysis is offered of the meaning of each model, and on the basis of this analysis, the model ontology is constructed step by step. Finally, Section 5 summarizes the discussion and sets out the conclusions.

2 Related works

In ISs, although much effort has been concentrated on finding technological solutions, little has been done to create discussion forums for concepts of their daily use; the number of people in the IS community using terms like model, ontology or schema is increasing by the day. In fact, one of the most common proposals for the development of ISs, model-driven architecture (MDA)

(Miller & Mukerji, 2003), focuses its efforts on models that are treated as elements of the first category. Following the example of MDA, ontology-driven architecture (ODA) (Tetlow *et al.*, 2006) launches a proposal in which the main elements are ontologies rather than models.

Examples such as MDA or ODA justify a discussion of the concept of model, on the lines of such studies as Bézivin's (2005) and Favré's (2004), in which each author presents his own ideas on models and the roles they play in model-driven engineering (MDE). On the basis of MDA and the four-layer architecture, which will be explained in Section 3.4, both authors discuss the rules of the relation that exists between the M0-layer (Real World) and the M1-layer (Models), which are compared to the rules of the relation between the M1-layer and the M2-layer (metamodels) or between the M2-layer and the M3-layer (meta-metamodels).

Atkinson and Kühne (2003) present an interesting debate on how to understand the latest version of the four-layer architecture, in which the object model is associated to the M1 layer, rather than the M0 layer. To clarify the new four-layer architecture, these authors use two different views: the linguistic metamodelling view and the ontological metamodelling view, both acting within model-driven development (MDD). Another interesting discussion of the meaning of models was carried out by Seidewitz (2003), who considers the meaning of a model as having two aspects: the meaning as interpretation of a model and the meaning as semantics of the model. Others' related works are: Cook (2004), who makes a deep reflection on the role of models in software development, in particular the definition and use of domain-specific languages (DSL) oriented to product lines; Cañete Valdeón *et al.* (2004), with a discussion on the capacity of representation of current modelling languages; and finally Ludewig (2004), who analyses the role of models in software engineering.

Nevertheless, none of those studies focused on creating a clear conceptual framework of the terms most widely used in ISs to attribute a role to each type of model. To fill this gap, in this study we present an original ontology on the concept of model.

3 Discussion on different types of models

Here, we present some concepts to contextualize our work and to use in the construction of the ontology presented in Section 4. These concepts are schema, diagram, architecture, ontology, pattern and metamodel, each being briefly discussed and our definition presented, which will be the one used in the construction of the ontology.

3.1 Schema

Generally speaking, a schema is 'an outline of a plan or theory' (Hornby, 2000). In computing, the use of schemata is associated basically to databases and XML.

In the field of databases, a schema is a description of the structure of the data to be implemented in a database (i.e., the metadata). As we said in the earlier section, in some branches of computing, it is normal to assign the name 'data model' to what we call 'schema' here, so confusion often arises from using the term 'data model' to designate both the representation and the formalism used (relational model, E/R model, etc.).

Within the database community, however, the distinction is clear, and a data model is 'a set of concepts that can be used to describe the structure of a database. Most include a set of operations for specifying retrievals and updates on the database' (Elmasri & Navathe, 1994). Therefore, a data schema is the result of applying a data model to a domain. In other branches of computing, however, the terms used are different, and the term 'model' is used for what we here call a 'schema', and the term 'metamodel' for what we here call a '(data) model' (Date, 2003; Dittrich, 1994).

On the other hand, an XML schema 'expresses shared vocabularies… provides mechanisms to define and describe the structure, content, and to some extent semantics of XML documents' (W3C, 2004a). An XML schema describes the structure of a set of XML documents. This definition of schema will be observed to be very similar to the one used in databases.

The construction of our definition of schema is based on the definitions above and on the use given to them. Therefore, *a schema is the abstract and organized representation of a domain*. It is abstract because data irrelevant from a given point of view are eliminated, and it is organized because its aim is to show a domain clearly so as to facilitate its comprehension.

According to the type of schema to be constructed, different techniques or formalisms of representation may be used (for example, the *E/R data model*). These techniques may be graphical or written, and will specify how each element of the domain is to be represented within a schema. As stated above, these techniques are also called metamodels, an example being the E/R model used in databases. Figure 2 shows two examples of data schemata representing sales in a shop. The first is made by using the E/R model and the second is a fragment of the XML schema.

A schema forms part, in principle, of the model-as-copy category, as it comprises simplified representations of a domain. Nevertheless, as we shall see in Section 4, it may also act as a model as original.

3.2 Diagram

The arrival of UML (Booch *et al*., 1998) and the boom of the unified process have brought diagrams to the forefront of IS development, and just as schemata are habitually associated with databases and XML documents, diagrams are associated with the representation of a view in UML (Booch *et al*., 1998). In UML, each view has an aim and fulfilling it means using one or more types of diagram. For example, in the interaction view showing the set of messages between various objects, sequence and collaboration diagrams are used.

Etymologically, the word diagram comes from the Latin *diagramma*, meaning 'figure'. Therefore, bearing in mind both the origin of the word and its use in UML, we define diagrams as *the abstract and organized graphic representation of a domain*. What distinguishes diagrams from schemata is that the former will incorporate graphic elements, so a diagram cannot consist only of text.

Examples of diagrams are those of classes, use cases, components, activity, states and deployment. Figure 3 shows an example of a use-case diagram.

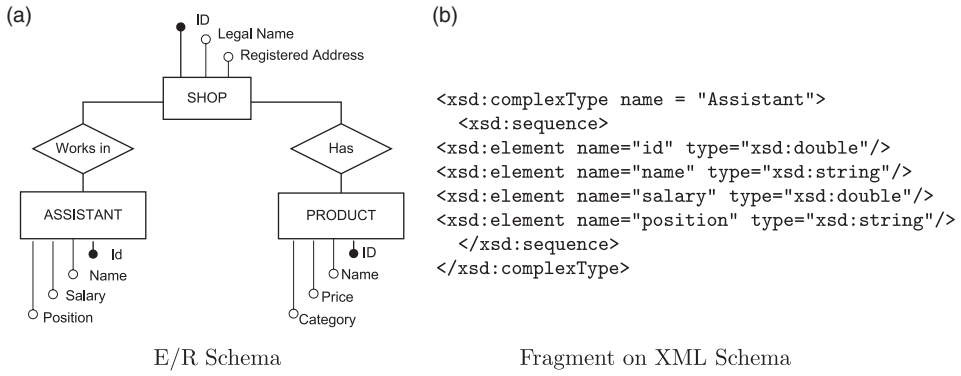

Figure 2 Examples of schemata

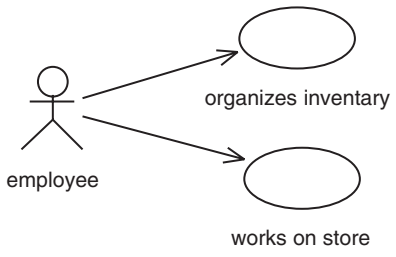

Figure 3 Example of use-case diagram

A diagram is a special type of schema based on the use of graphics. Therefore, the discussion on the role of models of this type will relate diagrams to schemata and will be detailed in Section 4.

