A Conceptual Approach to Meta–Modelling *

E. Domínguez, M. A. Zapata, J. Rubio

Dpt. de Informática e Ingeniería de Sistemas.
Facultad de Ciencias. Edificio de Matemáticas.
Universidad de Zaragoza. E-50009 – Zaragoza. Spain.
e-mail: ccia@posta.unizar.es

Abstract. In this paper we propose a conceptual–based approach to meta–modelling as a technique in which modelling knowledge can be expressed. Our approach claims to be sufficiently flexible and to homogenize the construction of meta–models, independently of the field of application. Besides, the components of meta–models have been thought up for improving the properties of adaptability, understandability and usability. A meta–model is defined as a perspective, a system of concepts and a so–called conceptual support. A conceptual support is, in essence, an IS–A hierarchy of concepts whose purpose it is to establish the specification elements that can be used to construct models. In order to illustrate the different notions that appear in the paper we present two examples of meta–models (statecharts and data flow diagrams) taken from two different application fields.

1 Introduction

Meta–modelling, as a technique in which modelling knowledge can be expressed, has been independently analysed by some authors. Perhaps CASE–tools is the field in which it has had most applications (see, for instance, [20] and [12]), although meta–modelling has also been studied in other fields such as, for example, Interoperable Information Systems [15] and for comparing formalisms for user interface specification [5].

The techniques used in each of these fields are of a distinct nature since they are guided by different goals. For example, in the field of CASE tools the meta–modelling techniques are used to enable a flexible customization of the methods in CASE [12]. In the case of Interoperable Information Systems, one goal is to facilitate the definition of data model translations [15]. In the third application field that we have indicated as an example, the goal would be to facilitate the comparison of notations regarding, for example, their descriptive power [5].

Despite these differences, in all cases the approach consists in representing modelling knowledge by means of a meta–model, where a meta–model is a conceptual schema of the objects constituting the method, data model or notation, depending on the case. However, quite varied techniques are used to construct a

* This work has been partially supported by the projects CICYT TAP95–0574 and UZ 287–06.

meta–model. Some authors make use of semantic data models as a basis for the meta–models ([19], [17], [22]), whereas others make use of theoretical formalisms ([1], [5]), generalization hierarchies [15] or task structures [22]).

The variety of techniques not only makes the comparison and integration of which the authors propose extremely difficult but it also makes it difficult to obtain the high degree of conectivity between systems required by current technological developments (heterogeneous data base, interoperability, federated and distributed systems, open systems, etc.).

In order to palliate the aforementioned difficulties it will be necessary to develop a meta–modellization technique which should gather and unify aspects abstracted from the particular systems to which they are applied. For this reason, flexible and versatile conceptions and tools must be used so that they can be applied to different fields. With this, as a by–product, a feasible frame for gathering meta–modellization proposals of the literature would be obtained, making way for carrying out analyses and comparisons between them.

In this paper we present an approach which aims to advance in the above–mentioned lines. Specifically, we introduce a conceptual–based framework in which meta–modelling of different tasks can be achieved in a homogeneous way.

This paper is organised as follows. The next section is an overview of our approach explaining the main characteristics of meta–models. In section 3 we define our notion of concept which is used as an epistemological primitive for knowledge representation. The two following sections are aimed at explaining two components of meta–models: the system of concepts and the conceptual support (a key component of our approach). Section 6 introduces our notion of a meta–model together with the definition of a model. In section 7 we discuss related works and, finally, conclusions and plans for future work are presented in section 8.

In order to illustrate the versatility of our proposal practically we will present two examples taken from two different fields: a meta–model of statecharts [7], [6] and a support of Data Flow Diagrams [11].

2 Overview

In the development of our approach we have taken into account the three abstraction levels which are considered in [22]. The least abstract level, called *application level*, is concerned with the development of a specific application. The next level of abstraction, which is called *method level*, is concerned with defining the application level; the models constructed at this level are called meta–models. Lastly, the highest level of abstraction, which is called *axiomatic level*, is concerned with defining the method level. At each level, as an orthogonal dimension, it is necessary to distinguish between the way of modelling (type of models to be used) and the way of working (how the modelling process is structured and carried out).

The conceptual–based framework we present establishes a way of expressing knowledge of the method level so that it is situated at the axiomatic level and

PERSPECTIVE OF STATECHARTS
Statecharts are a visual formalism for specifying the behaviour of complex reactive systems. They extend to classical state transitions principally in three ways: hierarchy, concurrency and broadcast communication. Hierarchy allows a state to be AND/OR–decomposed into several substates. A state which is OR–decomposed into several substates must be in only one of its substates. On the contrary, a state which is AND–decomposed into several substates must be in all of its substates. Transitions from states can be clustered. A transition originating at the boundary of a state applies to all its substates. History states allow a previous visit to a state 'to be remembered'.