3.3 Ontology

In computing, ontologies arose as an answer to the problem of interoperability of systems and the need to represent concepts in a computing medium. Although the concept of ontology has been discussed by several authors (Gruber, 1993; Guarino, 1995; Studer *et al*., 1998), the most widely used definition is Gruber's (1993): '*an ontology is an explicit specification of a conceptualization*'. This definition of ontology is the one used in this paper. Conceptualization is the process of creating a concept of 'something' in the world (Genesereth & Nilsson, 1987), which is why ontologies in ISs are associated with concepts and conceptual schemata (Gómez-Pérez *et al*., 2003). Conesa *et al*. (2003) define conceptual schema as the specification of the knowledge possessed on a domain in a specific language and they also say that '*conceptual schemas are elaborated during the conceptual modelling activity*'. For us, the difference between ontology and conceptual schema will be one of field. An ontology seeks for its specification to be universal, whereas in a conceptual schema (for example, a database schema), the specific knowledge is taken as valid for a special case, although this does not rule out its possible application to other cases.

Ontologies are considered higher-level structures in which the concepts involved in a domain along with all the possible semantic variations that a concept may have are set out and specified. As a consequence of their concept-specifying task, ontologies are used as elements for creating correspondences between variables or processes of two or more systems. They may therefore be used as intermediaries facilitating interoperability. At present, there are several languages for specifying ontologies, like the Ontology Web Language (OWL) (McGuinness & Harmelen, 2004), the Resource Description Framework (RDF) (W3C, 2004b) and the Web Service Modelling Language (WSML) (De Bruijn *et al*., 2005), which seek to add semantics to the definition of a concept.

Ontologies, as a concept very similar to the schema, are models as copy. A computing ontology seeks to create a representation of a set of concepts, so what it does is to take the concepts from the real world as its frame of reference and to reflect them through any of the ontology-specifying languages. In Section 4, it is explained that, once created, ontologies may also act as models as original.

3.4 Metamodel

The creation of a schema or of any other model as copy must be done by means of some formalism or technique, establishing how each element of a domain is to be represented. These 'techniques' are called metamodels.

The prefix *meta* is used to specify recursive definitions of a term. Thus, a metalanguage is a language of languages and a metamodel is a model of models (OMG, 2004). It should be noted that even within this widely accepted definition, the term *model* takes on both roles. Analysing the definition shows that the first use of the word *model* concerns the model as an existing reality that is taken as an example, whereas its second use refers to models as representations of domains. With these considerations in mind, *a metamodel* is defined *as a model as original used to describe models as copies*. As we have already stated, the database community uses the term *data model* for what other computing communities, such as software engineering, call a *metamodel*.

A metamodel makes it possible to understand a model (as copy) by showing the types of elements that can be found within that model as copy and specifying the relationships between them. For example, the metamodel that is used to create the schema in Figure 2 is the E/R (meta)model. Again in the domain of data, another example of a metamodel is the relational (meta)model. The use of metamodels has been fomented by MDA, MDD and model-based n-layer architectures. The number of layers that can exist in these architectures is, in principle, unlimited. Each new layer represents a higher level that supports the one immediately below it. Figure 4 shows a four-level architecture

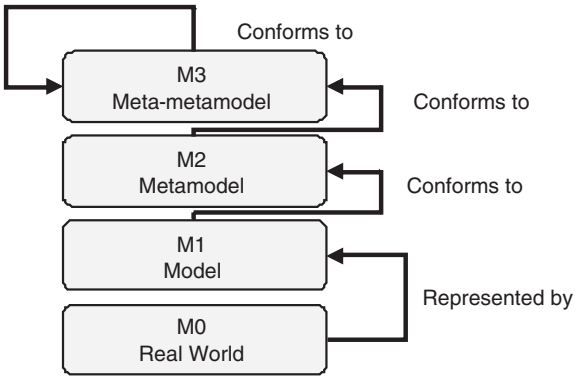

Figure 4 Metamodels in a four-level architecture

and the relationship between each of the layers. Bézivin (2005) presents a far-reaching discussion on the relationships between layers in this architecture.

The models classified in level M1 correspond to what we call schemata or diagrams (in UML) in Sections 3.1 and 3.2.

MOF is the formalism created by OMG for the construction of metamodels (OMG, 2002). MOF is, therefore, a meta-metamodel with the purpose of specifying, constructing and managing metamodels. To carry out these tasks, MOF has a language of its own. The use of MOF makes it easier to transform models and to carry out exports and imports between applications.

A metamodel may act as a model as original or as a model as copy, as will be explained in Section 4.

UML 2.0 (OMG, 2005, 2006) has changed some features of the four-level architecture. Unlike the versions 1.x of UML, in which the objects' diagram (instances) belonged to level M0, the latest version of UML associates this kind of diagrams to the M1 layer. Now, the M0 layer contains 'the runtime instances of model elements' (OMG, 2005). These *runtime instances* refer to real objects, that is, entities being modelled. Objects in the object diagram are now just snapshots of this M0 level instances. Atkinson and Kühne (2003) present a good reflection on this topic, proposing two ways of understanding the latest interpretation of the four-layer architecture: a linguistic and an ontological view.

3.5 Pattern

Patterns afford IS development a set of optimal solutions to problems that are frequent within a context. The definition of *pattern* is attributed to Alexander (1979), who explained that '*each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution a million times over, without ever doing it the same way twice*'. From this reasoning, Alexander concluded that '*each pattern is a three-part rule which expresses a relation between a certain context, a problem, and a solution*', which is the definition used in this paper.

There are different types of patterns in IS development, among which we may mention architectural, analysis and design ones. Design patterns are the most common in ISs and have been defined by Gamma *et al.* (1995) as '*descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context*'. Patterns favour the reuse of software as they describe solutions that can be used many times over in different situations and by any developer.

An example of a design pattern is the Observer pattern, which establishes that if several objects depend on the possible change of state of a single object, this will notify the others of its change of state instead of each of the dependent objects consulting the state of the object from time to time. This pattern is illustrated in Figure 5.

As a proven solution to a common problem, a pattern takes on the role of model as original, whereas in its application to a given case, which is its implementation, it is a model as copy. This duality will be explained in more detail in Section 4.

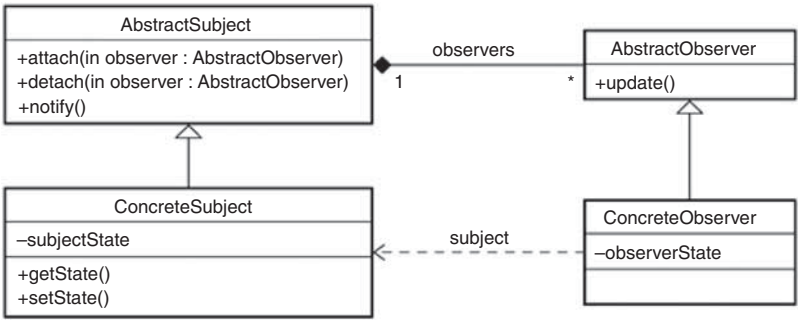

Figure 5 The Observer pattern

3.6 Architecture

Another fundamental element in IS construction is architecture, in which the main components of the system, their mutual relationships and way they are coordinated to reach an objective of the IS are all shown (Clements, 1996; Shaw & Garlan, 1996).

Smolander (2002) presents an interesting reflection on the meaning of 'architecture' in Software Engineering, concluding that it has four meanings: the architecture of a system to be implemented, architecture as the documentation of the system for future developers, architecture as the concerted structure of the system and architecture as decisions critical to the creation of the system.

The standard definition, given by the IEEE, and which we use here, is of architecture as '*the fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution*' (IEEE, 2005). This definition embraces the last three of those given by Smolander.

Smolander's first meaning centres the work of architectures on the compilation of a set of principles governing a strategy for designing the architecture of the system. Perry and Wolf (1992) define an *Architectural Style* as an abstraction of types of elements and formal aspects from different specific architectures. An architectural style embraces the essential decisions on architectural elements and emphasizes major restrictions of the elements and their relationships. Examples of architectural styles are the 'Client-Server', 'Pipes and Filters' and 'Blackboard' styles. Therefore, an architecture may follow an *Architectural Style*.

The relationship between architecture and architectural style is similar to that between schema (*model as copy*) and model (*as original*). The architectural style conditions the constructors and types of elements that can be used in an architecture just as a data model (E/R or relational) conditions the constructors to be used in a data schema. If a given style has been followed in the construction of the architecture of a system, the style plays the role of model as original as it is taken as a reference point, whereas the architecture will be a model as copy of the style, as it will represent the system through the style. Once created, this architecture can also take on the role of model as original, as it can be implemented in a system or may even be used to generate new architectural styles.

Figure 6 shows the relation between these concepts by means of an example. On the right, we have represented the client–server architectural style, which is composed of three elements (client, server and connector), and a set of constraints showing how these elements can be combined to create an architecture. The left-hand side shows an example of an architecture, which follows the guidelines laid down by the architectural style.

3.7 Process model

A process model is '*an abstract representation of a process. It presents a description of a process from some particular perspective*' (Sommerville, 2001).

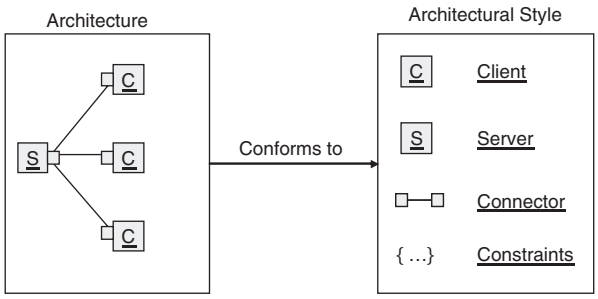

Figure 6 An architecture created that conforms to the Client–Server architectural style

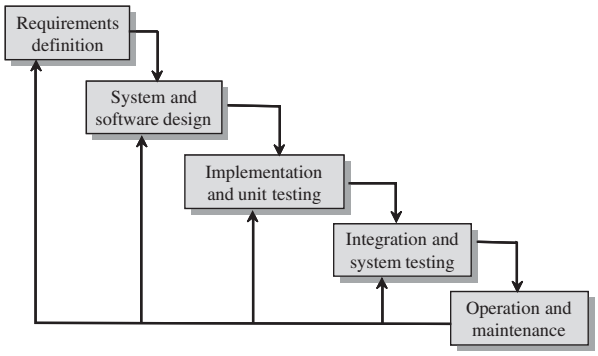

Figure 7 Cascade model for software development

Process models are useful abstractions that explain different approaches to software development. They propose a division of the software creation process into stages, each of which shall produce a group of results to be used in the next stage until the software is complete. Although a process model does not have to detail the exact activities of each stage, it does specify the relationship between them, the order in which they should be executed and whether the software is to be constructed in a single cycle or in a number of repetitions. The result of applying a process model to a given development gives rise to a process in which, apart from software, models, documentation, etc., are also obtained. An example of a process model is the Cascade Model (Figure 7), which divides software development into five stages, which must be developed in sequence, so that one does not begin until the earlier one is finished.

A process model is a model as original, and the process resulting from its application is a model as copy.

To sum up this section, Table 1 summarizes the concepts studied and specifies the definition of each term. These definitions are those used for the construction of the ontology in the following section.

4 Proposal for an ontology of models

In this section, we propose an ontology that seeks to clarify and classify some terms concerning models, which have been defined. This ontology will be constructed step by step throughout the section and will be expressed by means of a UML class diagram.

This ontology presents the models according to the roles established by Marcos and Marcos (2001), who state that they may behave as *models as copies* or *models as originals*. The former are those that represent a domain, whereas the latter are realities in themselves and are used as examples. Figure 8 shows the starting point of our ontology.