Fig. 1. Perspective of the statecharts meta–model

can be viewed as a meta–meta model. In this paper, we will draw our attention to the way of modelling, bearing in mind that a conceptual perspective can also be used to describe the way of working.

Our approach claims to be sufficiently flexible and to homogenize the construction of meta–models at the method level, independently of the field of application. We also consider it essential for the meta–models to satisfy the properties of adaptability, understandability and usability in order to facilitate the work at the application level.

The meta–models must be, on the one hand, customizable [20], in the sense that a meta–model could be adapted to different situations and to particular necessities.

On the other hand, we consider that the proper meta–model should also be easy to use. In the literature the usability is proposed as a property which must be held by the resulting system ([21], [16]). However, we think that it is a demandable property for the actual methods and tools used for modelling a system. In this way, the five usability attributes Nielsen describes in [16] can be reinterpreted in our context and can therefore be desirable properties for the meta–models (easy to learn, efficient to use, easy to remember, few errors and subjectively pleasing). Lastly, since the person that constructs a meta–model (meta–analyst, according to [8]) is different from the person that will use it (analyst [8]), understandability is another property to encourage in the meta–models [13]. In this way, an analyst will be able to understand a meta–model without needing previous specific knowledge.

The several elements of a meta–model, which are described in more detail in the next sections, have been thought up for improving the properties we have just quoted. In order to facilitate the understanding of a concrete meta–model, it must firstly contain a brief description explaining its *perspective*. The description must contain, for example, the application field, a general description of the models that can be constructed by means of the meta–model, their utility, etc. These descriptions are aimed not only at presenting the purpose of the meta–model but also at centering the context so that the analyst can understand the concepts formalized in the system more easily. In figure 1 a simplified perspective of the statecharts meta–model is presented.

A meta–model also contains a *system of concepts* which is composed of all the concepts representing aspects related to the meta–model. The main goal of the system of concepts is to present, as accurately as possible, the meaning of

the elements or components the meta–model provides so that the analyst can use it efficiently and without errors.

The main component of a meta–model is the *support*. In essence, it is an IS–A hierarchy of concepts, previously formalized in the system of concepts, whose purpose it is to establish the specification elements that can be used to construct models. The hierarchized concepts have associated attributes, which satisfy an inheritance property. We consider that the support is a simple enough structure and thus it is easy to understand and easy to use. Besides, a support facilitates the adaptation of a meta–model to different situations thanks to it being quite flexible. In order to adapt a meta–model, several mechanisms can be used such as, for example, refinement (specializing an existing source concept), extension (adding new source concepts), pruning (removing existing concepts) or modification of the attributes associated to the concepts.

Lastly, a meta–model must also contain the definition of the models that can be constructed with the support. This definition will be included, as another concept, in the system of concepts.

In order to facilitate the analyst with the use of a meta–model, it may be convenient for there to exist a document in which the meta–model will be specified (*specification document*, from now on). The suitability of this document is independent of whether there exists or not an automatic tool for the use of the meta–model. As we describe the elements of a meta–model we will also indicate how they must be included in the specification document.

We consider that one of the main features of our proposal is that objects and relationships are treated uniformly in the meta–modelling process. This feature also appears in the Predicator Set Modelling Technique (PSM) [9] and in the Telos knowledge representation language [14]. However, an important difference between PSM and our proposal is that in PSM only entity types can act as subtype, whereas in our approach relationship types also can be hierarchically structured. With regard to Telos, let us notice that, while Telos rests on the single notion of 'proposition', our primitive notion is that of concept. We use the notion of concept because we consider it to be close to human knowledge and therefore it improves the usability and understandability of meta–models.

In order to test the versatility of our framework we have used it for constructing meta–models of formalisms which have been proposed in the literature corresponding to two different application fields: user interface specification and information modelling. To be precise, we have defined meta–models for several user interface specification formalisms (Transition Diagrams, Statecharts, Petri Nets with Objects, etc.) and for two information modelling techniques (Data Flow Diagrams and Entity/Relatioship).

We have also defined a meta–model formalizing our actual meta–meta model. The presentation of this formalization could have been the goal of this paper, however, we have considered it more suitable to present the use of meta–models. We have not even included the support corresponding to the formalization in this paper due to problems of space; not only of the size of its specification but also because it would force us to explain supplementary definitions.

Besides, we have also defined two new meta–models for user interface specification which has been used for developing applications under a contract with Telefónica S. A. and Fundación Formación y Tecnología within projects partially supported by FEDER. The two meta–models were adapted to our particular necessities and were succesfully used for designing and specifying several applications. These experiences have been useful to check that the meta–models are easy to adapt, understand and use.

3 Concept as Epistemological Primitive

Nowadays, there exists a clear trend in Knowledge Modelling which makes use of concepts as epistemological primitives. However, throughout the literature forms of concepts and, consequently, use are varied. The structure by which a concept is presented can be simple, as in the case of Sowa [18], or complex, as in the case of Kangassalo [10], due to the presence of several knowledge components. Our form is closer to Sowa, although with different nuances in order to increase understandability without losing usability.

Following the ideas of Sowa [18], we consider that concepts are mental representations of objects of the application domain (or universe of discourse). However, Sowa specifies a concept by means of a type label and a referent, whereas we represent a concept by using a name and a description (optionally, only one of these elements can be omitted). A *name* is a linguistic object (that is, a grammatically correct expression in written natural language) which is used with the intention of naming what the concept represents. A *description* is a linguistic object which is used with the intention of describing what the concept represents.

So as to facilitate the language, concept will be used to refer to a linguistic representation of a mental representation as well as to the mental representation itself. We make distinctions between *non–named* concept, *non–described* concept and *full* concept, understanding by full concept one which is specified by means of a name and a description. In order to formally specify a non–described concept, a non–named concept or a full concept, the following syntax will be used respectively: ⟨name⟩**, *⟨description⟩* or ⟨name⟩*⟨description⟩*.

The symbol * is considered as a reserved symbol which delimits the description. If the name is composed of several words, the words will be joined by the underscore reserved symbol (_). Several examples of full concept specifications are included in the system of concepts we have constructed of the statecharts (see figure 2).

A full concept must be used when, and only when, the name does not allow it to recognize, without ambiguity, the object represented by the concept. For example, the notion of IS–A hierarchy has been used in literature with different interpretations [4], two of them are:

- IS_A_connection *relation between two generics so that one is less general than the other*

- IS_A_connection *relation between an individual and a generic so that the individual is describable by the general description of the generic*

For this reason in a concrete situation it is necessary to represent the notion of IS–A hierarchy as a full concept in order to determine the exact interpretation used for the concept. In short, each concept must be specified so that what it represents may be recognised without ambiguity.

Like Sowa [18], we differentiate individual concepts from generic concepts. An *individual concept* represents a concrete object in the domain of discourse and a *generic concept* represents a generic element of a category. Besides, we will say a concept C_1 is an *instance* of a generic concept C_2 if C_1 represents an element of the category generically represented by C_2. For example, the specialization relationship existing between the generic concepts 'mammal' and 'animal' is an instance of the 'IS_A_connection' concept we have indicated firstly in the previous example. Let us notice that an instance of a generic concept can be an individual or a generic one.

According to the above, from our perspective the perception of one concept can be found at three different levels of abstraction: the level of ideas, the level of linguistic representation of ideas and, lastly, the level of linguistic representation of the ideas' instances.

4 System of Concepts

The construction of a meta–model requires, among other things, conceptualising the elements perceived in a method or notation. We will call the resulting family of concepts *system of concepts*. Its function, like a dictionary, consists in accurately establishing the different notions an analyst must know in order to use the method or notation. The goal we pursue with this is to make the use and understanding of the meta–models easier.

When the meta–analyst establishes a system of concepts, he assumes that the analyst has a certain previous epistemological knowledge. For this reason, to describe a concept he can make use of *epistemological concepts*, that is, non–described concepts which belong to common knowledge. Starting out from this knowledge, the meta–analyst constructs further concepts which represent new knowledge he considers the analyst must acquire.

In a system of concepts we make a distinction between primitive concepts and derived concepts. A *primitive concept* is based only on epistemological knowledge, whereas a *derived concept* is based on knowledge the analyst acquires trought the system of concepts. A primitive concept can be described or non–described. A non–described primitive concept is an epistemological concept explicitly stated in the system because it is considered an essential component of the meta–model. If a primitive concept is described, the description will be written in epistemological terms. For example, the concept 'reactive_system' is a described primitive concept of the statecharts system of concepts (see figure 2).

A derived concept is always described and the description will contain, besides epistemological concepts, primitive concepts or derived concepts of the

system. In short, from these definitions, it follows that the interpretation of a derived concept is always based on the interpretation of other concepts of the system. This leads us to consider a notion of derivation relationship between concepts of a system. A concept C_1 is *directly derived* from another concept C_2 if the description of C_1 contains the name of C_2. The *relationship of derivation* is the transitive closure of the direct derivation relationship. For example, in figure 2 it can be observed that the concept 'hierarchy_relation' is derived from the concept 'reactive_system'. The derivation relationship suggests to the analyst a suitable order in which he could acquire new knowledge. For example, it is convenient to try to understand the concept 'reactive_system' before trying to understand 'hierarchy_relation'. This is precisely the criterion we have used to sort out the system of concepts in figure 2.