To determine the role played by each type of model, the meaning of each term will be used. The association between roles and each specific model may be represented by means of an association

Table 1 Summary of concepts presented so far

Concept	Definition
Schema	The simplified and ordered representation of a domain.
Diagram	A schema based on the use of graphics.
Ontology	'…Is an explicit specification of a conceptualization' (Gruber, 1993).
Metamodel	A model as original used to describe models as copies.
Pattern	'…Is a three-part rule which expresses a relation between a certain context, a problem, and a solution' (Alexander, 1979).
Architecture	'…The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution' (IEEE, 2005).
Architectural style	'…Is that which abstracts elements and formal aspects from various specific architectures' (Perry & Wolf, 1992).
Process model	'…Is an abstract representation of a process. It presents a description of a process from some particular perspective' (Sommerville, 2001).

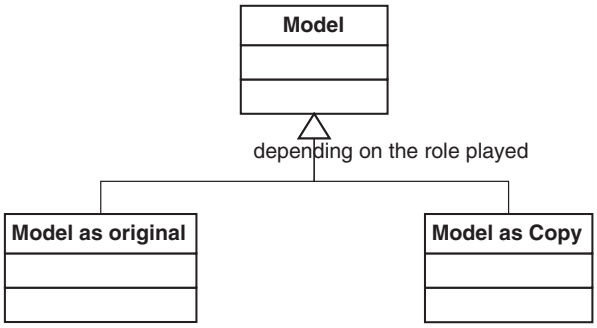

Figure 8 Classification of the models depending on the role played

of the *view-of* type or by means of a generalization. A generalized relationship will represent the role played by a model according to its definition. We shall term this role *default role*. When a model type can take on both roles, the second one will be represented through a stereotyped dependency as *view-of*. This stereotype expresses that this role is another way of seeing (a view) the model type being analysed.

4.1 Schemata and diagrams

The definitions of schema and diagram are similar. Both are simplified and ordered representations of a domain, with the difference that diagrams are basically graphic, that is, a diagram cannot consist only of text, whereas a schema can. Therefore, a diagram may be said to be a class of schema, but characterized by the use of drawing.

From its definition, a schema assumes the role of *model as copy*, as illustrated in Figure 1. Nevertheless, when the schema has already been created and exists, it can become a *model as original*, which happens when a schema is taken as the basis for the construction of a system. An example is given in Figure 9.

It should be made clear that a schema cannot play both roles at the same time in the construction of an IS. Its primary role is as a *model as copy* and its conversion into a *model as original* will depend on its implementation, that is, any schema will be a model as copy by default whereas not all schemata can become models as originals. With this dissertation we present the first approach to our ontology in Figure 10.

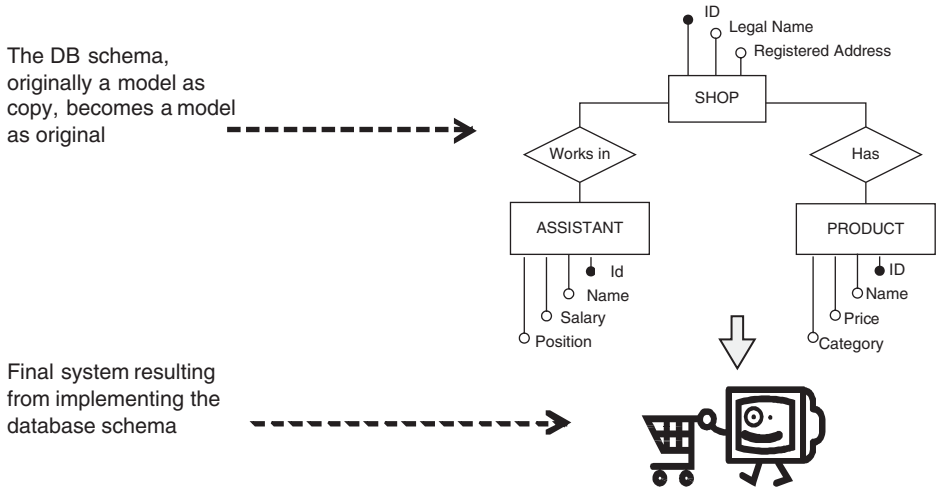

Figure 9 Schemata in the role of models as originals

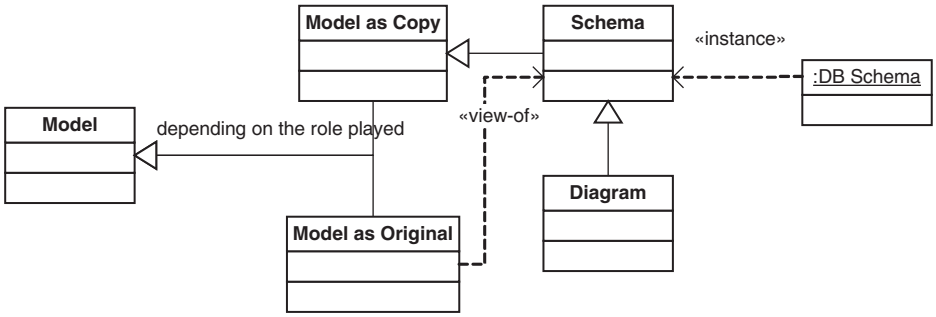

Figure 10 Schemata and how they can take on each role

4.2 Ontologies

By defining an ontology as the representation of a conceptualization, we assume that the role of ontologies is that of *models as copies*. Ontologies primordially represent the concepts present in a domain. Representing a concept involves rather more than specifying a definition, for it is necessary to be able to visualize and understand all the semantics that a term can imply. The inclusion of the semantic properties of a term is what makes ontologies one of the model types most widely used in IS development today.

Like a schema, an ontology may also play the role of *model as original* once it has been created. Indeed, an ontology may be taken as a *model as original* to generate other types of computing models like conceptual models or DB schemata. An example of the use of ontologies as models as originals is presented by Conesa *et al*. (2003), who use them to generate conceptual schemata.

Although an ontology may play either role, its default role is that of a model as copy, as this is the one reflected in its definition. Figure 11 shows how an ontology may have either role.

The inclusion of ontologies in our model ontology is shown in Figure 12.

Ontologies, diagrams and schemata may be represented using the same formalism (for example, UML).

4.3 Metamodels

In the field of ISs, each schema that represents a domain (*model as copy*) must fit a metamodel (Miller & Mukerji, 2003). Metamodels permit the description of models as copies, that is, the

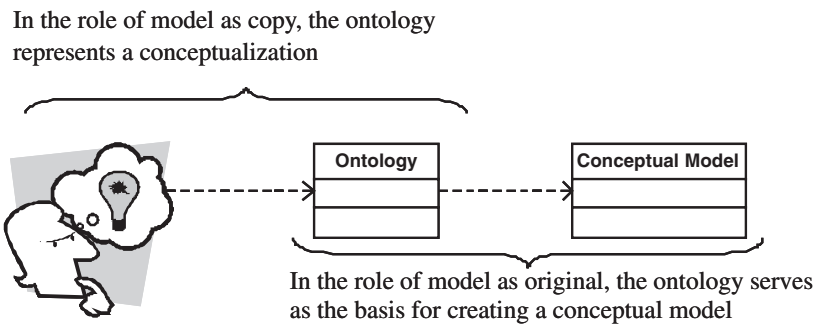

Figure 11 Ontologies assuming both roles

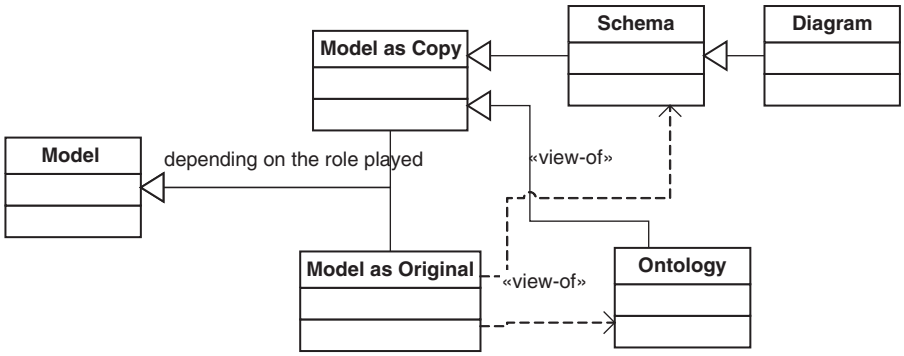

Figure 12 The inclusion of ontologies in the model ontology

domain of a metamodel is the type of models as copy that it describes. To clarify this dichotomy, we offer the following example:

Problem	To represent the interaction of a shop assistant.
Method	The interaction may be represented by means of a use-case diagram. Therefore it is necessary to have rules for making that diagram and the answer is found in the UML metamodel. The metamodel makes it possible to know the meaning of each element included in the diagram (actor, use cases, relationships) and the possible types of interactions between them. Finally, bearing the metamodel in mind, the diagram illustrating the interaction is constructed.
Solution	Use-case diagram (see Figure 3).

In this example, the UML metamodel (*model as original*) does not represent the interaction of the assistant in the shop, but concentrates on describing how to show interactions that, in turn, are models as copies, in this case a use-case diagram. The UML metamodel governs all the use-case diagrams and, conversely, any use-case diagram must *fit* the UML metamodel. A more thorough explanation of this idea is given by Bézivin (2005).

A metamodel is a model in itself and, as such, may take on either of the roles studied. When we view the metamodel as the representation and specification of a method, technique or procedure, we are associating it with the role of a *model as copy*. Nevertheless, when we take the metamodel as a reality that we refer to in order to create a schema or diagram, that is, when we take it as a guide, it has the role of a *model as original*.

By default, a metamodel acts as a model as original, that is, it is created as a frame of reference for others. In the example given, the UML metamodel takes on the role of a model as original from the point of view that it is taken as a reference for creating the use-case diagram.

A metamodel is associated with a language in which a domain is defined according to the rules of the metamodel. This language must be sufficiently expressive and complete to represent

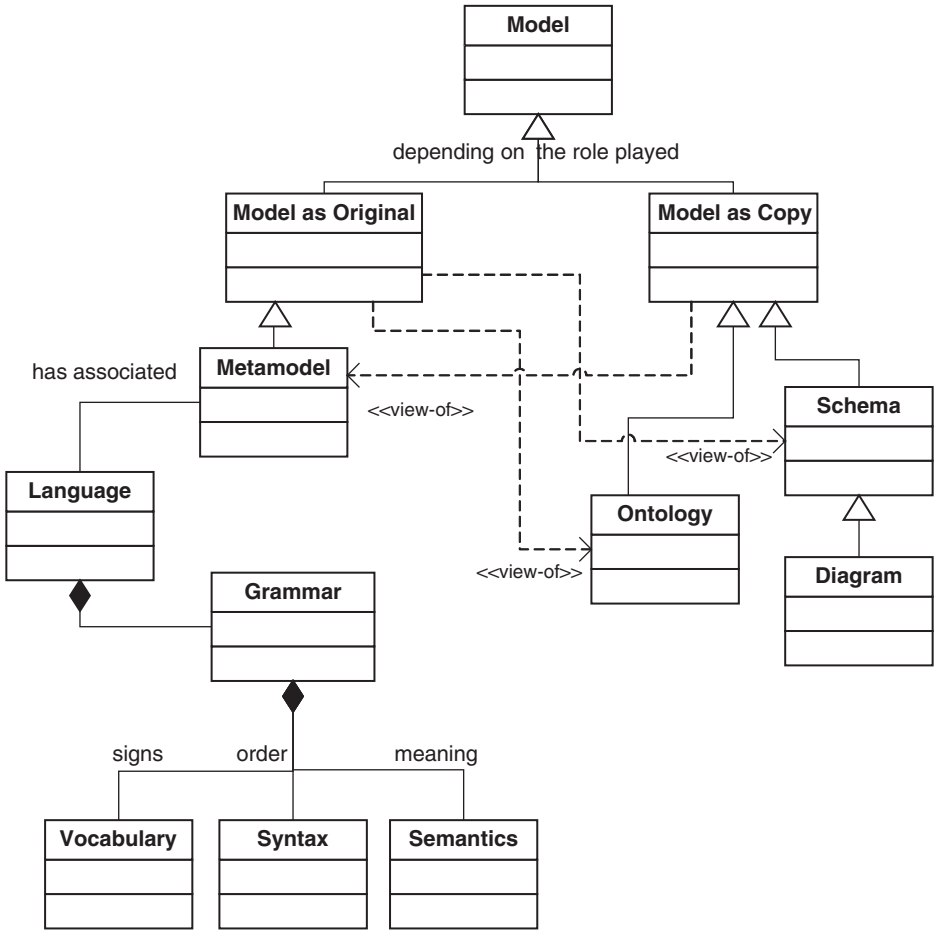

Figure 13 Metamodels in our ontology

everything that is specified in the metamodel. A language must afford a lexicon or vocabulary that can cover all the possible elements, a syntax or set of rules as to how the lexical items must be combined and organized, and semantics or meanings of both the individual lexical items and combinations of them.