In order to facilitate the analyst with the acquisition of knowledge, circularities (two concepts simultaneously derived from each other) must be avoided in a system of concepts. However, in order to increase the understandability, we consider it convenient to admit circularities in certain situations. This is the case of definitions which are, in essence, recursive. In any case, it is necessary to make sure that the concepts involved in the circularity can be correctly interpreted. For example, we consider that the recursive concept 'expression *number, variable or algebraic operation between expressions*' is correctly described since it allows us to simply and accurately determine whether something is an expression or not.

Let us notice that a system of concepts can be seen as a knowledge base, from which automatic inferences can be made, within a knowledge representation system. Along this line, we have developed a prototype, using Lisp as programming language, in which the derivation graph of a system of concepts is determined in a semi–automatic way. By means of this graph the prototype can automatically deduce several properties of the system. For example, among other things, it calculates the primitive concepts and the derived ones, it determines if there exists some circularity, and it calculates the concepts from which a given concept is derived.

A system of concepts can also contain *principles* determining the behaviour of the concepts. A principle is generally a non–named concept which establishes a condition. We will use principles basically to specify constraints regarding the instances of the generic concepts. A principle can be associated with a concept or can be related to several concepts. For example, in figure 2 the 'statechart' concept has five associated principles, one of which establishes that two different states of a statechart cannot have the same set of substates.

In figure 2 we have specified the principles by means of natural language with the aim of facilitating their interpretation. At present we are defining a formal language for this specification.

A system of concepts is specified as a list of concepts, each one of which is indicated following the syntax previously given. The concepts can be ordered in two ways: alphabetically or in such a way that, for each concept, all the concepts from which it is derived are specified first (except in the case of circularity). This

SYSTEM OF CONCEPTS	source: [7]
reactive_system *system characterized by being, to a large extent, event–driven, continuously having to react to external and internal stimuli*	pag. 54
state *state of a reactive system*	interp.
hierarchy_relation *relation between a state s and a set of states so that the set of states represents a decomposition of s*	[6] p 233 func. ρ p 59
s_1*_is_substate_of_*s_2 *state s_1 with other states represent a decomposition of a state s_2*	[6] p 233
s_1*_is_descendent_of_*s_2 *state s_1 is a substate of a state s_2 or s_1 is a substate of a state which is a substate of another state, and so on until finding, after a finite number of steps, a state which is a substate of s_2*	func. ρ^+ p 59
s_1*_is_superstate_of_*s_2 *state s_2 is a substate of a state s_1*	interp.
s_1*_is_ancestor_of_*s_2 *state s_1 is a superstate of a state s_2 or s_1 is a superstate of a state which is a superstate of another state, and so on until finding, after a finite number of steps, a state which is a superstate of s_2*	interp.
or_state *state which is a superstate of others states, so that, when the reactive_system is in the state it is in only one of the substates of the state*	p 59, 1.a
and_state *state which is a superstate of other states, so that, when the reactive_system is in the state it is simultaneously in all the substates of the state*	p 59, 1.a
root *state which is not a substate of any other state*	p 59, 1.a
basic *state without substates*	p 60, 2.a
history *entity that represents the substate most recently visited of an or_state*	[6] p 238
history_state_relation *relation between an or_state and a history so that the history represents the substate most recently visited of that or_state *	func. γ p 59
history_is_contained_in_state *history is history state related with a state*	interp.
default_relation *relation between a state and a set of entities which are descendants of the state or they are histories contained in the state or in the descendents of the state, so that, when the system is in the state it is in some of these entities unless otherwise specified*	[6] p 235 func. δ p 59
expression *number, variable, $cr(v)$ meaning current value of expression v or $op(v_1, v_2)$ meaning algebraic operation between expressions v_1 and v_2*	p 59, 1.b int. p 56, 3
condition *T standing for true, F standing for false, primitive condition, $in(s)$ meaning to be in a state s, $ny(e)$ meaning not to have been yet an event e, uRv meaning comparison between expressions u and v being R a symbol of the set $\{=, \langle, \rangle, \neq, \leq, \geq\}$, $cr(c)$ meaning current value of condition c, $c_1 \wedge c_2$ meaning conjunction of conditions c_1 and c_2, $c_1 \vee c_2$ meaning disjunction of conditions c_1 and c_2 or $\neg c_1$ meaning negation of condition c_1*	p 59, 1.c int. p 56, 3
atomic_event *λ meaning null event, primitive event, $tr(c)$ meaning change from false to true of condition c, $fs(c)$ meaning change from true to false of condition c, $ch(v)$ meaning change of value of expression v, $en(s)$ meaning enter to state s or $ex(s)$ meaning exit from state s*	p 59, 1.d int. p 56, 3
event *atomic event, $e_1 \wedge e_2$ meaning conjunction of events e_1 and e_2, $e_1 \vee e_2$ meaning disjunction of events e_1 and e_2 or $e[c]$ meaning event e that occurs while condition c is true*	p 59, 1.e int. p 56, 3
atomic_action *μ meaning null action, $c := d$ meaning assigning the value of condition d to primitive condition c or $v := u$ meaning assigning the value of expression u to variable v*	p 59, 1.f int. p 56, 3
action *atomic action or $a_0; a_1; \ldots; a_n$ meaning sequence of actions a_i ($i = 0, \ldots, n$)*	int. p 59, 1.f
transition *change from one state to another triggered by an event and which produces the execution of an action*	int., p 59
source *state from which a transition occurs*	p 59, 1.g
target *state to which a transition leads or history that represents a state to which a transition leads*	p 59, 1.g
states_related_with_transition *set formed by the source states of a transition, by the states which are a target of the transition and by the states history state related with the histories which are a target of the transition*	interp. func. ω p 59
statechart *set of states, set of histories, set of transitions, set of hierarchy_relations, set of history_state_relations and set of default_relations*	int., p 59
Principle 1. Two different states cannot have the same set of substates	
Principle 2. A statechart has only one root	
Principle 3. The set of initial states of a statechart is a maximal state configuration of the root state	
Principle 4. A state cannot have two histories associated to it	
Principle 5. Only an or_state may have a history associated to it	