Some metamodels create a language of their own and give it all the resources necessary to specify all the elements constituting the metamodel. The specification of a language includes both the creation of a notation and the meaning to be given to each element of the language. Let us consider the case of UML. UML means Unified Modelling Language, but its scope is wider than that of a language. As well as a set of diagrams and elements that may be used in these diagrams, UML defines a whole strategy for modelling domains in computing through five views, each of which approaches the modelling of a system from a specific point of view. All this strategy (views, diagrams, elements of diagrams, etc.), is formalized through the UML metamodel. Furthermore, the UML metamodel is recursive, that is, it is represented by means of UML syntax (classes, associations, etc.), and must be interpreted according to UML semantics.

Figure 13 includes the concept of metamodel in our ontology and its relationship with language.

4.4 Architectures and architectural styles

The concepts of Architecture and Architectural Style are intimately bound up. Architectural Styles, as the descriptions of ways a system can be structured, are a frame of reference taken into account in the construction of a system. To choose an architectural style is to adopt a way of

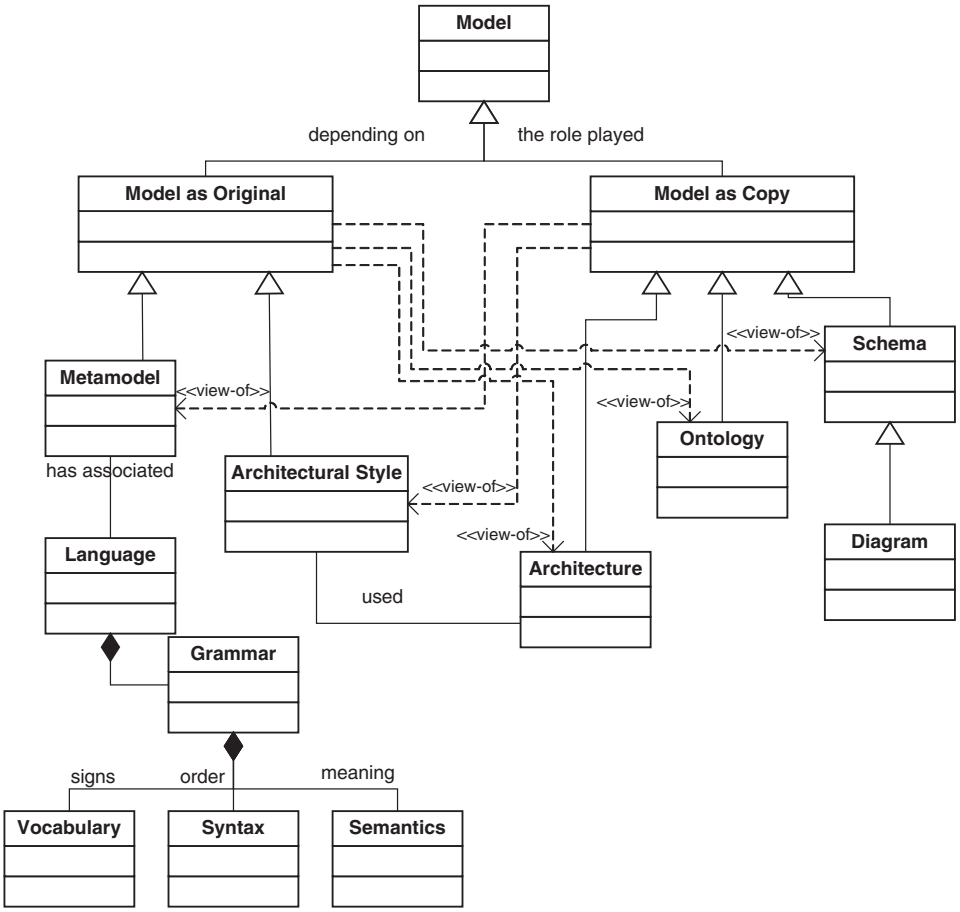

Figure 14 Architectures and architectural styles in our ontology

viewing a system and taking it as a reference for applying it to a reality and creating an architecture. For its part, the *architecture* will set out, in a real case, the elements that a system will have and the way in which they will communicate and evolve.

From the point of view of the roles, if an architecture is based on an architectural style, the architecture acts as a *model as copy* of the architectural style, which acts as a *model as original*. The relationship between the two is similar to that existing between model and metamodel. The difference lies in the fact that an architecture does not always fit an architectural style, which is the case of models that have to fit a metamodel.

By default, an architecture acts as a *model as copy*, that is the main role of an architecture is to represent a system. However, an architecture can become a model as original. This role change occurs when the architecture is implemented in a complete system or when it is taken as a reference for other systems.

In contrast, the main role of architectural styles is to be *models as originals*, for they seek to serve as examples of how to create an architecture. An architectural style may make use of metamodels on which it can lean to represent some of the different facets of a domain. The architectural styles and architectures included in the ontology are shown in Figure 14.

4.5 Patterns and process models

Patterns as the expression of the relationship between a problem, a context and a solution (Alexander, 1979) are *models as originals*. A pattern is taken as a reference point for creating a solution and applying it to a specific case. A pattern offers a solution to a given problem, and may