Fig. 2. System of concepts of the statecharts meta–model

second criterion improves the learnability of the meta–model and at the same time enhances the understandability of the concepts. As we have said previously, in figure 2 we have used this criterion. The complete specification document of a meta–model should contain the two access ways to the concepts.

The process of constructing a system of concepts is not easy. If the meta–model corresponds to a method or notation which exists in the literature, then the meta–analyst should carry out an interpretation of what the authors propose in their papers. In this case it is advisable for each concept and principle to be specified together with a reference to the place from which it has been extracted: paper, page, section, etc. (see figure 2). This information would allow a control of quality to be carried out.

5 Conceptual Support

A *conceptual support*, or simply a *support*, is a family of named generic concepts, which are contained in the system of concepts. These concepts are related to each other by specializations and attributions. The attributed concepts represent the components which can be perceived in a model (for example, states, transitions, hierarchy relations, etc.), whereas the attributive concepts represent the aspects we can or must use to specify these components.

The attributed concepts are related to each other by specializations constituting an IS–A hierarchy of concepts (that is, a directed acyclic graph with only one sink, where *sink* is a vertex with a zero external degree). Besides, the next principle must hold:

Total specialization principle: Each instance of a non–source concept C in the support (that is, C is a vertex with a non–zero internal degree) is an instance of some specialization of C.

Let us notice that, in the literature, the total specialization principle is not usually imposed (nor is any similar one) to the IS–A hierarchies (see, for example, [4], [3], [2] or [22]). However, we think that this principle increases the understandability of the structure and facilitates its management. Besides, this principle allows an IS–A hierarchy to be perceived as a classification (not necessarily disjoint), at several levels, of the instances represented by the source concepts.

In our opinion, this is a main feature of our perspective. The components of the models of the application level will be represented only by instances of the source concepts. As for the non–source concepts, they should be understood as an organization of the source concepts. The purpose of these concepts is to improve the usability and, mainly, the adaptability.

The concepts of the IS–A hierarchy are represented graphically by their name inside a rectangle. The specialization relationships are represented by an arrow from the subconcept to the superconcept (see figure 3).

Each concept of the IS–A hierarchy can be attributed by means of other concepts. These attributes, as we have said before, represent the specification elements of a model. For example, the concept 'entity' of the support in figure 3