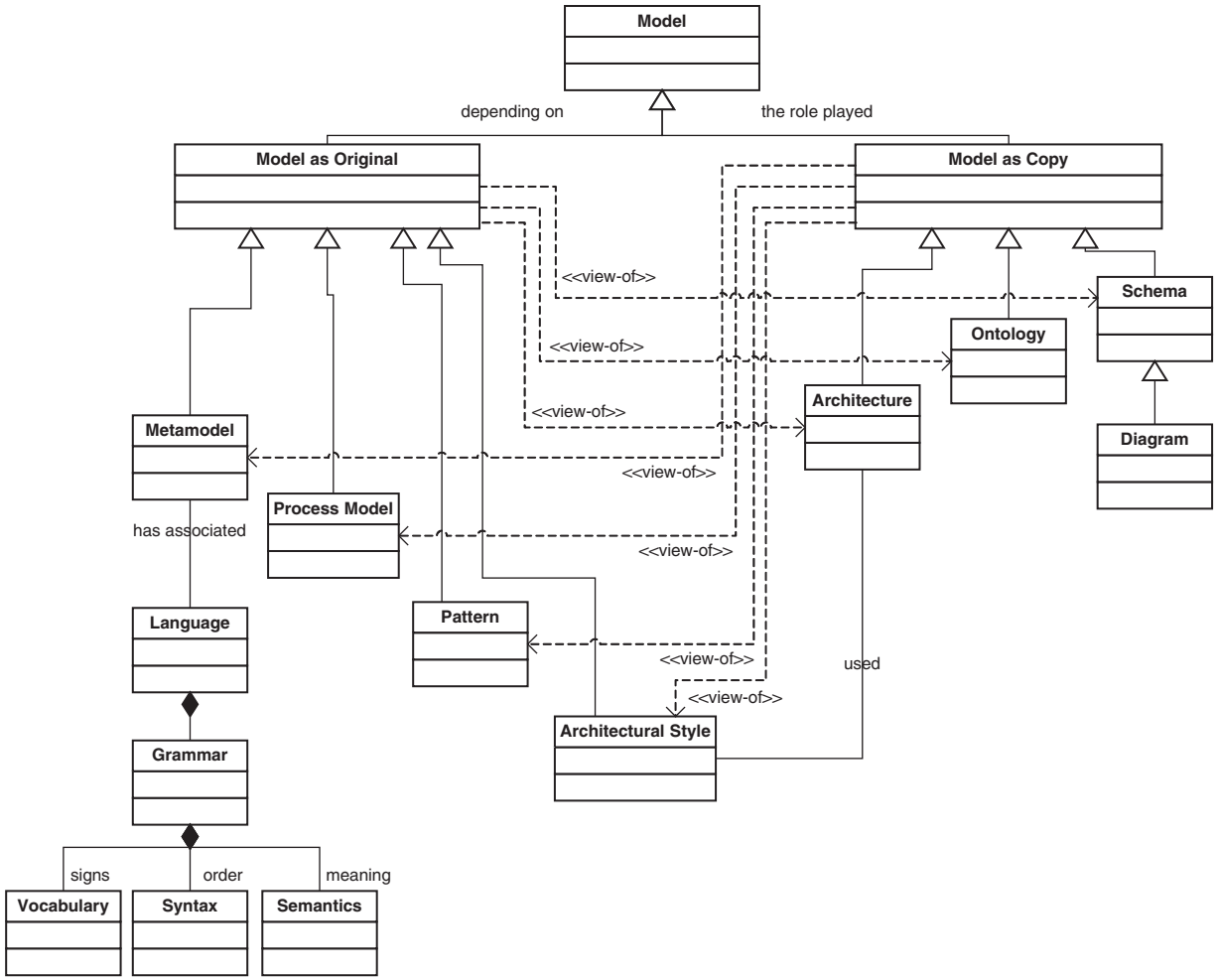

Figure 15 Ontology of models depending on their role in an IS.

Table 2 Types of model and their principal roles in ISs

	Default role	
Type of model	Model as original	Model as copy
Schema		X
Diagram		X
Ontology		X
Metamodel	X	
Architecture		X
Architectural style	X	
Pattern	X	
Process model	X	

be used many times in different solutions. Furthermore, a pattern tells us how the elements of the solution offered have to relate to one another.

A pattern is a restricted architectural style on a small scale. It is restricted by the context in which it can be applied, and on a small scale, because it offers a solution for a specific problem. When a pattern is implemented, the implementation is a model as copy of the pattern.

A pattern takes on the role of *model as copy* when we see it as the representation of the compilation of good practices on a problem. This is why many patterns are represented by means of schemata.

The case of process models is similar to that of patterns. A process model presents a way of constructing software that can be followed by a development group in a software project. The process model will specify a set of stages and results of each one and will guide the development group, that is, the process model will act as a *model as original*.

Nevertheless, a process model may also be seen as the representation of a good idea for guiding a software development process, in which case it becomes a *model as copy*, although this is not its primary role.

With the inclusion of patterns and process models, we complete the construction of the ontology of models, which is shown in Figure 15.

4.6 Considerations of the ontology

According to the above discussion, it is obviously not possible to attribute a single role to the different types of models considered here. The above cases show that it is possible for a model to have either role, as original or as copy. It will also be noted that it is impossible for a model to play both roles simultaneously, and that playing one or the other will depend on the stage of IS development.

It may be seen that although a model may play either role, one will always prevail over the other. Furthermore, it will be reinforced through the definition of the model type. We have termed this as the primary, or default, role.

Table 2 shows a synopsis of the models studied, with the roles it can play and the default model for each type.

5 Conclusions

Although the roles that a model can play within an IS are very different, they become confused and the differences become blurred. Our aim in this article has been to create a clear conceptual framework of the terms most widely used in ISs to attribute a role to each type of model. To this end, we present an ontology on the concept of model.

Bearing in mind that *model* is one of the terms most widely used in IS development, the construction of an ontology on models is a great challenge, implying working with a concept of

model accepted by the whole community. In this ontology, we have sought to clarify both the concept of model and other similar concepts, which are sometimes confusing. In this debate we have included the concepts of schema, diagram, ontology, metamodel, pattern and architecture.

First, a discussion was made concerning each term, specifying the meaning of each one and the branches they are used in. On the basis of these definitions, their roles within ISs were observed, which enabled us to conclude that it is impossible to attribute a single role to each model type. Aside from the uses assigned to each type of model, it is possible to establish a default role for each one. This role is the one that a model may be associated with according to its meaning in the field of ISs. This study has established that the default role of schemata, diagrams, ontologies and architectures is one of *models as copies*. Within the meaning of these models is the idea that they are representations of a domain.

On the other hand, the default role of metamodels, architectural styles, patterns and process models is that of *models as originals*, that is, they are taken as 'realities' serving as examples.

Studies like this one require the consensus of the whole IS community for the approval and generalization of the correct use of terms, despite which we consider that ontologies fuel more theoretical debates, which lead ISs to create a solid conceptual basis that this field of computing can depend on. We hope that our study contributes to the creation of these new discussion forums.

Acknowledgments

This study was financed partly by the Spanish Ministry of Science and Technology under project GOLD TIN2005-0010, Rey Juan Carlos University and the Government of the Madrid region under projects FoMDAs (URJC-CM-2006-CET-0387), IASOMM (URJC-CM-2007-CET-1555) and M-DOS (URJC-CM-2007-CET-1607) and by the Spanish project 'Agreement Technologies' (CONSOLIDER CSD2007-0022, INGENIO 2010). We thank Carlos Cuesta Quintero for his valuable suggestions and comments. We also acknowledge the clarifying comments raised by the anonymous referees.

References

Alexander, C. 1979. *The Timeless Way of Building*. Oxford University Press.

Atkinson, C. & Kühne, T. 2003. Model-driven development: a metamodeling foundation. *IEEE Software* **20**(5), 36–41.

Bézivin, J. 2005. On the unification power of models. *Software and System Modeling* **4**(2), 171–188.

Booch, G., Rumbaugh, J. & Jacobson, I. 1998. *The Unified Modeling Language User Guide*. Addison-Wesley.

Cañete Valdeón, J. M., Galán Morillo, J. G. & Toro, M. 2004. Some problems of current modelling languages that obstruct to obtain models as instruments. In *Proceedings of the IX Spanish Conference on Software Engineering and Databases (JISBD'2004)*. Málaga, Spain.

Clements, P. C. 1996. A survey of architecture description languages. In *IWSSD '96: Proceedings of the 8th International Workshop on Software Specification and Design*. Schloss Velen, Germany, 16.

Conesa, J., Palop, X. d. & Olivé, A. 2003. Building conceptual schemas by refining general ontologies. In *14th International Workshop on Database and Expert Systems Applications (DEXA'03)*, 693–702.

Cook, S. 2004. Domain-specific modeling and model driven architecture. *MDA Journal*, January, 2–10.

Date, C. J. 2003. *An Introduction to Database Systems*, 8th edn. Addison-Wesley.

De Bruijn, J., Lausen, H., Krummenacher, R., Polleres, A., Predoiu, L. & Kifer, M., *et al*. 2005. *The Web Service Modeling Language (WSML). WSML final draft*. DERI. `http://www.wsmo.org/TR/d16/d16.1/v0.21/`

Dittrich, K. R. 1994. Object-oriented data model concepts. In *Advances in Object-oriented Database Systems. Proceedings of the NATO Advanced Study Institute on Object-oriented Database Systems, 1993*, Dogac, A., Özsu, M. T., Biliris, A. & Sellis, T. (eds). Springer, 30–45.

Elmasri, R. & Navathe, S. B. 1994. *Fundamentals of Database Systems*, 2nd edn. Benjamin/Cummings.

Favré, J. 2004. Foundations of model (driven) (reverse) engineering: Models—episode I: stories of the fidus papyrus and of the solarus. In *Language Engineering for Model-Driven Software Development. Dagstuhl Seminar Proceedings 04101*, Bézivin, J. & Heckel, R. (eds). Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. Available from `http://drops.dagstuhl.de/opus/volltexte/2005/13`

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. 1995. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley.

Genesereth, M. R. & Nilsson, N. J. 1987. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann.

Gómez-Pérez, A., Corcho-García, O. & Fernández-López, M. 2003. *Ontological Engineering*. Springer-Verlag New York, Inc.

Gruber, T. R. 1993. A translation approach to portable ontology specifications. *Knowledge Acquisition* **5**(2), 199–220.

Guarino, N. 1995. Formal ontology, conceptual analysis and knowledge representation. *International Journal of Human-Computers Studies* **43**(5–6), 625–640.

Hornby, A. S. 2000. In *Oxford Advanced Learner's Dictionary of Current English Edited by Sally Wehmeier*, 6th edn., Wehmeier S. (ed.). Oxford University Press.

IEEE. 2005. *IEEE Std 1471-2000—Recommended Practice for Architectural Description of Software-intensive Systems*. http://standards.ieee.org/reading/ieee/std public/description/se/1471-2000 desc.html

Ludewig, J. 2004. Models in software engineering—an introduction. *Software and Systems Modeling* **2**, 5–14.

Marcos, E. & Marcos, A. 2001. A philosophical approach to the concept of data model: is a data model, in fact, a model? *Information Systems Frontiers* **3**(2), 267–274.

McGuinness, D. L. & Harmelen, F. V. 2004. *OWL Web Ontology Language Overview*. http://www.w3.org/TR/owl-features/

Miller, J. & Mukerji, J. 2003. *MDA Guide Version 1.0.1*. http://www.omg.org/docs/omg/03-06-01.pdf

OMG. 2002. The Object Management Group. *Meta-Object Facility (MOF) Specification, version 1.4*. http://www.omg.org/mof/

OMG. 2004. The Object Management Group. Object and Reference Model Subcommittee of the Architecture Board (ORMSC). *A Proposal for an MDA Foundation Model. An ORMSC white paper*. http://www.omg.org/docs/ormsc/05-04-01.pdf

OMG. 2005. The Object Management Group. *Unified Modeling Language: Superstructure, version 2.0*. http://www.omg.org/docs/formal/05-07-04.pdf

OMG. 2006. The Object Management Group. *UML 2 Object Constraint Language Specification*. http://www.omg.org/docs/ptc/03-10-14.pdf

Perry, D. E. & Wolf, A. L. 1992. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes* **17**(4), 40–52.

Sánchez, D. M., Cavero, J. M. & Marcos, E. 2005. An ontology about ontologies and models: a conceptual discussion. Paper presented at the *Proceedings of the ONTOSE 2005. First International Workshop on Ontology, Conceptualizations and Epistemology for Software and Systems Engineering*. Alcalá de Henares, Madrid, Spain, **163**. http://CEUR-WS.org/Vol-163/

Seidewitz, E. 2003. What models mean. *IEEE Software* **20**(5), 26–32.

Shaw, M. & Garlan, D. 1996. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice-Hall.

Smolander, K. 2002. Four metaphors of architecture in software organizations: finding out the meaning of architecture in practice. *ISESE '02: Proceedings of the 2002 International Symposium on Empirical Software Engineering*.

Sommerville, I. 2001. *Software Engineering*, 6th edn. Addison-Wesley.

Studer, R., Benjamins, V. R. & Fensel, D. 1998. Knowledge engineering: principles and methods. *Data & Knowledge Engineering* **25**(1–2), 161–197.

Tetlow, P., Pan, J. Z., Oberle, D., Wallace, E., Uschold, M. & Kendall, E. 2006. *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering*. http://www.w3.org/2001/sw/BestPractices/SE/ODA/

W3C. 2004a. *Extensible Markup Language (XML) Schema*. Architecture Domain. XML Working Group. http://www.w3.org/XML/Schema

W3C. 2004b. *Resource Description Framework (RDF)*. Technology and society domain. http://www.w3.org/RDF/