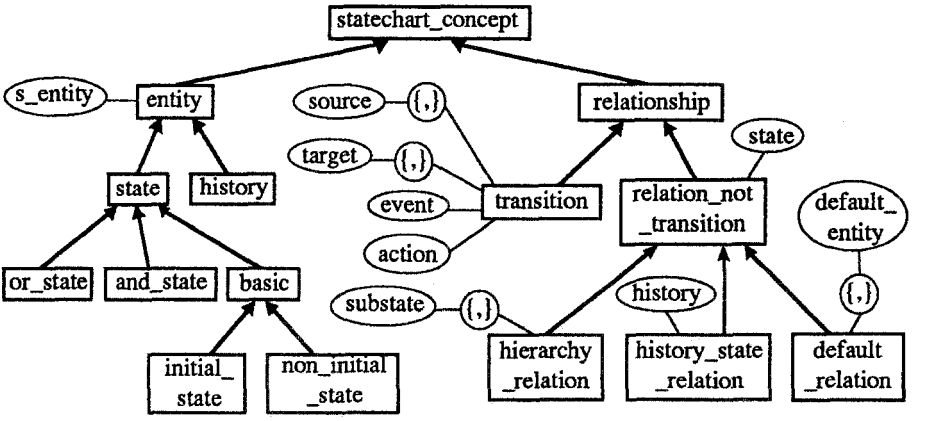

Fig. 3. Support of the statecharts meta–model

has the attribute 'symbol_of_entity' associated to it ('symbol_of_entity' appears abbreviated as 's_entity') which represents that each entity of a statechart must have an associated symbol, this symbol identifies the entity. In this example, the property generically represented by the attribute is simple and can be expressed by means of only one generic concept. Besides, each instance of the concept 'entity' has only one instance of the attribute 's_entity' associated to it. This type of attribute will be called *simple*, the relationship of attribution is said to be *univalued*. The simple attributes are represented graphically by means of an ellipse, and the relationship of attribution by means of a solid line from the representation of the attribute to the representation of the attributed concept.

On the contrary, that is, if the property is complex and it is necessary to use some type of structure and, perhaps, several related generic concepts to represent it, the attributes will be called *complex*. For example, a complex attribute is necessary to represent that, in a statechart, a transition is described, among other things, by the set of symbols associated to its source states. In this case, the attribute is defined by means of a set structure (see support of figure 3). Generally, mathematical structures such as sets, ordered sets, tuples or multisets can be used, even semantic structures, like simple entity/relationship schemata. For simplicity, we will not give any further details of this type of attributes.

The attributes satisfy an inheritance property according to which the attributes are considered to be inherited by all concepts more specialized than the ones they are attached to. Since a support is generally an acyclic graph the inheritance can be multiple, that is, inheritance from several distinct concepts. Let us notice that, although a concept can generalize to two different concepts with the same attribute, the attribute is inherited only once.

The inheritance property provides several well–known benefits [3]. Inheritance allows natural redundancy to be abbreviated reducing the amount of repetitions in descriptions. At the same time, it provides an organization of the information that facilitates its location.

6 Conceptual Meta–Models

We will define a *conceptual meta–model* as a perspective, a system of concepts and a support. Given a meta–model, a model in the application level is made up of instances of the source concepts from the support. Every instance of a source concept must be described through instances of the actual and inherited attributes of the concept. In addition, the model must fulfill the principles included in the system of concepts. It must be pointed out that a meta–model does not include specification rules of the models. Evidently, models must be represented in one way or another, but a clear distinction should be made between the modelling concepts and their external notation [20].

A meta–model determines the modelling concepts, which can have different external notations associated to them. A meta–model together with some specification rules for its models will be called a *communicable meta–model.* Thus, the chosen notation can be adapted to the particular necessities of a work environment and this still does not imply changing the meta–model.

As models include only instances of the source concepts from the support, the definition of a notation basically lies in determining the representation associated with every source concept as well as the specification rules of their attributes.

Sometimes the models represented by means of a meta–model have a dynamic meaning associated to them. For example, the statecharts represent the behaviour of reactive systems, that is, the allowed sequences of system configurations starting from an initial configuration. In these cases, the operational semantics of the models should be able to be specified. For this reason, a meta–model can also contain principles defining the dynamic of the models. The principles of the figure 4 are a simplification of a subset of the principles of the full statecharts meta–model.

7 Related Work

Meta–modelling, as a technique in which modelling knowledge can be expressed [20], has been used mainly in the field of CASE tools. In this field there exist two outstanding proposals: GOPRR conceptual data model [11] (an evolutionary extension of OPRR [17]) and Predicator Set Modelling Technique (PSM) used as a specification technique for representing modelling knowledge ([20], [8]).

Perhaps the main difference between the GOPRR technique and our approach is that his view of the world is based on differentiating types of objects from types of relationships, making different use of each of them. On the contrary, for the purpose of improving the flexibility and uniformity of the meta–modellization process, we do not establish this differentiation since our only primitive notion is that of concept. This uniform treatment can also be found in the Telos knowledge representation language [14]. Telos instead of using a concept–based approach, as we do, rests on the notion of 'proposition'. Making use of concepts we try to enhance the usability of the meta–models.

In the case of PSM, although relationship types are considered a special type of objects, there exists a restriction according to which a relationship cannot

DYNAMIC OF STATECHARTS
Principle.– The initial system configuration is $SC_0 = (X_0, \Pi_0, \theta_0, \xi_0)$ where X_0 is the set of initial states of the statechart, Π_0 is the set of primitive events occurring at the first time interval $[\sigma_0, \sigma_1)$, θ_0 is the set of primitive conditions whose value is true at $[\sigma, \sigma_1)$ for some $\sigma \geq \sigma_0$ and ξ_0 is a function such that for a variable v $\xi_0(v) = x$ if v's value is x in $[\sigma, \sigma_1)$ for some $\sigma \geq \sigma_0$.
Principle.– In order to know the system reaction at an instant σ_i from a system configuration $SC = (X, \Pi, \theta, \xi)$, a step is abstractly divided into microsteps. Given a micro system configuration, the system reaches a new micro system configuration by each microstep.
Principle.– A transition t is *enabled* under a $\mu SC = (\mu X, \mu\Pi, \mu\theta, \mu\xi, \mu Y)$ if t is structurally relevant to the state configuration $\mu X \cup \mu Y$ and the event that triggers the transition occurs at μSC. A microstep from a μSC consists in considering a consistent sequence of transitions where each transition is enabled under μSC.
Principle.– A step from a $SC = (X, \Pi, \theta, \xi)$ consists in considering a sequence of microsteps where the union of transitions considered in the microsteps is a maximal structurally consistent set γ of transitions, that is, the set formed by γ together with any other enabled transition in the last microsystem configuration is not structurally consistent.

Fig. 4. Principles of the dynamic of the statecharts meta-model

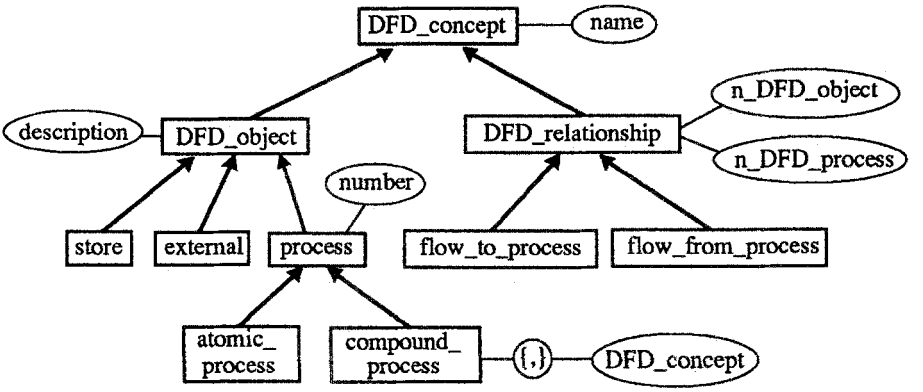

Fig. 5. Support of the Data Flow Diagrams meta-model

be a specialization of another object, that is, relationships always act as 'pater familias' [9]. In our case (just like Telos) both objects and relationships can be hierarchically structured in the support.

The support of a meta-model appears in figure 5. This meta-model has been constructed starting from the GOPRR meta-model of Data Flow Diagrams described in [11]. The GOPRR meta-model contains a hierarchy of DFD_objects and two relationships between DFD_objects and Process (a specialization of DFD_object). Constructing the support of figure 5 we have hierarchized the two relationships giving rise to the hierarchy of DFD_objects. This hierarchy allows a similarity between DFD_objects and DFD_relationships to be expressed: instances of all DFD_concepts must have a name.

Another difference between the two quoted meta-modelling proposals (GOPRR and PSM) and our approach is that they use the notion of 'role' to describe the link between an object and a relationship. We do not use this notion since the role of each concept can be included in its description. In this way we do

not lose expressiveness, since the information can be expressed, and from our point of view the support is more simple and consequently more usable. Perhaps the problem of our proposal is that adaptability and reusability of concepts decreases because we do not separate what the concept represents from the role that it plays in a particular case.

Lastly, the different management that we make of complex objects such as set type, sequence type, and schema type [20] must be pointed out. Whereas the other approaches establish specific mechanisms for each type of complex objects (without a uniform treatment for them), we capture all these structures by means of complex attributes. For example, in figure 5 we have represented a schema type structure by using the attribute 'set_of_DFD_concepts' associated with the concept 'compound_process'. This attribute expresses the fact that a process can be recursively described by means of DFD_concepts.

8 Conclusions and Future Work

Throughout this paper we have described a conceptual–based framework to construct meta–models. Its main characteristics are:

- it claims to be flexible enough to be applied to different fields;
- it encourages the properties of adaptability, understandability and usability in the meta–models.

On the one hand, the flexibility is obtained using the concept as epistemological primitive. On the other hand, we have seen how the elements of meta–models have been thought up for improving the three aforementioned properties.

We plan to extend this work along several lines: definition of a methodology for the construction of meta–models, definition of a formal syntax for the specification of the principles and use of the meta–models to compare different proposals of a given field.

References

1. P. Bergsten, J. Bubenko, R. Dahl, M. R. Gustafson, L. A. Johansson, Ramatic–a CASE shell for Implementation of Specific CASE tools, *SISU*, Stockholm, 1989, first draft of a contribution to section 4.4 of the TEMPORA project.
2. A. Borgida, Knowledge Representation, Semantic Modeling: Similarities and Differences, *E–R Approach'91*, Elsevier Publ., 1991.
3. A. Borgida, J. Mylopoulos, H. K. T. Wong, Generalization/Specialization as the Basis for Software Specification, in M. Brodie, J. Mylopoulos, J. Schmidt (Eds.) *On Conceptual Modeling: Perspectives from Artificial Intelligence, Databases and Programming Languages*, Springer–Verlag,1984.
4. R. J. Brachman, What IS–A is and isn't: an analysis of taxonomic links in semantic networks, *Computer*, v. 16, n. 10, October 1983, 30–36.
5. M. Green, A survey of three dialogue models, *ACM Transaction on Graphics*, 5, 3, 1986, 244–275.
6. D. Harel, Statecharts: A visual formalism for complex systems, *Science of Computer Programming*, vol 8, North–Holland, 1987, 231–274.

7. D. Harel, A. Pnueli, J. P. Schmidt, R. Sherman, On the Formal Semantics of Statecharts, *Proceedings of the 2nd IEEE Symposium on Logic in Computer Science*, Ithaca, N. Y., June, 22–24, IEEE Press, New York, 1987, 54–64.
8. A. H. M. ter Hofstede, T. F. Verhoef, E. R. Nieuwland, G. M. Wijers, Integrated Specification of Method and Graphic Knowledge, *Proceedings Fourth International Conference on Software Engineering and Knowledge Engineering*, IEEE Comput. Soc. Press, 1992, 307–316.
9. A. H. M. ter Hofstede, Th. P. van der Weide, Expressiveness in conceptual data modelling, *Data and Knowledge Engineering*, 10, 1, February 1993, 65–100.
10. H. Kangassalo, Foundations of conceptual modelling: a theory construction view, in H. Kangassalo, S. Ohsuga, H. Jaakkola (Eds.), *Information Modelling and Knowledge bases: concepts, methods and systems*, IOS Press, 1990.
11. S. Kelly, K. Lyytinen, M. Rossi, MetaEdit+: A fully configurable multi–user and multi–tool CASE and CAME environment, in P. Constantopoulos, J. Mylopoulos, Y. Vassiliou (Eds.), *Advanced Information Systems Engineering, Proceedings of the 8th International Conference CAISE'96*, Lecture Notes in Computer Science 1080, Springer, 1996, 1–21.
12. S. Kelly, K. Smolander, Evolution and issues in metaCASE, *Information and Software Technology*, 38, 1996, 261–266.
13. J. A. Larson, *Interactive software: tools for building interactive user interfaces*, Prentice–Hall, 1992.
14. J. Mylopoulos, A. Borgida, M. Jarke, M. Koubarakis, Telos: Representing Knowledge About Information Systems, *ACM Transactions on Information Systems*,8, 4, October 1990, 325–362.
15. C. Nicolle, D. Benslimane, K. Yetongnon, Multi–Data models translations in interoperable Information Systems, in J. Mylopoulos, Y. Vassiliou (Eds.), *Advanced Information Systems Engineering, Proceedings of the 8th International Conference CAISE'96*, Lecture Notes in Computer Science 1080, Springer, 1996, 1–21.
16. J. Nielsen, *Usability Engineering*, AP Professional, 1993.
17. K. Smolander, OPRR–A Model for Modelling Systems Development Methods, in K. Lyytinen, V. P. Tahvanaimen (Eds.), *Next Generation CASE tools*, IOS Press, 1992.
18. J. F. Sowa, *Conceptual structures, Information Processing in Mind and Machine*, Addison–Wesley, 1984.
19. D. Teichroew, P. Macasovic, E. Hershey, Y. Yamamoto, Application of the Entity–relationship approach to information processing systems modeling, in Chen (Ed.), *Entity–relationship Approach to Systems Analysis and Design*, North–Holland, 1980, 15–38.
20. T. F. Verhoef, A. H. M. ter Hofstede, Feasibility of flexible Information Modelling Support, in J. Iivari, K. Lyytinen (Eds.), *Advanced Information Systems Engineering, Proceedings of the 7th International Conference CAISE'95*, Lecture Notes in Computer Science 932, Springer, 1995, 168–185.
21. A. I. Wasserman, P. A. Pircher, D. T. Shewmake, M. L. Kersten, Developing Interactive Information Systems with the User Software Engineering Methodology, *IEEE Transaction on Software Engineering*, 12, 2, 1986, 326–345.
22. G. M. Wijers, A. H. M. ter Hofstede, N. E. van Oosterom, Representation of Information Modelling Knowledge, in K. Lyytinen, V. P. Tahvanaimen (Eds.), *Next Generation CASE tools*, IOS Press, 1992.